Team Artificial Intelligence

Simulation of infantry combat manoeuvres in a game environment

Masters Thesis

Spencer Drayton

MSc Computer Animation

N.C.C.A Bournemouth University

10th September 2007

Table of Contents

List of Illustrations

- "The question of whether computers can think
is like the question of whether submarines can swim."

Edsger W. Dijkstra
1930 - 2002

1 Introduction

In the quest for increased realism and more immersive game play, convincing team artificial intelligence is cited by many industry experts as being one of the next major steps in game development technology. However, it is an area that to date (with a few notable exceptions) remains relatively unexplored, compared with the advances in individual agent artificial intelligence. In the past, team based artificial intelligence techniques may have been discounted as being too computationally expensive, but as the processing power in personal computers and dedicated game consoles continues to increase, there is more spare capacity available now than ever before to support the extra demand for machine cycles that highly complex AI system places upon the host machine.

The objectives of the project described within this thesis were to: 1) design and implement a system for emulating team behaviour in groups of artificially intelligent graphical characters organised in a command hierarchy, and 2) demonstrate the use of this system within a game development environment by constructing a scenario in which a representation of a real-world military tactical manoeuvre is enacted by soldier characters.

Chapter 2 contains a summary of the issues that need to be considered when designing team artificial intelligence and some examples of the approaches taken by others who have confronted these issues in the past.

Chapter 3 introduces the concepts of Hierarchical Finite State Machines (HFSM) and Blackboard architectures with brief technical descriptions. It also contains descriptions of the actual military manoeuvres that were emulated in the final solution.

Chapter 4 details the project solution itself, including the development environment, the third-party software development kits and tools utilised, and descriptions of the key aspects of the finished system. It also describes the development process that was followed, from domain analysis, through design to implementation.

Chapter 5 comprises a discussion of the results obtained from testing of the solution, together with explanations of the some of the technical difficulties that were encountered.

Chapter 6 contains a summary of the conclusions that were drawn after analysing the system test results and outlines some of the areas for potential future development.

The Appendices contain the documented outputs of the analysis, design and implementation stages of the project, including UML design diagrams and source code listings.

2 Previous work

2.1 Background

Game artificial intelligence is a subject which has received a great deal of attention in the recent years, as developers recognise it as a means of enhancing the player's gaming experience. Although artificial intelligence can theoretically be applied to any aspect of a game, one of the most popular applications is the facilitation of autonomous behaviour in software agents, represented visually by graphical characters. Many games involve the player interacting directly or indirectly with other entities which have specific roles within the game environment. Artificial intelligence is used to cause these entities to behave in a way which makes the game more enjoyable for the player, for example through increased realism, increased difficulty or decreased predictability.

Funge [Fu99] describes autonomous agents as those that can "decide how to behave on their own". He also states that to allow this to happen, there exists the need to develop "a computational model of the character's behaviour" and an "explicit representation of some knowledge about [the character's] world". Many of the techniques explored and subsequently employed by the game development industry in the process of addressing these requirements have been borrowed from other disciplines, which have attempted to solve similar problems in different contexts. Examples include cognitive psychology, which attempts to find methods of modelling and representing human perception, understanding and behaviour, and robotics which is concerned with the creation of autonomously controlled mechanical structures.

This report does not include an in-depth discussion of the specific artificial intelligence techniques used to create autonomous agents, but rather an examination of how those techniques can be used to control the behaviour of teams as well as individual agents. In the following sections, some of the issues that need to be considered when designing team artificial intelligence are outlined, and examples of the approaches taken by others who have examined these issues previously are discussed.

2.2 Domain Knowledge

At an individual level, agents need to be able to acquire and record knowledge about their environment, to allow them to make decisions about their next course of action. This is also true at team level - to be able to work towards an objective collaboratively, agents may need to share the knowledge they have obtained with others in their group (*Illustration 1*). For example, Sterren [St02] suggests that an agent engaged in a combat scenario should maintain knowledge of the position, current activity, destination, and enemy sightings for each of its team members. Armed with knowledge obtained from team members, agents can decide on a course of action that contributes to a team objective, rather than being indifferent to it, or even hindering it. Examples include, not heading to a destination that has already been claimed by another team member, or engaging a target that has been spotted by another team member but who is currently unable to engage it himself. The concept can be extended to support the notion of "situational awareness", where an entity responsible for the execution of a high-level mission achieves this objective in part by maintaining a 'picture' of the state of the mission, which is composed of information obtained from individual team members.

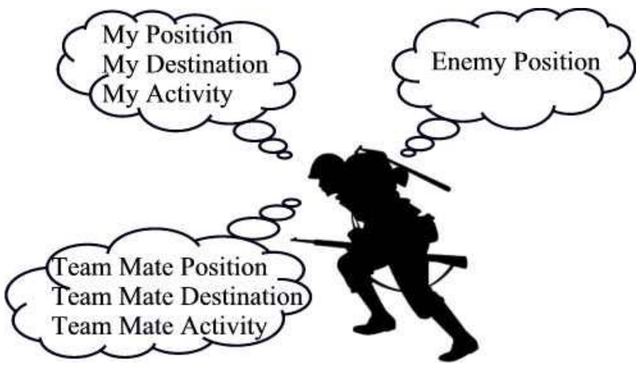


Illustration 1: Agent Domain Knowledge

2.3 Communication

Consideration should be given to the nature of the information communicated between agents, and the manner in which it is communicated. This is a key concept in team artificial intelligence, as without it, an agents 'picture' of its surroundings is limited purely to its own perceptions (*Illustration 2*). However, if lifelike behaviour is required, care needs to be taken to avoid 'perfect' knowledge – where every agent can communicate everything to all other agents without restriction. Sterren [St02] suggests that agent team communications should be implemented through message passing, it facilitates the emulation of the conditions which can occur in real-life communication, such as range limitations, message latency/delay, misinterpretation, or indiscretion (e.g inadvertently broadcasting sensitive tactical information to enemies). A less orthodox approach might be to use the movements of an animated character to convey information. Sastri and Sodiry [SV04] describe the development of a system to recognise the gestures made by aircraft carrier flight deck officers when guiding aircraft on to a landing area. It is conceivable that techniques such as ray tracing might be used in a graphical environment to interpret information encoded in gestures made by a game character in a similar manner.

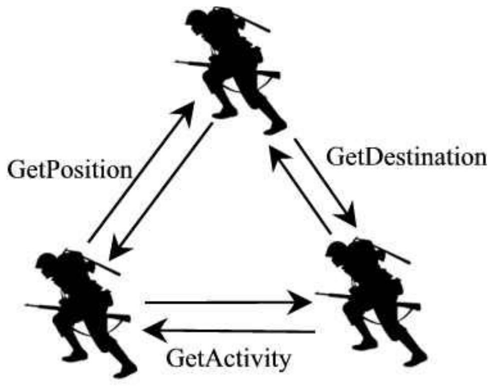


Illustration 2: Inter-agent communication using message passing

2.4 Navigation

In this context, navigation is an extension of the concept of domain knowledge, in that it is concerned with selection of the agent's next destination point, based on its logistical value with respect to the state of the environment at that point in time. Liden [Li02] describes how discrete locations can be classified in a combat scenario according to their tactical usefulness, taking into account the visibility and accessibility of the location (*Illustration 3 – note that inside nodes = "I", outside nodes = "O", ambush nodes = "N" [Li02]*). For example, if a position offers low visibility from an enemy entity and also is adjacent to a position which offers high visibility *of* an enemy, then it might be selected as tactically useful as a point to launch an ambush attack from. Conversely, a point which is located in a room with only one exit into a narrow corridor might be considered as tactically dangerous as it leaves the agent vulnerable to being cornered. Analysis of this kind can be computationally expensive and so may be carried out as a pre-processing step at the game level design stage, however as a point's logistical value may depend in part on in-game conditions, some real-time processing may be unavoidable. An approach which represents a compromise is described by Kamphuis, Rook and Overmars [KRO05], in which the path taken through a corridor by a small team of agents engaged in urban combat is calculated based on the line of sight to an enemy entity, the distance to the enemy entity and the distance from areas providing cover from fire. A generalised 'road map', generated at a pre-processing stage, is augmented with tactical data in real-time as the scenario progresses.

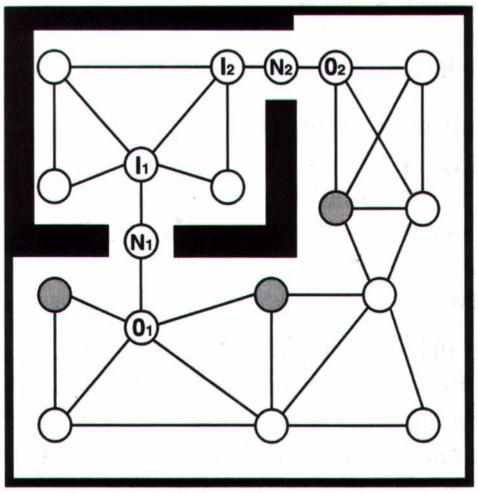


Illustration 3: Navigation using tactical selection of waypoint nodes

2.5 Decision making

Funge [Fu99] suggests that an agent's behaviour arises as a result of selecting and following a set of instructions, based upon knowledge about the current state of its environment, or: *Knowledge + Instructions = Behaviour.* Put simply, behaviour is a description of how an agent acts under a certain set of conditions. It follows that an agents personality can be thought of as a description of how an agent's behaviour changes as environmental conditions change. From this, it can be seen that both the behaviour and personality depend heavily upon how an agent chooses the set of instructions that it will follow next. The decision-making approaches which are used to allow an agent to choose its course of action fall into two broad categories: deterministic and non-deterministic. Funge states that an agent's action selection approach is non-deterministic if it is capable of assessing whether the selection of an action, or set of actions, will result in a favourable outcome with respect to the agents current goal. Deterministic action selection, on the other hand, is pre-defined - the agent does not have the capability to compute the suitability of an action to its current goal. It is furnished with a fixed set of rules which specify precisely which action to select from a finite collection of discrete sets of circumstances. It can be seen from this description how deterministic selection shifts the decision processing overhead from the agent (real time), to the developer (design time) – this is an important consideration to bear in mind when determining the balance of realism and performance required within a game.

2.6 Organisation

Effective team behaviour depends heavily on how the team is organised, that is, where the action selection intelligence is located and how it is distributed. An examination of past and present military manoeuvre theory reveals that organisational approaches fall into two broad categories: centralised and decentralised. Centralised organisation seeks to locate the decision making intelligence in one place within a command entity, away from the entities that are carrying out the manoeuvre. Centralised control leads to teams that tend to be efficient and easy to coordinate, but are poor at adapting to change and are vulnerable to disruption (for example if commander is 'killed'). Decentralised organisation distributes the intelligence between the executive entities and does not have a single, central point of control. Decentralised control leads to teams that achieve high level objectives through emergent behaviour – that is, complex behaviour which is brought about indirectly as the collective result of many simpler behaviours, and which cannot be easily be predicted or deduced from behaviour in the lower-level entities [An72]. Such teams tend to adapt well to changes in their circumstances, but are difficult to control (as a team) and often lack the direct coupling required to reach collaborative agreement on a common course of action. It is interesting to note that the current thinking in organisation of tactical team artificial intelligence in gaming [St02] concurs with that in 'real-life' military theory [Le91].

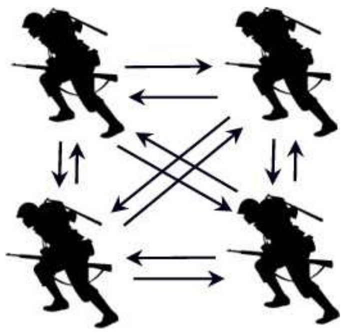


Illustration 4: De-centralised team structure

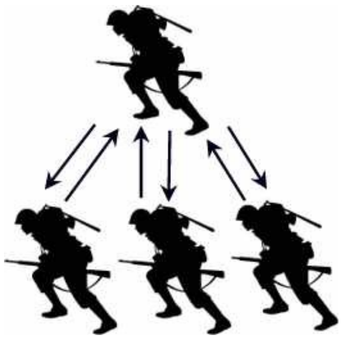


Illustration 5: Centralised team structure

3 Technical background

3.1 Hierarchical Finite State Machines

Finite State Machines are models of the behaviour of a system or a complex object, with a limited number of defined modes or states, where transitions between states occur in response to changes in circumstance [Br03] *(Illustration 6[Ou99])*. They have applications in a wide range of areas, but have been used for many years by game designers as a means of implementing artificial intelligence. In this context, the behaviour of a game agent is decomposed into a discrete set of activities [Bu05]. The state machine is then used to cause the agent to switch from one activity to another when certain conditions are found to be true.

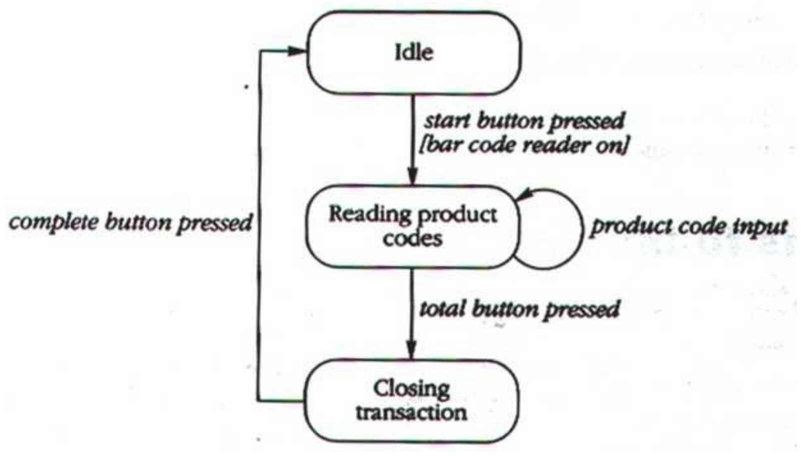


Illustration 6: Example of a Finite State Machine (FSM)

Hierarchical Finite State Machines support the notion of nesting state machines *(Illustration 6[Ou99])*. A behaviour defined using a finite state machine at a low level may be represented as a single state within a higher level state machine arranged in a hierarchy. This concept is particularly useful when considered in the context of team artificial intelligence as it can be exploited to mimic the command hierarchy within a team of agents.

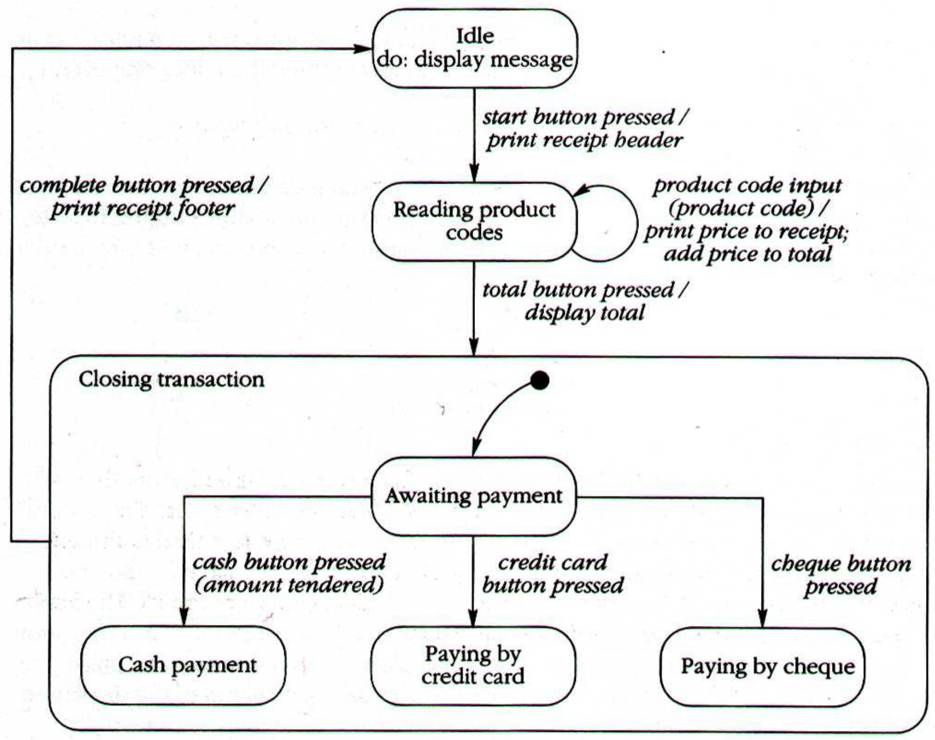


Illustration 7: Example of a Hierarchical Finite State Machine (HFSM)

3.1.1 Advantages of Finite State Machines

- Easy to design – the process of translating a behaviour expressed in an abstract form into formal software design is relatively straightforward as approaches such as UML contain notations which support the common concepts of states, events, conditions and actions.
- Easy to implement – most programming languages contain constructs which support the construction of simple state machines in software, such as switch/case, if-then-else statements. Object oriented languages such as C++ also support inheritance, method overloading and function pointers which can be used to construct finite state machines which avoid the use of relatively inefficient logic statements.
- Easy to debug – as agents state under a given set of circumstances can easily be deduced, debugging effort can be directed to the relevant area of source code quickly.
- Efficient – finite state machines have very little computational overhead.
- Intuitive – finite state machines express behavior in a manner which humans find easy to understand.
- Flexible – finite state machines are easy to adapt to changes in purpose and scale [Bu05].

3.1.2 Disadvantages of Finite State Machines

- Finite state machines are not suited to situations where a system's behaviour cannot be easily decomposed into discrete states.
- The behaviour resulting from the application of a finite state machine can seem predictable, if the implementation does not specifically recognise this drawback and counter it

accordingly. One solution to this problem is to introduce uncertainty into the state change conditions (for example using fuzzy logic) to achieve non-deterministic behaviour.
- Maintenance of large, complex systems implemented using a finite state machine can be problematic without a well thought out (and well documented) design to refer to [Br03].

The project described in this thesis utilises an event-driven, hierarchical state machine to manage both agent and team behaviour.

3.2 Blackboard architectures

The blackboard model was originally conceived as a means to handle complex, ill-defined problems. It is analogous to the blackboard found in classrooms, in that it is used as a shared repository of information by a number of entities which contribute information to it and process the information upon it *(Illustration 8, Note that "KS" = Knowledge Source [IB02]).* The theory supporting the concept suggests that blackboard systems are a means of efficiently pooling resources to collaboratively solve a common problem. In general a typical blackboard architecture consists of three main components [Co91]:

- The 'blackboard' itself: a shared repository of problems, partial solutions, suggestions, and contributed information.
- The 'knowledge sources', which provide specific expertise needed to solve the common problem.
- The control mechanism, which controls the flow of problem-solving activity in the system.

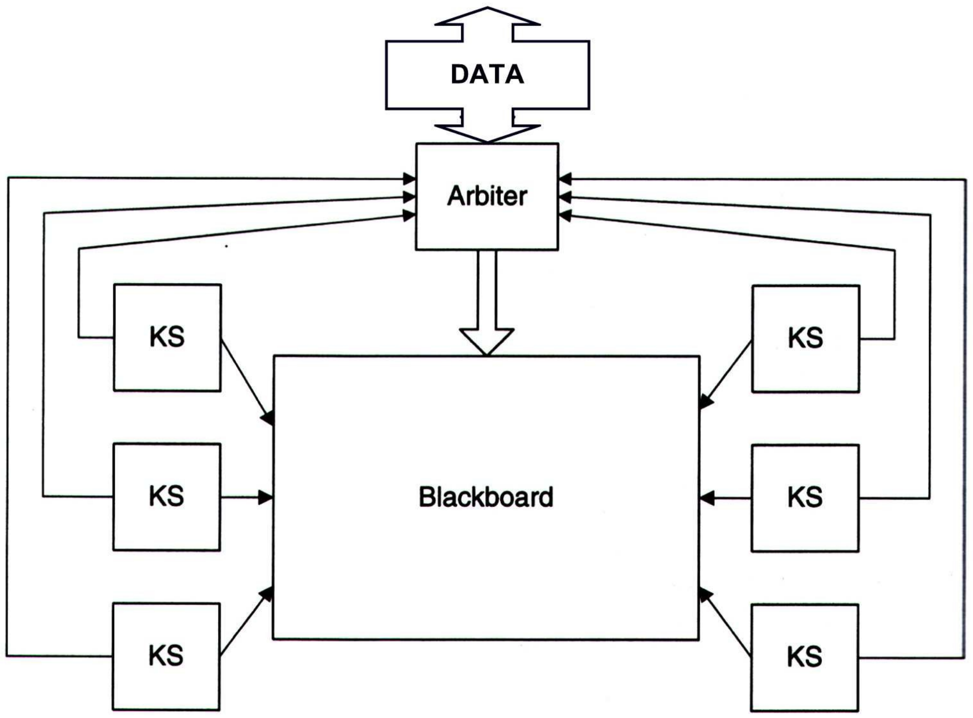


Illustration 8: Representation of a typical Blackboard architecture

Orkin [Or03] suggests that the power of blackboard architectures comes from the ability to support interaction and cooperation among the knowledge sources within the system allows for a great deal of flexibility at the design and maintenance phases of the development of a game application. Isla and Blumberg [IB02] claim that the ability to impose strict control over inter-agent communication is where the strength of the blackboard architecture lies. Whilst it is possible to concur with both of these view points, it should be noted that as the architecture permits for universal information sharing, it represents a stark contrast to the key notions of the object-oriented software development paradigm, namely information hiding and data encapsulation. It should be born in mind however, than object orientation was conceived principally to assist the process software *development,* as it facilitates concepts such as re-use, inheritance and polymorphism. These concepts, whilst they have undoubtedly revolutionised the software development process are often at odds with the need to maximise performance and minimise memory footprints and CPU loads. Orkin [Or03] argues that in game design, "object orientation is not always the right choice", explaining that actively seeking to decouple game data from game behaviour facilitates easier design changes, easier re-use and easier maintenance. Perhaps the answer is not to re-think the design approach, but to reconsider the analysis approach. True object-oriented analysis fully supports the notion of objects which have no 'physical' counterpart; for example in a bookkeeping system a 'transaction' can feasibly be modelled as an object. It is suggested that modelling the blackboard entity as an object itself may be the way to reach a compromise between the two approaches.

The project described in this thesis utilises a blackboard object which acts a shared, managed repository of information used by all agents within the system.

3.3 Military Combat Manoeuvres

3.3.1 The Boyd or "OODA" Cycle

Military strategist Col. John Boyd devised a decision making process, called the "Boyd", or "OODA loop", intended to be employed in combat as a means of out-manoeuvring an opponent *(Illustration 9[Bo96])*. "OODA" stands for Observation, Orientation, Decision, Action and describes a set of states (and their associated actions) which are repeated indefinitely whilst in combat, with the aim of defeating an opponent. Boyd suggested that the faster this cycle is processed, the greater the advantage gained over the enemy [Le91],[Bo96].

While the approach seems at face value very directed and inflexible, the key idea is that this direction will lead to manoeuvring that is highly efficient and hence very rapid in relation to the opponents speed of manoeuvre. The speed difference created has the double advantage that it not only renders the opponent incapable of predicting the combatants next move, but also allows the combatant more time to to clarify and predict the opponents intentions.

The project described in this thesis makes use of the "OODA cycle" concept in the design of its agents behaviours.

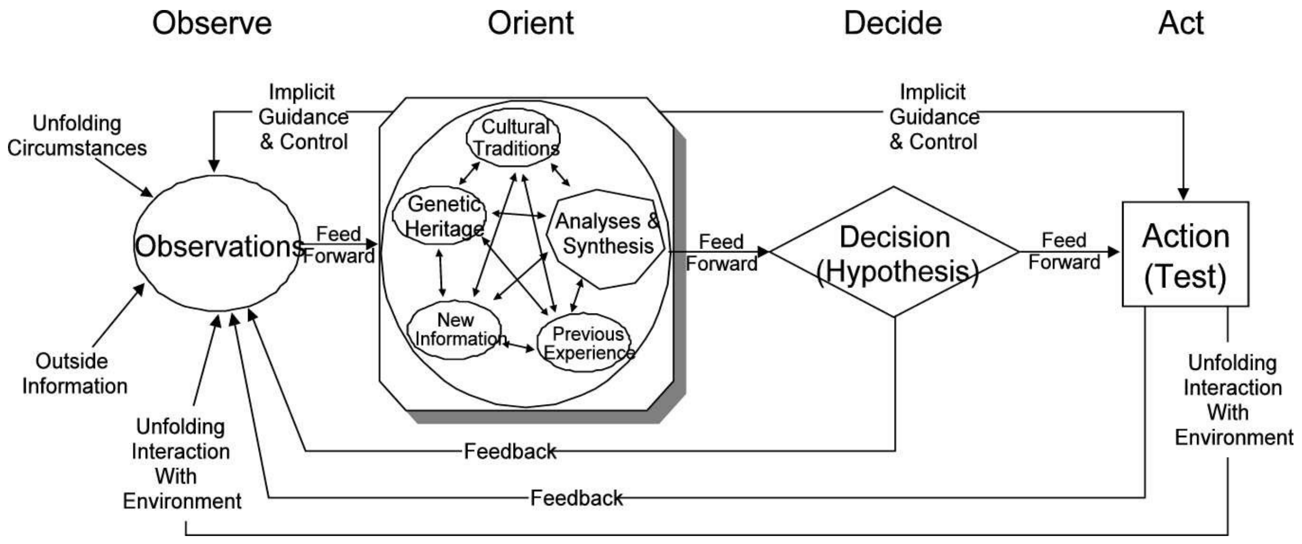


Illustration 9: The Boyd, or "OODA" loop

3.3.2 "Fire and Manoeuvre" or "Bounded Overwatch"

Fire and manoeuvre is the description given to the process of moving combatants in teams towards an objective whilst under fire *(Illustration 10 [La05])*. One combatant (or group of combatants) remains at a cover position and provides suppressing fire to prevent the enemy from issuing returning fire, whilst the other group moves rapidly to a new position further up the battlefield [La05]. Once under cover and with line of sight to the enemy, the process is repeated with reversed roles, with those previously providing cover now taking their turn to leap-frog their team members and move up the battlefield. This process is repeated until the objective is reached.

Illustration 10: "Fire and Manoeuvre" being used by troops to advance under fire

This concept has been implemented in the project described by this thesis in an attempt to simulate team behaviour.

3.3.3 Offensive Manoeuvres – The Assault

An examination of a number of military texts which focus on the theory, practice and history associated with principles of operation of armies past and present around the world has revealed that the success of a mission depends on, amongst other factors, selection of correct manoeuvre for the mission to be executed. The English military historian Sir Basil Henry Liddell Hart wrote that manoeuvre theory "seeks to defeat enemy by (1) avoiding enemy strengths (ie preventing them from bringing those strengths to bear) (2) deceiving the enemy (again preventing them from bringing their strengths to bear) (3) creating and attacking enemy vulnerabilities (physical and/or psychological)" [Le91]. This accords with the writings of the 6th century B.C. Chinese General Sun Tzu, who asserted in his book of military strategy theory "The Art of War", that: "*...to subdue the enemy without fighting is the acme of skill* ".

Many different manoeuvre strategies have been developed with the aim of implementing these and countless other tactical doctrines. One such manoeuvre, known simply as the "Assault" is described briefly here and was used to demonstrate team behaviour in the project associated with this thesis.

The Assault manoeuvre is intended to be used against an enemy that is static and firmly established in a defensive line. Rather than embark on a head to head battle along line parallel with the defensive line, the intention is identify a weakness in the defensive line, then use distraction or suppression techniques to occupy the opposing forces while a separate small team breaches the line at the weak point, moves through it and attacks the enemy from behind their own forward line of troops *(Illustration 11 [La05]).*

The key stages in this manoeuvre are:

- Occupy the mission starting point ("rally" point)
- Conduct reconnaissance of the enemy position (using a scout or scout team)
- Fix the opposing forces position.
- Identify a breach point.
- Organise the squad into a cover team and a breach team.
- Move the cover team into position to engage targets on enemy defensive line
- While the cover team provides suppressing fire and distraction, the breach team moves to breach point covertly.
- Breach team breaches the enemy forward line at the weak point covertly and neutralises the enemy, using surprise to their advantage.
- Breach team clears the enemy position and all squad members move to the objective point.

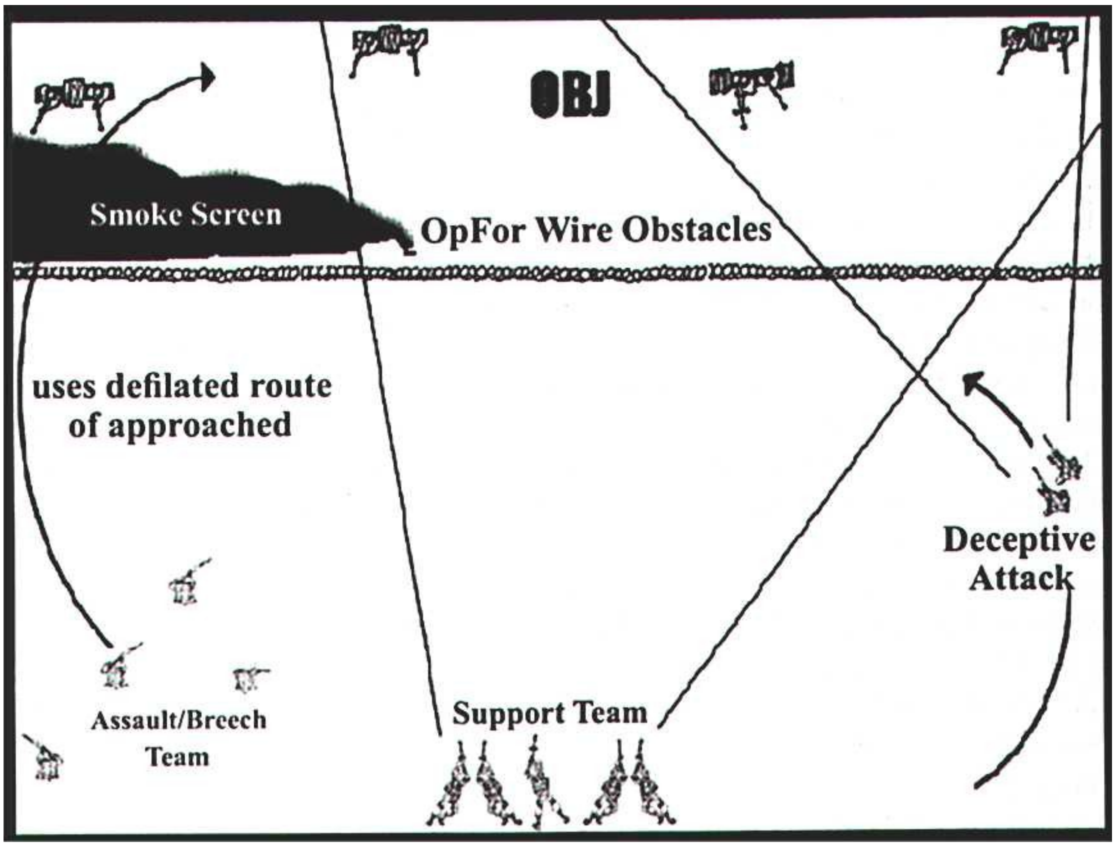


Illustration 11: Position and movement of squad members with respect to opposing forces defensive line during an assault manoeuvre

4 The solution

4.1 Approach

4.1.1 Objectives

The objectives of the project described within this thesis were as follows:

1. To design and implement a system for emulating team behaviour in groups of artificially intelligent graphical characters organised in a command hierarchy.
2. To demonstrate the use of this system within a game development environment by constructing a scenario in which a representation of a real-world military tactical manoeuvre is enacted by soldier characters.

4.1.2 Scope

The project scope encompasses only the artificial intelligence system within the game. This includes technical aspects of the presented solution (analysis, design, implementation of artificial intelligence code) and the completeness and realism of the behaviour of the agents in the demonstration, as viewed from a game player's point if view.

The quality of the character modelling, animation and rendering are outside the scope of the project.

3D graphical manipulation (collision detection, pathfinding) were not originally intended to be part of the project scope, but it was ultimately discovered that some work in this area was necessary to overcome some of the problems that were encountered in addressing the stated objectives.

To provide a demonstration of the system within a real gaming environment, it was decided that the project would be implemented as a modification to an existing, commercially available game application, rather than a standalone application developed from scratch. This decision was reached after giving consideration to the balance required between the ease of implementation of the solution and the visual impact of the demonstration. It was felt that although choosing to follow the commercial game modification route would be more of implementation challenge, the ability to exploit other sub-systems within the game would lead to a solution that was more complete and easier to assess visually.

4.2 Details of selected development system

4.2.1 Description

The development system chosen for this project was the software development kit (SDK) for the first person shooter personal computer game “Far Cry” [Fc04] *(Illustration 12).* The game was published in 2004 by Ubisoft Entertainment [UBI]. It was developed by CryTek GmbH [CRY] and makes use of their game engine “CryEngine1.x”[CRE].

The SDK is open source and was obtained from the official CryTek Far Cry modding portal “CryMod” [CRM]. Distributed with the SDK are tools, libraries and source code and scripts which allow developers to modify and augment many aspects of the game, including the artificial intelligence system.

A brief description of each of the components relevant to this project is given in the following sub-sections, together with an explanation of how it was used within the project.

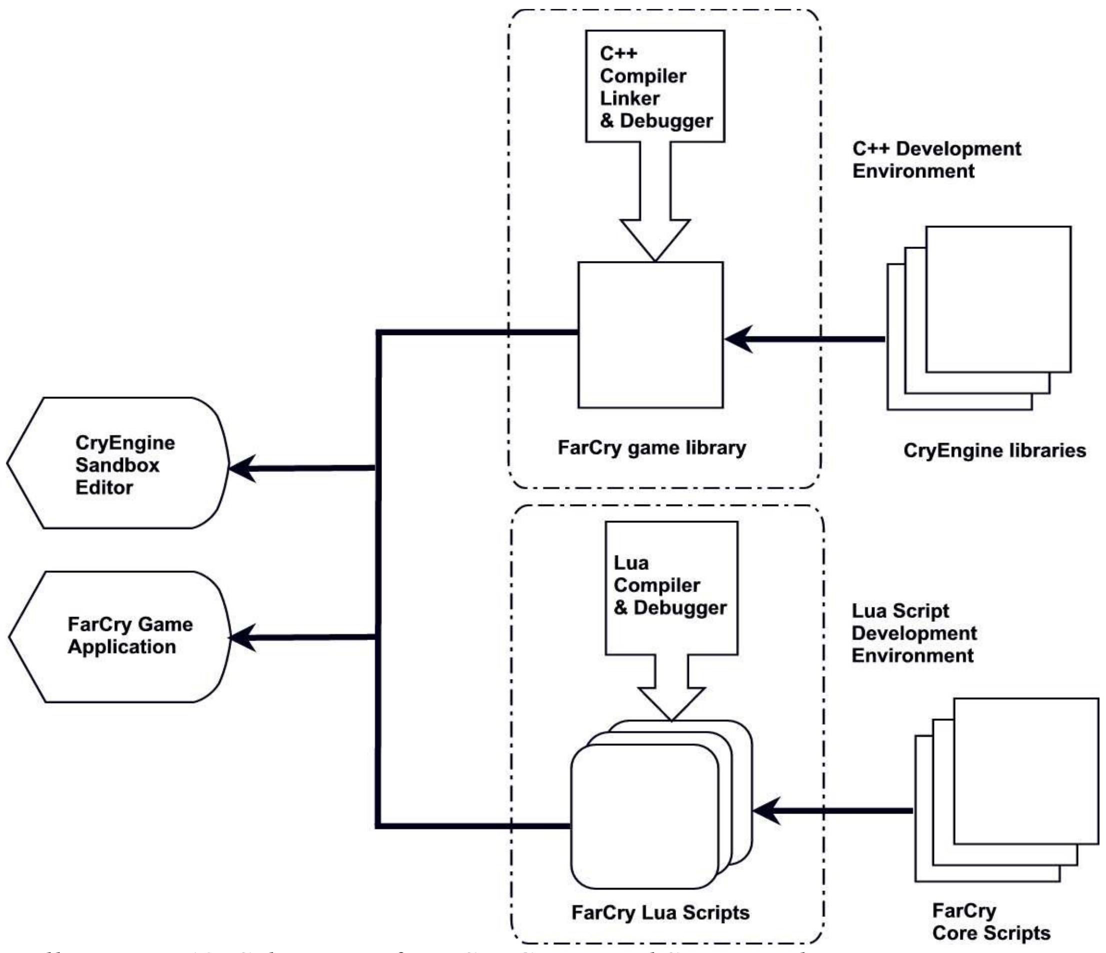
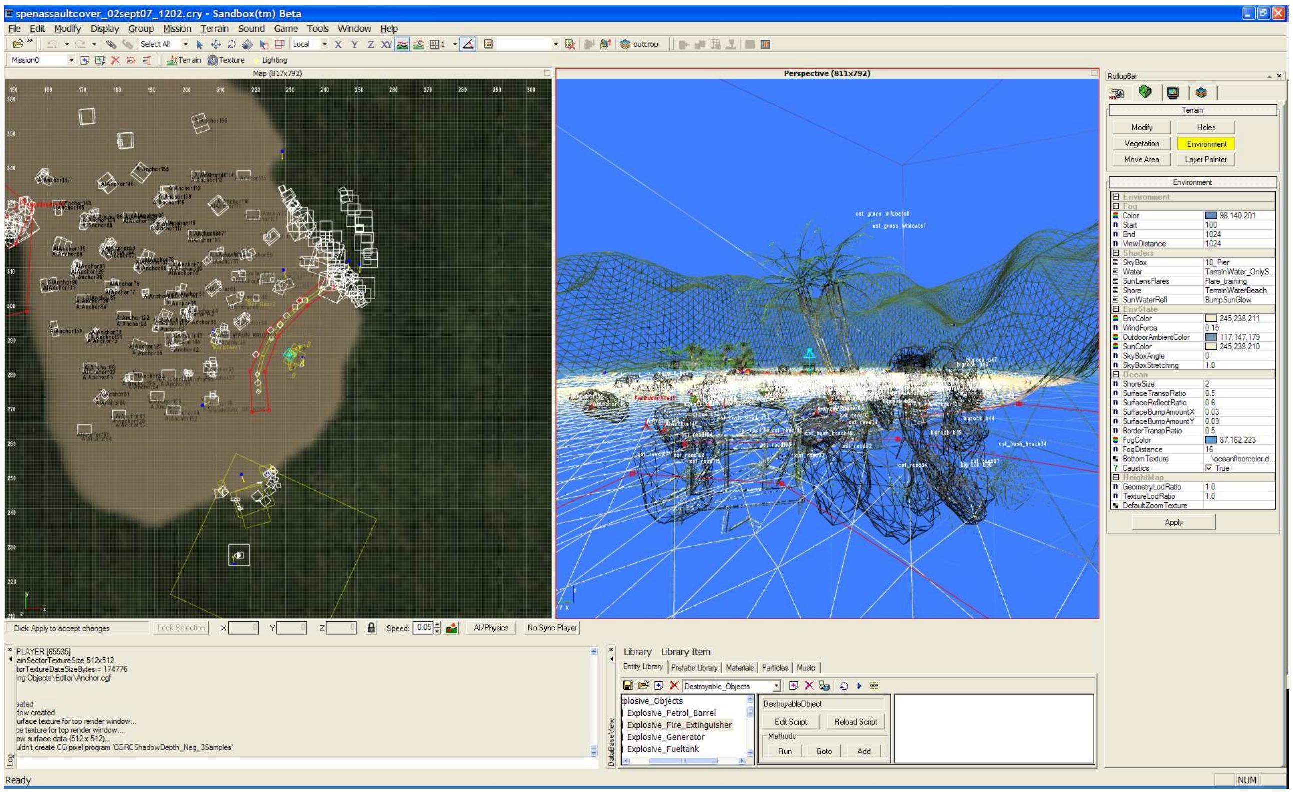


Illustration 12: Schematic of FarCry Game Mod SDK Development Environment

4.2.2 Level editor ("Sandbox")

The "Sandbox" level editor is a stand-alone application with its own graphical user interface *(Illustration 13[CRS])*. Its purpose is to allow the creation of new game levels, which can then be exported to the CryEngine game engine and subsequently used within the Far Cry game itself. The main activities it supports are the generation, texturing and lighting of terrain, the placement of objects and the assignment and modification of attributes to objects [Ab04].



Environment	
Fog	
Color	98,140,201
Start	100
End	1024
ViewDistance	1024
Shaders	
SkyBox	18_Pier
Water	TerrainWater_OnlyS...
SunLensFlares	Flare_training
Shore	TerrainWaterBeach
SunWaterRefl	BumpSunGlow
EnvState	
EnvColor	245,238,211
WindForce	0.15
OutdoorAmbientColor	117,147,179
SunColor	245,238,210
SkyBoxAngle	0
SkyBoxStretching	1.0
Ocean	
ShoreSize	2
SurfaceTranspRatio	0.5
SurfaceReflectRatio	0.6
SurfaceBumpAmountX	0.03
SurfaceBumpAmountY	0.03
BorderTranspRatio	0.5
FogColor	87,162,223
FogDistance	16
BottomTexture	...\oceanfloorcolor.d...
Caustics	✓ True
HeightMap	
GeometryLodRatio	1.0
TextureLodRatio	1.0
DefaultZoomTexture	

Speed: 0.05

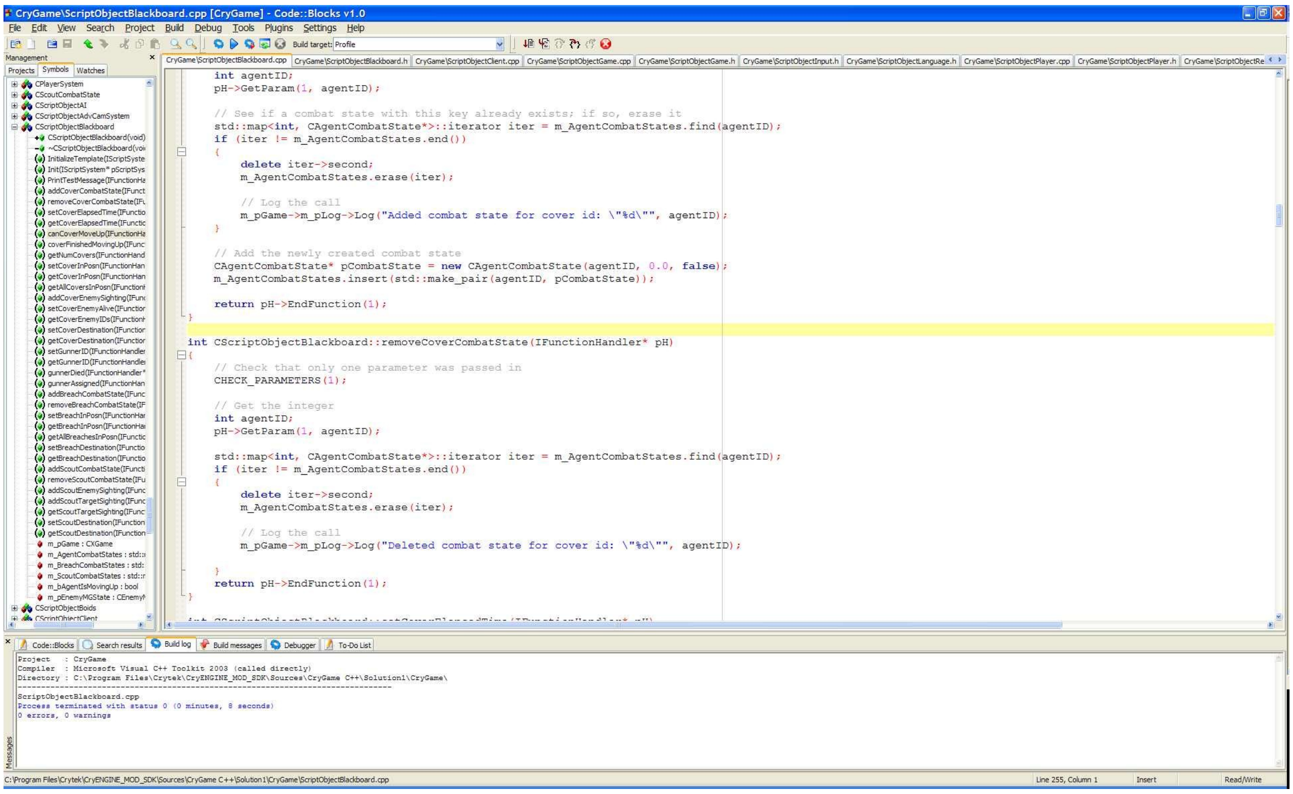
```
PLAYER [65535]
ainSectorTextureSize 512x512
torTextureDataSizeBytes = 174776
ng Objects\Editor\Anchor.cgf


eated
dow created
urface texture for top render window...
ce texture for top render window...
ew surface data (512 x 512)...
uldn't create CG pixel program 'CGRCShadowDepth_Neg_3Samples'
```

Destroyable_Objects
- xplosive_Objects
- Explosive_Petrol_Barrel
- Explosive_Fire_Extinguisher
- Explosive_Generator
- Explosive_Fueltank

Illustration 13: CryEngine1.x Sandbox Editor Screenshot

4.2.3 C++ compiler and libraries

The Far Cry game is made up of a collection of subsystems (sound, animation, physics, 3D etc), each of which is housed within its own shared library. The SDK provides the C++ source code for the “game” subsystem library, which makes use of all the other libraries and contains the Far Cry game-specific functionality. The source can be modified and rebuilt using open source tools and then incorporated into either the installed Far Cry game, or the level editor application [Ab04]. The required compiler maybe called from within a IDE if desired (*Illustration 14 [CBL]*).



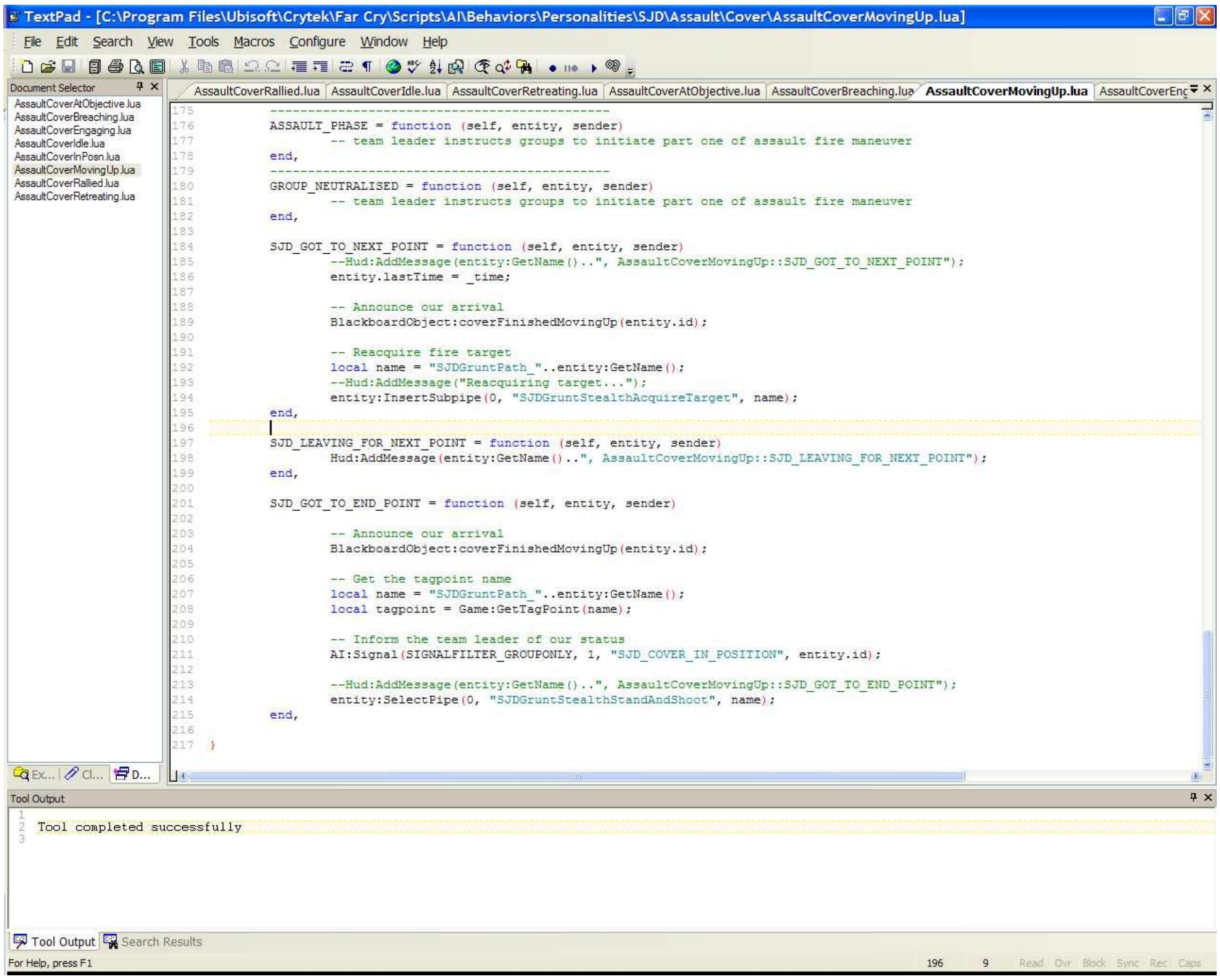
```
    int agentID;
    pH->GetParam(1, agentID);

    // See if a combat state with this key already exists; if so, erase it
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        delete iter->second;
        m_AgentCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Added combat state for cover id: \"%d\"", agentID);
    }

    // Add the newly created combat state
    CAgentCombatState* pCombatState = new CAgentCombatState(agentID, 0.0, false);
    m_AgentCombatStates.insert(std::make_pair(agentID, pCombatState));

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::removeCoverCombatState(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int agentID;
    pH->GetParam(1, agentID);

    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        delete iter->second;
        m_AgentCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Deleted combat state for cover id: \"%d\"", agentID);

    }
    return pH->EndFunction(1);
}
```

```
Project   : CryGame
Compiler  : Microsoft Visual C++ Toolkit 2003 (called directly)
Directory : C:\Program Files\Crytek\CryENGINE_MOD_SDK\Sources\CryGame C++\Solution1\CryGame\
--------------------------------------------------------------------------------
ScriptObjectBlackboard.cpp
Process terminated with status 0 (0 minutes, 8 seconds)
0 errors, 0 warnings
```

Illustration 14: C++ Integrated Development Environment Screenshot

4.2.4 LUA scripting environment

Although the functionality provided by the artificial intelligence subsystem provided via the library interface is accessible but not modifiable, much of the artificial intelligence functionality that is specific to the Far Cry game is scripted in Lua. All the source Lua scripts for the Far Cry game are included in the SDK, as is a Lua compiler which can be combined with a standard text editor to provide a reasonably integrated script development environment (*Illustration 15 [TEX]*). The level editor also contains an embedded Lua debugger, which assists with the process of tracing script problems at run time [Ab04].


TextPad - [C:\Program Files\Ubisoft\Crytek\Far Cry\Scripts\AI\Behaviors\Personalities\SJD\Assault\Cover\AssaultCoverMovingUp.lua]
File Edit Search View Tools Macros Configure Window Help
Document Selector
AssaultCoverAtObjective.lua
AssaultCoverBreaching.lua
AssaultCoverEngaging.lua
AssaultCoverIdle.lua
AssaultCoverInPosn.lua
AssaultCoverMovingUp.lua
AssaultCoverRallied.lua
AssaultCoverRetreating.lua
AssaultCoverRallied.lua
AssaultCoverIdle.lua
AssaultCoverRetreating.lua
AssaultCoverAtObjective.lua
AssaultCoverBreaching.lua
AssaultCoverMovingUp.lua
AssaultCoverEng
Ex...
Cl...
D...
Tool Output
Tool Output
Search Results
For Help, press F1
196
9
Read Ovr Block Sync Rec Caps

```lua
        -------------------------------------------
        ASSAULT_PHASE = function (self, entity, sender)
                -- team leader instructs groups to initiate part one of assault fire maneuver
        end,
        -------------------------------------------
        GROUP_NEUTRALISED = function (self, entity, sender)
                -- team leader instructs groups to initiate part one of assault fire maneuver
        end,

        SJD_GOT_TO_NEXT_POINT = function (self, entity, sender)
                --Hud:AddMessage(entity:GetName()..", AssaultCoverMovingUp::SJD_GOT_TO_NEXT_POINT");
                entity.lastTime = _time;

                -- Announce our arrival
                BlackboardObject:coverFinishedMovingUp(entity.id);

                -- Reacquire fire target
                local name = "SJDGruntPath_"..entity:GetName();
                --Hud:AddMessage("Reacquiring target...");
                entity:InsertSubpipe(0, "SJDGruntStealthAcquireTarget", name);
        end,

        SJD_LEAVING_FOR_NEXT_POINT = function (self, entity, sender)
                Hud:AddMessage(entity:GetName()..", AssaultCoverMovingUp::SJD_LEAVING_FOR_NEXT_POINT");
        end,

        SJD_GOT_TO_END_POINT = function (self, entity, sender)

                -- Announce our arrival
                BlackboardObject:coverFinishedMovingUp(entity.id);

                -- Get the tagpoint name
                local name = "SJDGruntPath_"..entity:GetName();
                local tagpoint = Game:GetTagPoint(name);

                -- Inform the team leader of our status
                AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_COVER_IN_POSITION", entity.id);

                --Hud:AddMessage(entity:GetName()..", AssaultCoverMovingUp::SJD_GOT_TO_END_POINT");
                entity:SelectPipe(0, "SJDGruntStealthStandAndShoot", name);
        end,

}
```

```
1
2 Tool completed successfully
3
```

Illustration 15: Lua Script Development Environment Screenshot

4.2.5 FarCry AI system

4.2.5.1 Architecture

The Far Cry artificial intelligence subsystem is implemented as an event-driven, hierarchical finite state machine (HFSM)[AI04].

4.2.5.2 Entities

Entities in Far Cry are an abstraction of any object which can have artificial intelligence properties associated with. This includes characters, animals, vehicles, weapons and structures [AI04].

4.2.5.3 Perception

Entities are equipped with “vision”, where visibility of other entities is performed by raytracing and bounded by attributes which specify field of view and sight range. They are also equipped with “hearing”, which operates in a similar manner to vision, but without the need for line-of-sight. Modifiers including distance, speed of movement and height from the ground all contribute the rate at which entities perceive each other and other objects around them.

Entities are also endowed with “memory”, in that they are able to remember where an entity was last seen, if it happens to move out of view. Unless reinforced by subsequent sighting, these memories degrade over time, so that an entity do not become invincible, by remembering everything it has ever seen.

4.2.5.4 Behaviour

Entities can exist in exactly one state at a time. In Far Cry, states are referred to as “Behaviours” and are implemented as Lua tables. They are analogous to classes in an object oriented language and as such support indirection – a concept similar in nature to inheritance. Each behaviour script implements event handlers which are designed to respond to a standard collection of game system events, as well as any custom events created by the developer. These event handlers contain the processing which determine how an entity which has been assigned this behaviour will react to the events [AI04].

4.2.5.5 Character

In state machines, the transition from one state to another is triggered by events. In Far Cry, the script which controls the mapping between states is called the “Character” script. Character scripts are implemented as Lua tables and define which behaviour state an entity will switch to upon receiving an event, given its current behaviour state. It can be seen how an entity's personality can be constructed by mapping behaviour scripts together with character scripts [AI04].

4.2.5.6 Events

In Far Cry, “Events” are a collection of messages which are issued by the game engine when certain occurrences take place. Examples of events are “EntitySeen”, “ReceivingDamage”, “Spawn”, “TargetLost”. All behaviour scripts implement event handler callbacks which respond to these events [AI04].

Example of an event handler:

```
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultCoverMovingUp::OnNoTarget");
		-- Reacquire fire target

		local name = "SJDGruntPath_"..entity:GetName();
		Hud:AddMessage("Reacquiring target...");
		entity:InsertSubpipe(0, "SJDGruntStealthAcquireTarget", name);
	end,
```

4.2.5.7 Signals

“Signal” is the name given to custom messages which are created by developers. They are not part of the collection of system events, however signal handlers are implemented in behaviour scripts in exactly the same way as event handlers. The main difference between signals and scripts is that the set of occurrences which trigger game events is fixed. A developer can cause signals to be triggered can be triggered at any time, however. In addition, it is possible to embed metadata into signals, such as a custom signal name, who should receive it, how “far” it should travel, what priority it has, etc [AI04].

4.2.5.8 Goals and goal pipes

Just as Behaviour and Character scripts control entity state management within the game engine, goal scripts control its actions: ie what an entity does when it is in any particular state. The Far Cry artificial intelligence subsystem defines a set of “atomic goals”, actions which cause an entity to engage in a simple activity, which crucially cannot be decomposed into simpler sub-activities. They are term “atomic” appears to have been borrowed from transaction processing theory, as goals are either executed through to successful completion, or do not execute at all. Complex entity behaviour is achieved by combining atomic goals into a linear list, called a “goal pipe” which is then executed serially. Goal pipes can be constructed from any combination of atomic goals and other goal pipes; highly complex hierarchies of goal pipes can be built up in this manner (*Illustration 16 [AI04])* .

There are two aspects of this approach which support the notion of multi-threading. The first is that goals can be blocking or non-blocking. On encountering a blocking goal, the artificial intelligence subsystem will not execute another goal until the current goal is complete. On encountering a non-blocking goal, it will execute the goal in parallel with other subsequent goals. The second is that goal pipes can be nested – at any point during the execution of a goal pipe, another goal or goal pipe can be inserted into the currently executing pipe. When this happens, execution of the current goal pauses, the nested pipe is executed to completion, then execution of the original goal resumes.

There is no practical limit to the depth of goal pipe nesting, which allows very complex and flexible behaviours to be constructed [AI04].

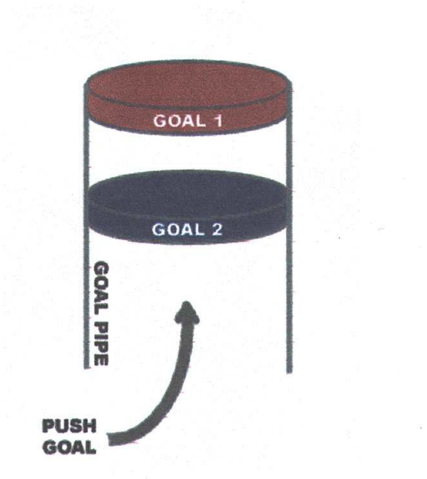


Illustration 16: FarCry Artificial Intelligence System Goal Pipe Structure

4.2.5.9 Navigation

The game engine uses an A* pathfinding algorithm to allow entities to be navigated through a game level. Paths are constructed over the accessible terrain around objects by a process of triangulation, which is carried out at design time in the level editor *(see Illustration 17. Note that the dark grey areas represent objects in the game level, the dark circles represent the nodes and the links between the nodes represent the paths that the AI entities may follow [*AI04*]).* The output of the triangulation process is the node graphs, which describe accessible points in the game level and links between the nodes which allow paths between the nodes to be constructed by the pathfinding algorithm. A “hide” graph is also generated, which describes points on the map which are located adjacent to impenetrable objects and as such can be regarded as cover, or hide points.

The node graph can be influenced by the developer by the placement of cover objects, and augmented by the placement of point and area objects which entities can recognise and react to by parsing meta-data associated with them.

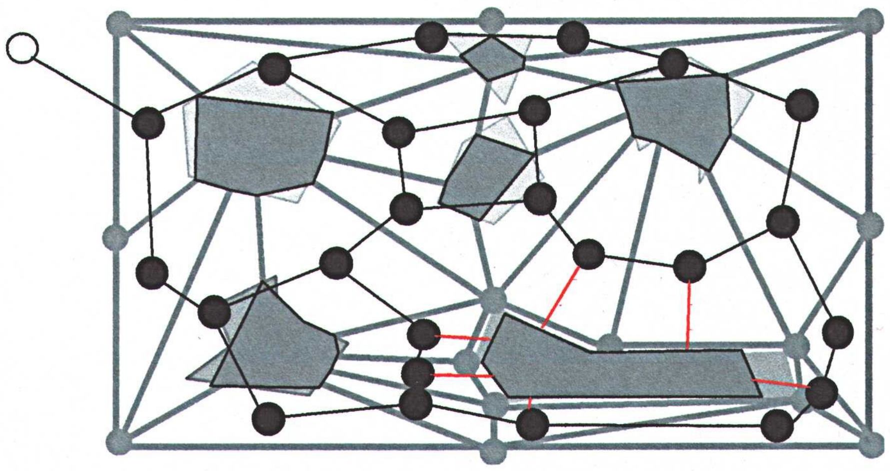


Illustration 17: Triangulation is used to generate the node graph

4.3 Development Process

4.3.1 Domain Analysis

The first step in the development process was to analyse the research material conducted on military combat manoeuvres. Descriptions of tactical manoeuvres were examined, with a view to identifying pieces of information that have the potential to be mapped onto a software model. By parsing several different textual descriptions of a manoeuvre, it was possible to identify candidate entities, relationships between entities and operations upon entities by isolating nouns noun phrases, verbs and verb phrases. After this, the list of candidate entities, operations and relationships was filtered to remove any which appeared redundant, irrelevant, vague, could be better modelled as attributes of an entity, which already existed within another sub-system, or which were concerned purely with implementation details.

The next step was to select one tactical manoeuvre to be the focus of the project and concentrate on modelling the software assets relevant to it. The “Assault” manoeuvre was chosen for this purpose, for the following reasons:

- Collaboration between team member entities with different roles was required.
- Information exchange to ensure the success of the mission was required.
- There was scope for incorporation of enemy entities with artificial intelligence.
- Co-ordination of the manoeuvre at team level by a team leader entity (centralised control) was required.
- Co-ordination of the manoeuvre at individual level by agent entities (decentralised control) was required.
- The manoeuvre had clearly defined start and end points.

- There was scope for alteration of the manoeuvre parameters to indirectly influence the outcome (emergent behaviour).
- There was scope for incorporating human player interaction into the simulation.

This resultant list of classes and operations is presented in Appendix A. It should be noted that effort was made to keep operation list generic, to facilitate inheritance and polymorphic method calling wherever appropriate and possible.

4.3.2 Design

The purpose of the design stage was to translate the plain text information derived during the analysis stage into constructs which represent the logical units of the software and document them graphically. The aim was to create a design in which these logical units were decomposed and refined to a point where implementation of a single unit could be regarded as relatively trivial.

The following subsections detail the results of this exercise.

4.3.3 Static Model

4.3.3.1 Class diagram

A class diagram was constructed to illustrate the entities identified during analysis and the relationships between them. The diagram itself is presented in Appendix B.

4.3.4 Dynamic Model

4.3.4.1 Event trace diagrams

Event trace diagrams were constructed to illustrate how the sequence of execution affects - and is affected by - each of the entities identified during analysis. The links which occur between each entity represent the events that each entity must handle in the finished application. The collection of event trace diagrams is presented in Appendix C.

4.3.4.2 State Transition Diagrams

During execution of the application, entities can be considered to be either reacting to events (which for design purposes can be considered as instantaneous activities), or carrying out activities which have some duration. These periods of activity are the states that the entity can be in. To illustrate the states identified for each entity, the events which cause changes of state and the actions carried out during each state, a series of state transition diagrams were constructed. These are presented in Appendix D.

4.3.4.3 Process description pseudocode

Each of the actions carried out by an entity whilst in a particular state must have some form of abstract description associated to allow them to specified in a programming language-independent manner. To this end, pseudocode processing specifications were constructed for each entity. An example of a typical processing specification, together with the event handler and goal pipe scripts derived from it are shown below:

Example of a typical pseudocode process specification:

```
function MOVE_UP
      get the identity of the destination waypoint
      get the destination waypoint object
      if the object is valid then
             send signal to indicate departure event
             enable upright body position animation
             enable running animation
             disable weapon firing
             ignore all previously registered attention targets
             request a path to the waypoint
             enable look in direction of travel animation
             get the distance to the destination waypoint object
             if the distance is greater than 20 metres then
                    approach 15% along the path to the waypoint
             else
                    approach 100% along the path to the animation
             endif
             get the location of the nearest cover point
             request a path to the cover point
             approach the cover point
             disable running animation
             enable weapon firing
             send signal to indicate arrival event
      endif
end function
```

Example of a typical event handler script:

```
MOVE_UP = function (self, entity, sender)

      -- We can move up, so get the destination tagpoint name
      local name = BlackboardObject:getDestination(entity.id);
      local tagpoint = Game:GetTagPoint(name);

      if (tagpoint) then
             -- Get our distance from the tagpoint
             local dist = entity:GetDistanceFromPoint(tagpoint);

             -- If distance is > 20 metres then advance by 15%
             -- else go directly to the tagpoint
             if (dist > 20) then
                    entity:InsertSubpipe(0, "SJDGruntJobPatrolPath", name);
             else
                    entity:InsertSubpipe(0, "SJDGruntJobRunToTagPoint", name);
             end
      end
end,
```

Example of a typical goal pipe script:

```
AI:CreateGoalPipe("SJDGruntJobPatrolPath");
AI:PushGoal("SJDGruntJobPatrolPath", "signal", 0, 1, “LEAVING_FOR_NEXT_POINT", 0);
AI:PushGoal("SJDGruntJobPatrolPath", "bodypos", 1, 0);
AI:PushGoal("SJDGruntJobPatrolPath", "run", 1, 1);
AI:PushGoal("SJDGruntJobPatrolPath","devalue",1,0);
AI:PushGoal("SJDGruntJobPatrolPath","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDGruntJobPatrolPath", "acqtarget", 1, "");
AI:PushGoal("SJDGruntJobPatrolPath","lookat", 1, 0, 0);
AI:PushGoal("SJDGruntJobPatrolPath", "approach", 1, 0.85, 1);
AI:PushGoal("SJDGruntJobPatrolPath", "locate", 1, AIAnchor.AIANCHOR_SHOOTSPOTCROUCH);
AI:PushGoal("SJDGruntJobPatrolPath", "pathfind",1,"");
AI:PushGoal("SJDGruntJobPatrolPath", "trace",1,1);
AI:PushGoal("SJDGruntJobPatrolPath", "run", 1, 0);
AI:PushGoal("SJDGruntJobPatrolPath","firecmd", 1, 1); -- resume firing at will
AI:PushGoal("SJDGruntJobPatrolPath", "signal", 1, 1, "GOT_TO_NEXT_POINT", 0);
```

4.4 *Game Asset Descriptions*

4.4.1 Game Level

The game level was designed to act as the battlefield upon which the assualt manoeuvre would be carried out (*Illustration 19*). The layout represents a volcanic island with a sandy beaches and a cove. An entrance to a hypothetical subterranean bunker is positioned at the rear of the cove. The objective of the assault team is to move from their rally point close the water's edge, through the cove to the bunker entrance. To do this they must first determine the location of any opposing force defence installations, and then neutralise them, whilst minimising the team's casualties.

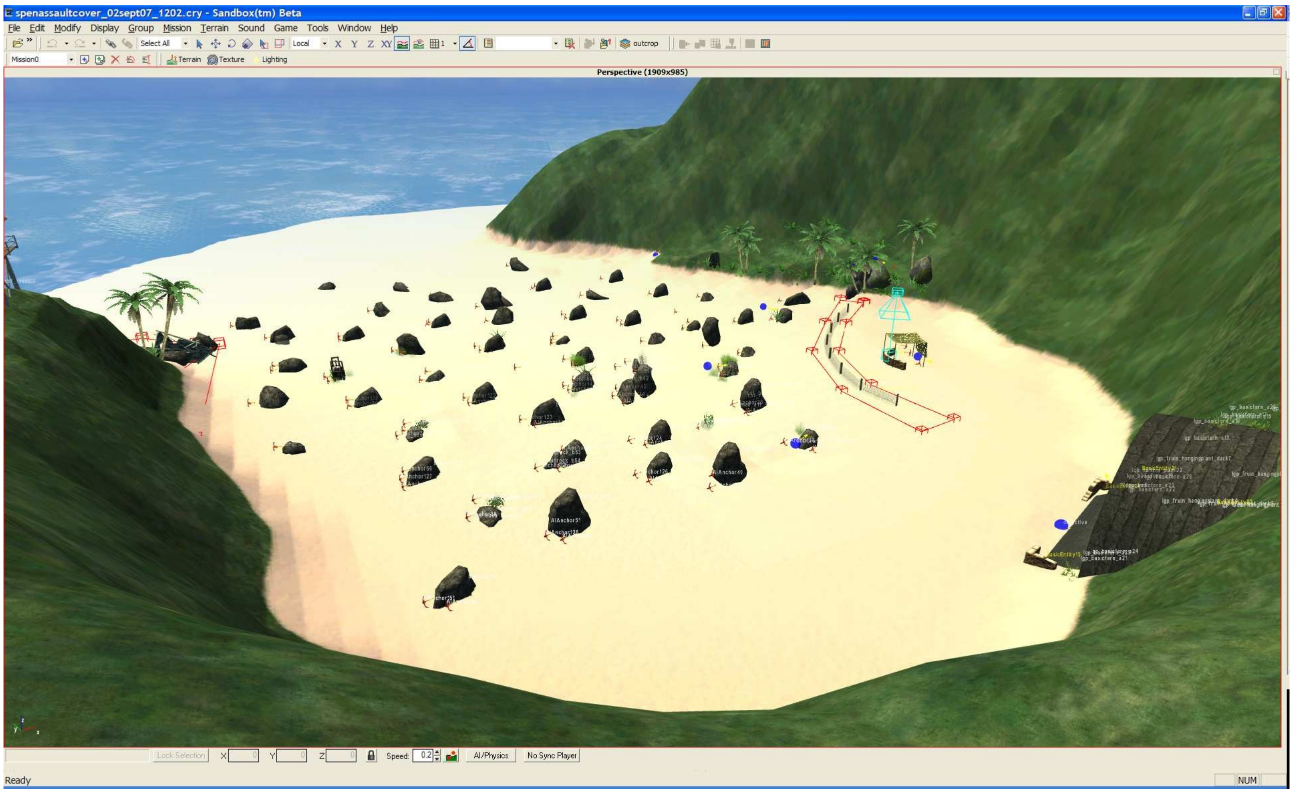


Illustration 18: Project Game Level Map

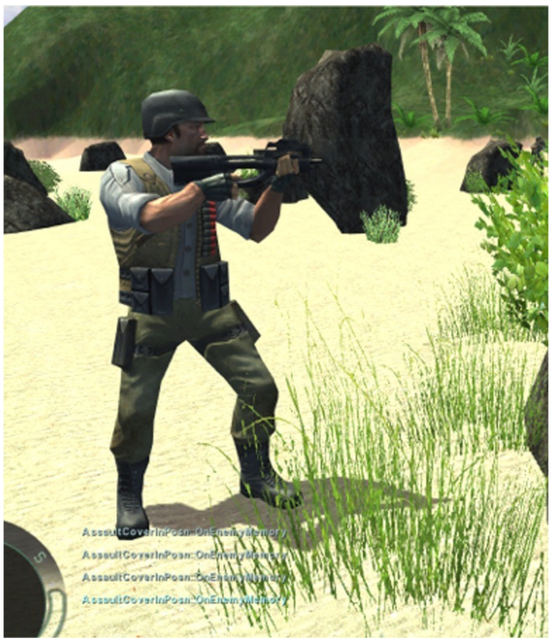
4.4.2 Scout entity

The Scout entity is responsible for carrying out reconnaissance of the assualt route, with a view to pinpointing the location of any opposing force defence installations. His goal is do this covertly, avoiding detection, and then report back to the rally point with the information obtained. He must be able to deal with foot soldiers he encounters along the way, but his overall priority in reconnaissance (*Illustration 19*).

Illustration 19: Scout Entity

4.4.3 Cover entity

Once the position of any opposing force defences have been pinpointed, the Cover's responsibility is to move towards that location and engage the opposing forces. The strength and protection of the defence will mean that a frontal attack would be futile, so the purpose of the cover team is to distract and pin down the enemy whilst the Breach team moves into position. Cover entities exhibit "fire-and-manouevre" behaviour – they only move up the battlefield if (a) no other cover team member is currently moving up (b) they have been waiting longer than all their other cover team members to move up (*Illustration 20*). If another cover team member is moving up, they provide cover fire for him if they currently have line of sight to the enemy position.



```
AssaultCoverInPosn::OnEnemyMemory
AssaultCoverInPosn::OnEnemyMemory
AssaultCoverInPosn::OnEnemyMemory
AssaultCoverInPosn::OnEnemyMemory
```

Illustration 20: Cover Entity engaging enemy

4.4.4 Breach entity

The role of the Breach entity is to stealthily move into a position which gives him a tactical advantage over the enemy, whilst they are being engaged by the Cover team. His aim is to exploit weaknesses in the defensive line to out-flank the enemy and launch a surprise attack using heavy munitions with the aim of neuralising the defence position in one rapid movement (*Illustration 21*).

Illustration 21: Breach Entity

4.4.5 Team Leader entity

The Team Leader's responsibility is to coordinate the efforts of the team and the information gathered by them, to ensure the successful completion of the mission. He is also responsible for recognizing when individual failures mean that the mission has to be aborted to minimise his own team's casualties.

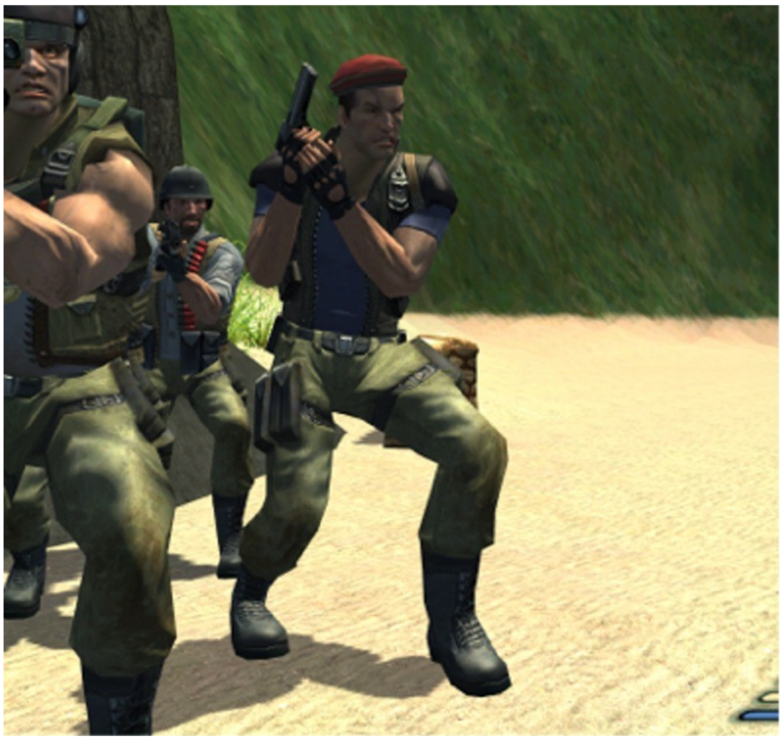
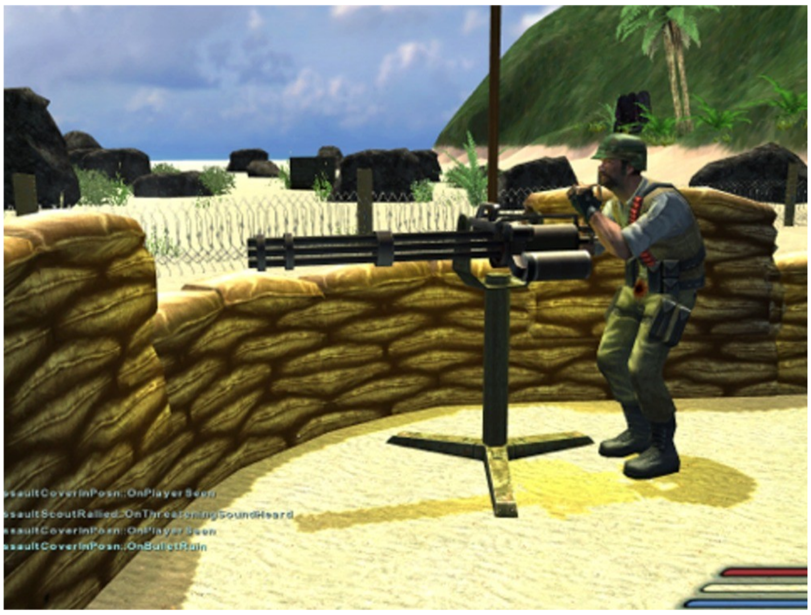


Illustration 22: Team Leader Entity Screenshot

4.4.6 Enemy Gunner entity

The Enemy Gunner entity's job is to man the defensive installation at the entrance of the bunker. He is heavily armed and heavily protected, but is reluctant to stray from his installation, as mounted weapons form the main strength of his defences. He must recognise when the team mate currently manning the mounted weapon has been killed and move to take his place before the assault team gains an advantage.



```
ssaultCoverInPosn::OnPlayerSeen
ssaultScoutRallied::OnThreateningSoundHeard
ssaultCoverInPosn::OnPlayerSeen
ssaultCoverInPosn::OnBulletRain
```

Illustration 23: Enemy Gunner Entity Screenshot

4.5 Implementation

The entity states, events and actions were translated in Lua entity behaviour, character and goal pipe scripts. The blackboard and combat state objects were translated into C++ source, together with the interface methods needed to expose their functionality to the Lua scripting environment. These decision to implement these in C++ in a separate library was taken (a) to maximise potential for reuse and portability (b) to impose logical separation of the stored game data from the entity processing code and (c) to reduce the potential for side-effects (ie “back-door” data modification) which was deemed much more likely in the largely global Lua scripting environment.

The Lua script and C++ source code is presented in Appendix F.

5 Results

5.1 Artificial Intelligence Performance

5.1.1 Scout Entity

The Scout entity proved to be adept at approaching the enemy position stealthily, identifying its exact location and returning to the rally point. However, the introduction of rogue enemy guard entities onto the battlefield tended to keep the Scout pinned down behind cover for long periods of time, due to the Scout's tendency to choose self-protection over engagement in an effort to ensure the delivery of the position intelligence it has gathered.

5.1.2 Cover Entity

Groups of Cover entities work very well together as a fire team, collaboratively alternating their fire and manouevre behaviour to ensure the safe passage of their team mates to the enemy defensive line whilst under fire. The beginnings of emergent behaviour can be seen as the line of attack moves forward in unison. This is also evident in their ability to continue as a team, in the event of losing a team mate.

However, if the number of Covers in team is too large relative to the size of the battlefield, the need to move away from the fire line of team members to avoid friendly fire incidents means that Covers can spend more time moving than firing and the effectiveness of the team is reduced, sometimes to the point where all Covers are killed by the enemy gunners before reaching the defensive line. Other observations are that recovery from ducking when under fire might be improved by moving to the side before returning to an upright position, thus forcing the enemy to re-aim instead of waiting for the entity to reappear at the same point.

5.1.3 Breach Entity

When the combat conditions allow the Breach entity to complete its tasks as designed, it performs very effectively, approaching the breach point with stealth and responding quickly and violently when instructed to do so. Although it is the most heavily armed entity, it is also the weak point of the team, in that if it is spotted and killed before carrying out the breach, the mission has failed and must be aborted. It may help to operate Breaches in teams of two, one acting as the bombardier and one acting as a rifleman to provide close fire cover.

5.1.4 Team Leader

The behaviour of the Team Leader is akin to that of a logic gate. It responds to events by generating new events and such has no complex actions of its own to execute. The concept of a single controlling entity in an command role is one that has proved very effective in this scenario, however.

5.1.5 Collaboration as a team

Given that no single entity is programmed to complete the mission on its own, the fact that an entire assault manouevre can be completed through the combined efforts of entities performing different tasks under high level command is an indication of the success of the team behaviour.

5.1.6 Adding more entities

Adding more entities of any type (excluding the Team Leader) is a trivial matter given the extensible nature of the solution's architecture. Consideration must be given to density of entities on the battlefield to avoid the deterioration in performance that overcrowding causes.

5.1.7 Incorporating a human player and playability

A human player can be incorporated into the application, simply by assigning a value to the "species" attribute of the assault team entities which is the same as that of the player (this is carried out at design time). However, it is not at the moment possible for the player to assume any of the roles of the Assault team (e.g Scout, Breach, Cover etc). Implementing this functionality would be possible but as the project was conceived as a simulation exercise, it was deemed to be outside the original scope.

5.1.8 Playability

As previously mentioned, the project was intended (at this stage) to be a simulation exercise. It is however, entirely possible with the application in its present state of development to "join in" as a human player and contribute to the battle on either side, albeit in a "renegade" capacity with no fixed role.

5.2 Technical Performance

5.2.1 Scalability

The solution stands up to scaling very well. This is principally because the entities have been coded to complete their assigned tasks and recognise when they done so. Expanding the game level, or increasing the number of waypoints in the level would not affect the operation of the artificial intelligence. This is in contrast to the standard Far Cry entities which, as described earlier in this document, rely heavily on their the layout of their environment for guidance.

5.2.2 Portability

There are other commercially available games which use Lua as a scripting language, but it is not known whether the behaviour, character and goal pipe constructs used in Far Cry are compatible with these. For this reason, it is unlikely that the scripts could be used on a different game application without significant modification. However, the entity behaviour, character and goal pipe scripts can be physically and logically decoupled from the standard Far Cry scripts themselves and so could be easily be reused with, for example, a newer version of the game engine.

5.2.3 Maintainability

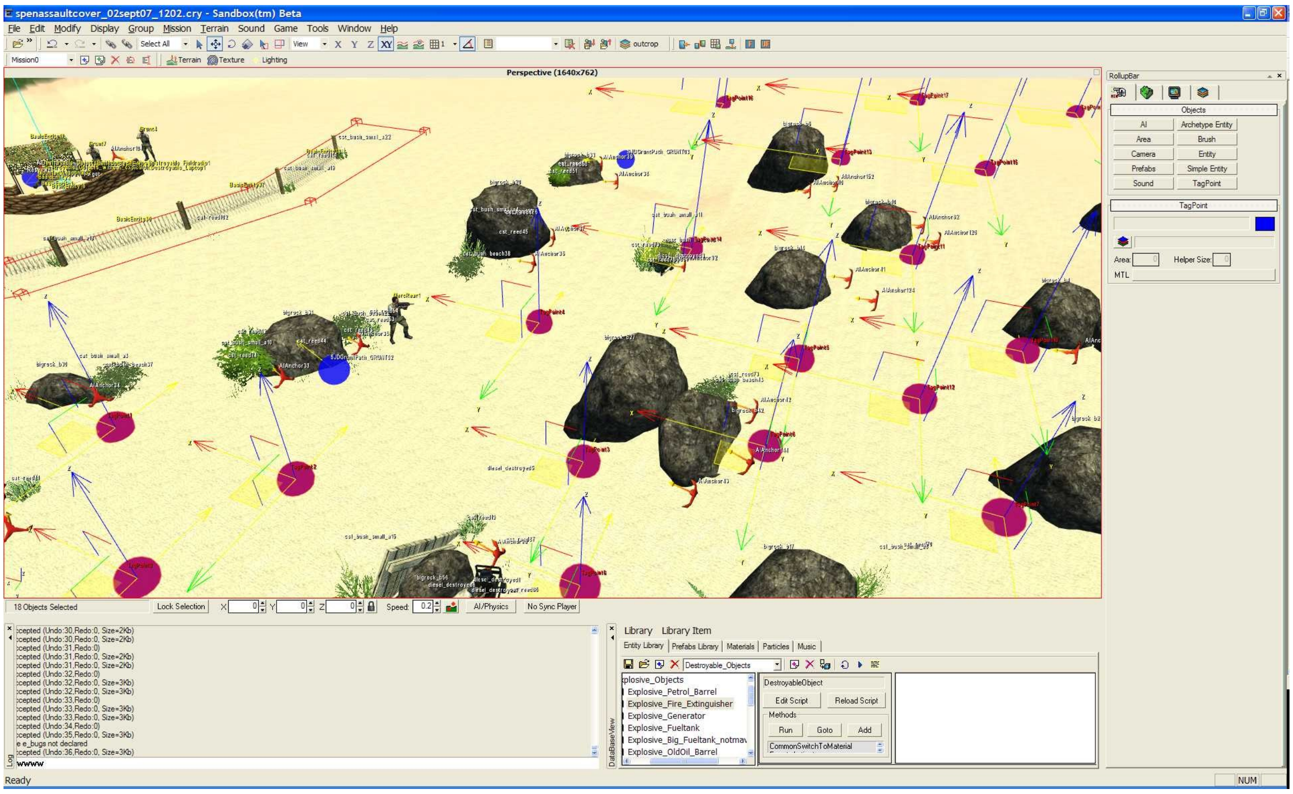
Development of the solution has been an iterative process, with the aim of achieving a working, but basic prototype early in the project, and then adding functionality in steps over time. It was recognised from the start that this approach would mean that the maintainability of the solution would have a significant impact the ease with which development progressed. Decoupling the game data (ie the blackboard and combat state objects) has without doubt this process much simpler. The inter-entity relationships changed many times during project and had the game data been encapsulated in the entities themselves, their interfaces would have had to change each time as well, creating a large development overhead. Instead, the single blackboard and combat state interface remained largely the same while the entity organisation changed significantly, but with none of the previously described overheads.

5.3 Problems encountered during development

5.3.1 Navigation and tactical waypoint selection

One of the key notions in manoeuvring is the ability to be able select the next destination point based on its tactical value (e.g visibility, exposure, accessibility, etc). It was discovered that the Far Cry SDK does not provide direct access to the node graph, which acts as the source of navigational data for an artificial intelligence entity. Methods are provided which allow the data to be queried to a degree, but these methods focus primarily on pathfinding to a given point, and do not allow for spatial querying. For example, the need to locate the node with the highest altitude within a given radius was identified during the project. With no direct access to the node graph, this type of querying ability was not possible.

The solution was create a separate logical data layer and overlay it above the node graph. This layer was populated with "tag points" - simple objects which are placed by the developer at design time – arranged in a loose grid format *(Illustration 24).* Method were then written to provide aptial querying of the tag point collection. Tag points are incorporated into the node graph during the triangulation process and can be passed as a destination points to the standard pathfinding routines. Using this method, entities were given a choice of waypoints to select from during navigation, whose metadata was readily accessible for the purposes of tactical assessment.



```
18 Objects Selected   Lock Selection   X 0  Y 0  Z 0  Speed: 0.2  AI/Physics  No Sync Player
```

```
:cepted (Undo:30,Redo:0, Size=2Kb)
:cepted (Undo:30,Redo:0, Size=2Kb)
:cepted (Undo:31,Redo:0)
:cepted (Undo:31,Redo:0, Size=2Kb)
:cepted (Undo:31,Redo:0, Size=2Kb)
:cepted (Undo:32,Redo:0)
:cepted (Undo:32,Redo:0, Size=3Kb)
:cepted (Undo:32,Redo:0, Size=3Kb)
:cepted (Undo:33,Redo:0)
:cepted (Undo:33,Redo:0, Size=3Kb)
:cepted (Undo:33,Redo:0, Size=3Kb)
:cepted (Undo:34,Redo:0)
:cepted (Undo:35,Redo:0, Size=3Kb)
e e_bugs not declared
:cepted (Undo:36,Redo:0, Size=3Kb)
wwww
Ready
```

```
xplosive_Objects
  Explosive_Petrol_Barrel
  Explosive_Fire_Extinguisher
  Explosive_Generator
  Explosive_Fueltank
  Explosive_Big_Fueltank_notmav
  Explosive_OldOil_Barrel
```

Illustration 24: Using Tag Points to augment the node graph

5.3.2 Dependence of AI on environment layout

The behaviour of artificially intelligent entities in Far Cry is influenced heavily by the layout of objects in the game environment. This is not a mistake or an oversight – many game developers seek to make the artificial intelligence aspect as game-independent as possible, by embedding information which can be used to direct entities within the objects in the game level, rather than the code. This resulted in a slightly different approach to development being taken than was originally anticipated. This is because as the intention was to create an environment which would allow emergent behaviour to develop where appropriate. It was believed that having entities influenced by objects would tend to produce results which might appear overly "directed". The solution was to embed the data originally extracted from game objects in the blackboard object, thereby retaining the entity's autonomy and flexibility, but reducing its dependence of its behaviour on the layout of its environment.

5.3.3 SDK documentation

The Far Cry software development kit is supplied with comprehensive suite of documentation which provides a great deal of very detailed information on the level editor, the scripting environment and the various tools that are used to create a game "mod". However there is very little information regarding the C++ libraries, other than descriptions of the functionality they provide; there is for example no C++ API specification. The documentation for the subset of library functionality which is exposed to the Lua scripting environment is very detailed, but beyond this, interpretation is very much up to the developer – either by static analysis of the source code, or by a process of trial and error. This did hinder the development process somewhat in the later stages of the project, as progress caused the demands placed on the SDK to exceed that provided by the standard scripting interface.

The approach taken was to identify within the C++ library source (by analysis) those methods which offered the required functionality and expose them to the Lua environment by writing script interface methods.

6 Conclusions and Future Work

In summary, the project can be considered to be very successful, as both of the original objectives have been met. The end result of the project is a system which allows collaborative team artificial intelligence to be constructed and demonstrated from a design derived from analysis of real life tactical manoeuvres. The system is robust, scalable and simple to alter, adapt and tune. Whilst initially considered as a simulation exercise, the system has the benefit of being able to accommodate a human player, albeit not as a true team member. Some uncertainty remains over the portability of the system, but it is hoped that this issue can be addressed with further work in the future.

The following is a list of recommended areas for future work:

Refine existing Assault manoeuvre
Increase the number of criteria used when selecting waypoints according to their tactical value during pathfinding.

Model more manoeuvres utilising centralised control
Construct scripts for other real world, team-based, centralised-control manoeuvres, including Ambush, Patrol and Raid.

Model more manoeuvres utilising decentralised control
Construct scripts for more manoeuvres which rely on decentralised control. Room clearing is a good example of this, as there are many texts which describe urban combat room clearing tactics in detail, which might prove useful during analysis and design.

More complex missions
With the current architecture, the ability to switch between behaviours under the control of a command entity means that hypothetically, teams could embark on lengthy campaigns, selecting the manoeuvre most suitable for the prevailing battle conditions. More complex game levels and mission scenarios would need to be built to exercise and explore this possibility.

Allow player to join teams
Investigate allowing the player to join a team and assume the role of a team member, working collaboratively with artificially intelligent entities.

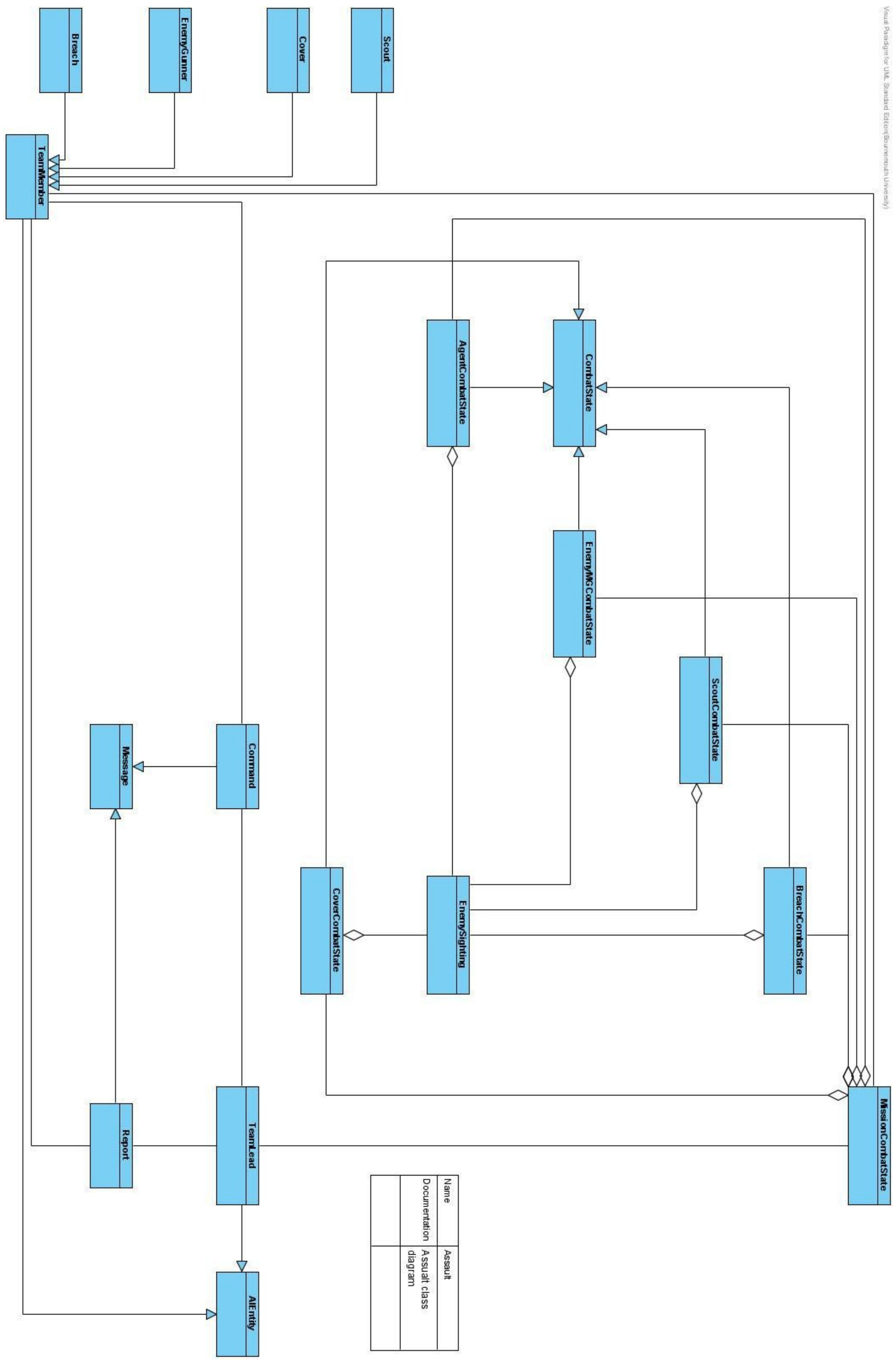
Decouple AI from game
Look at ways of decoupling the artificial intelligence behaviour functionality from the host game application, but at the same time retaining the flexibility of scripting environment.

Appendix A: Candidate Classes and Operations

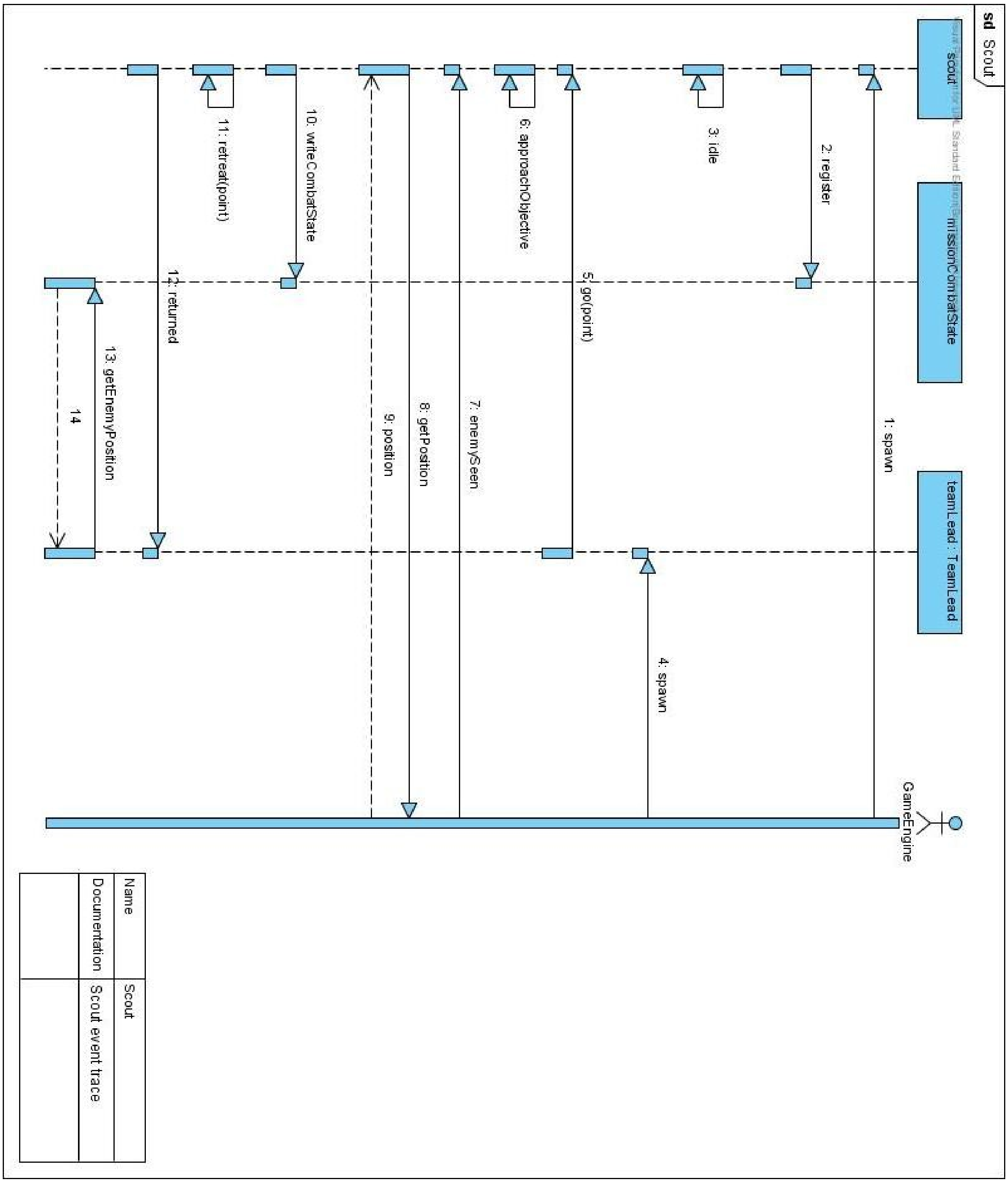
Classes:
Scout
Cover
TeamLead
Breach
EnemyGunner
ObjectivePoint
RallyPoint
BreachPoint
CoverPoint
OperationOrder
Blackboard
AgentCombatState
MissionCombatState

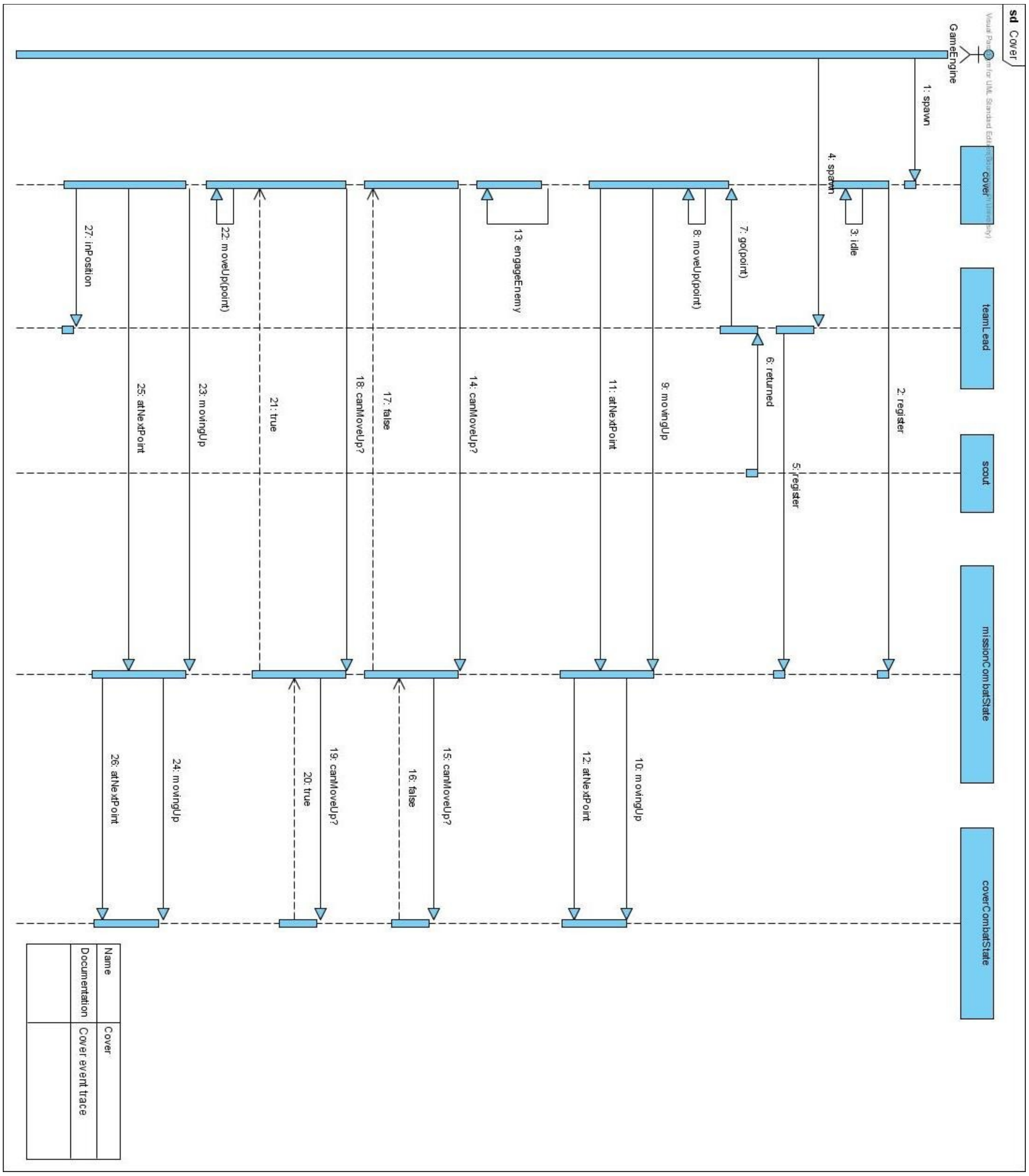
Operations:
spawn
register
idle
approachPoint
getEntityPosition
hide
shoot
takeCover
moveUp
digIn
issueOrder
receiveOrder
writeCombatState
readCombatState

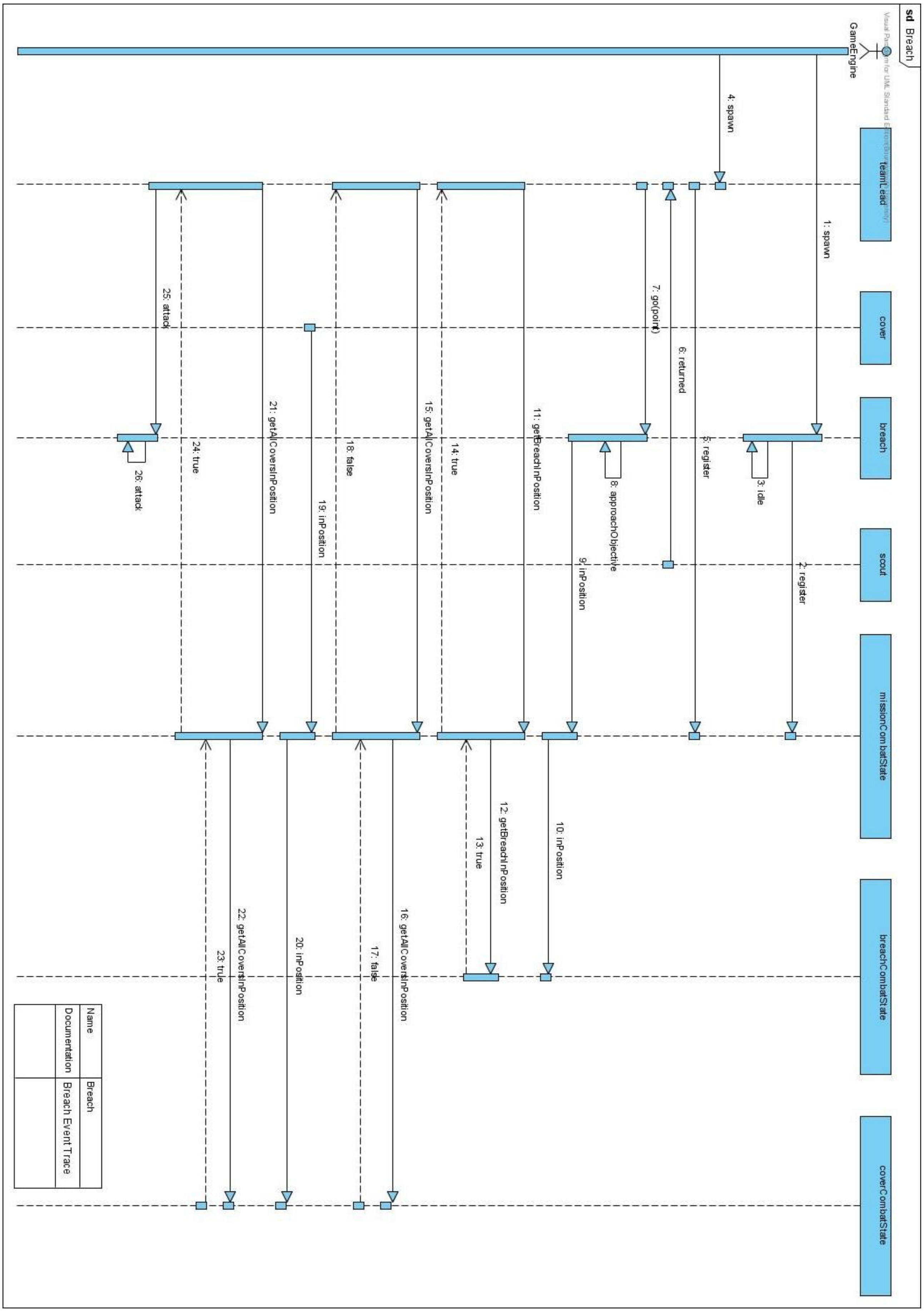
Appendix B: Class Diagram

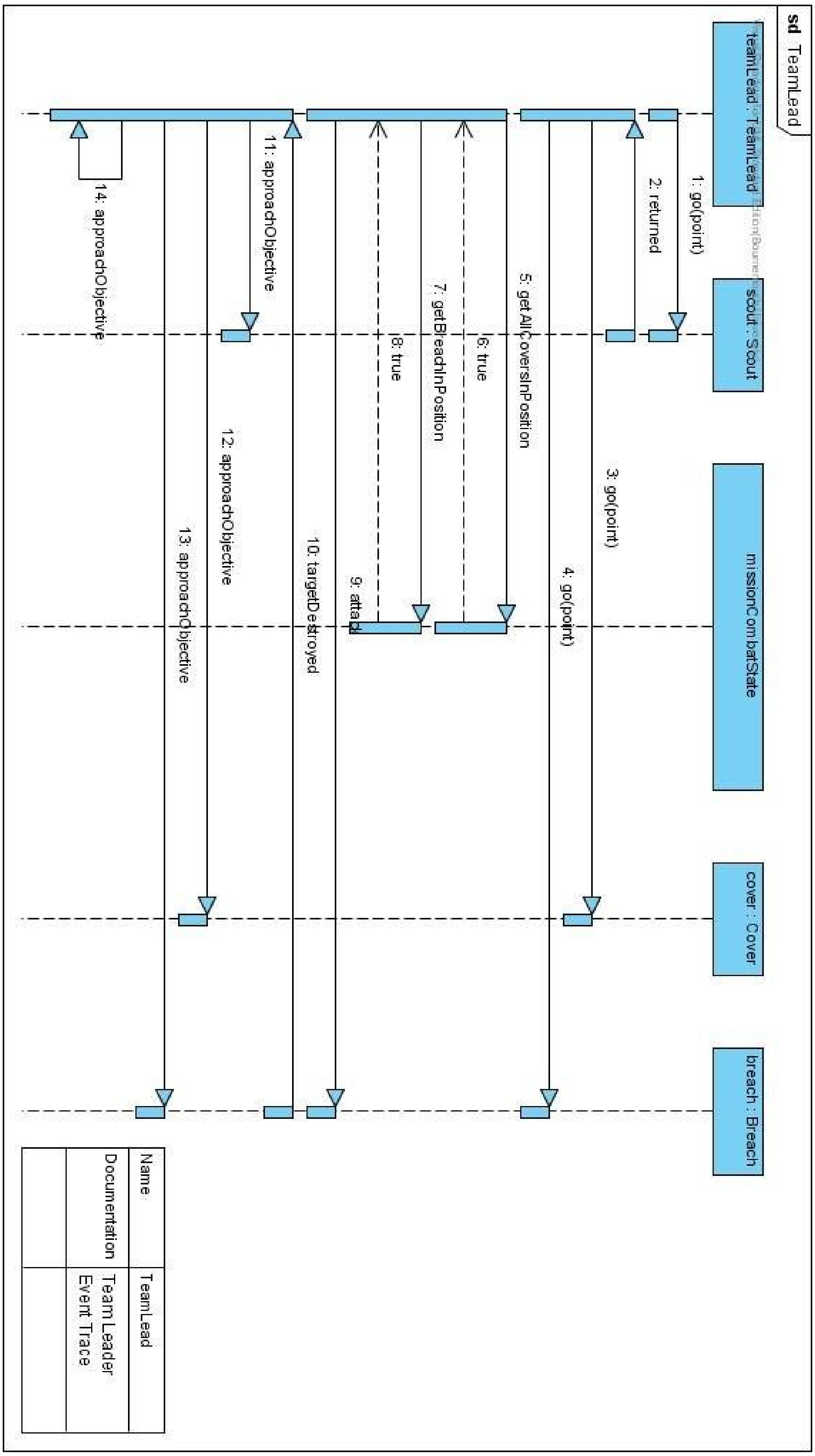


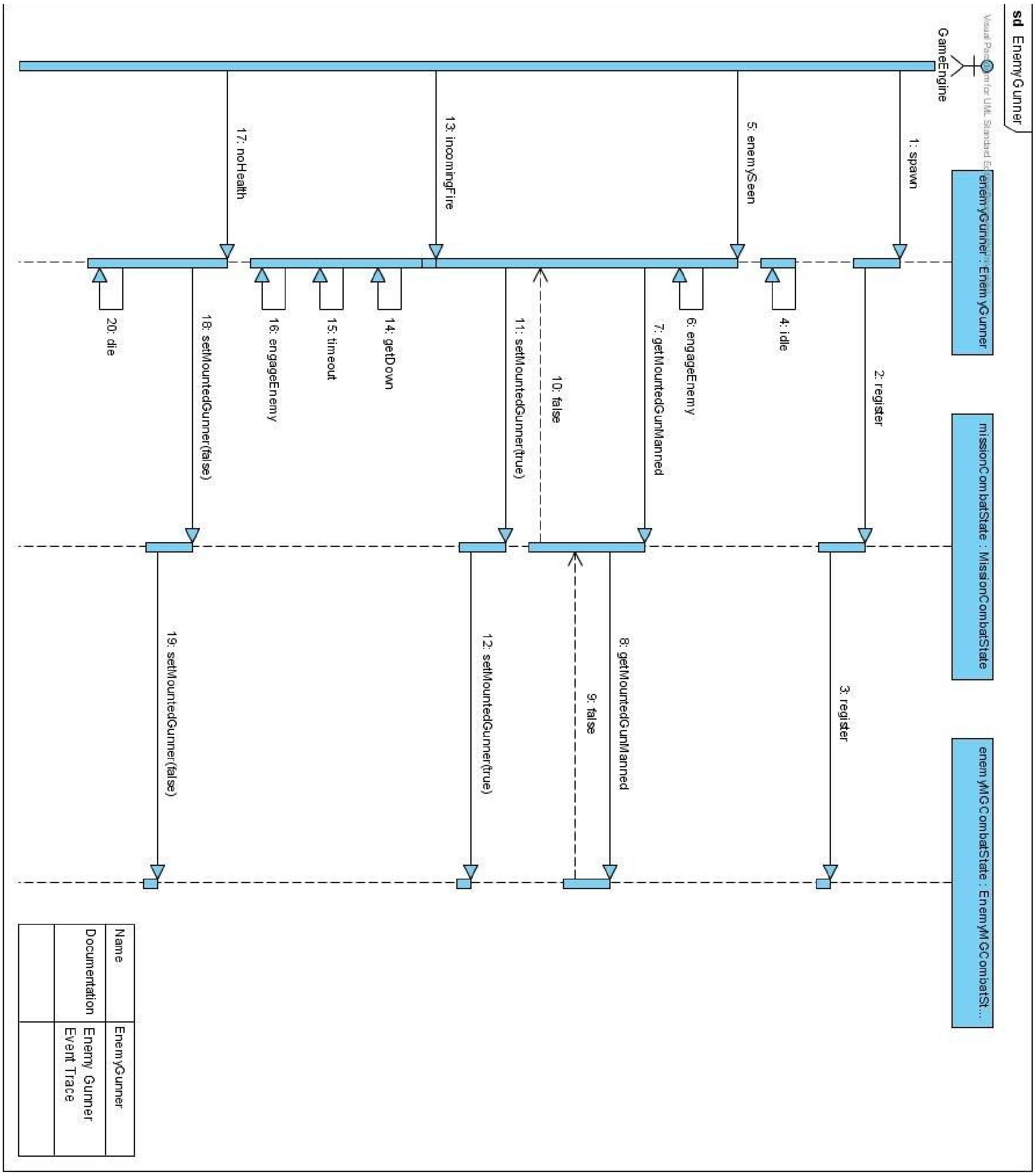
Appendix C: Event trace diagrams



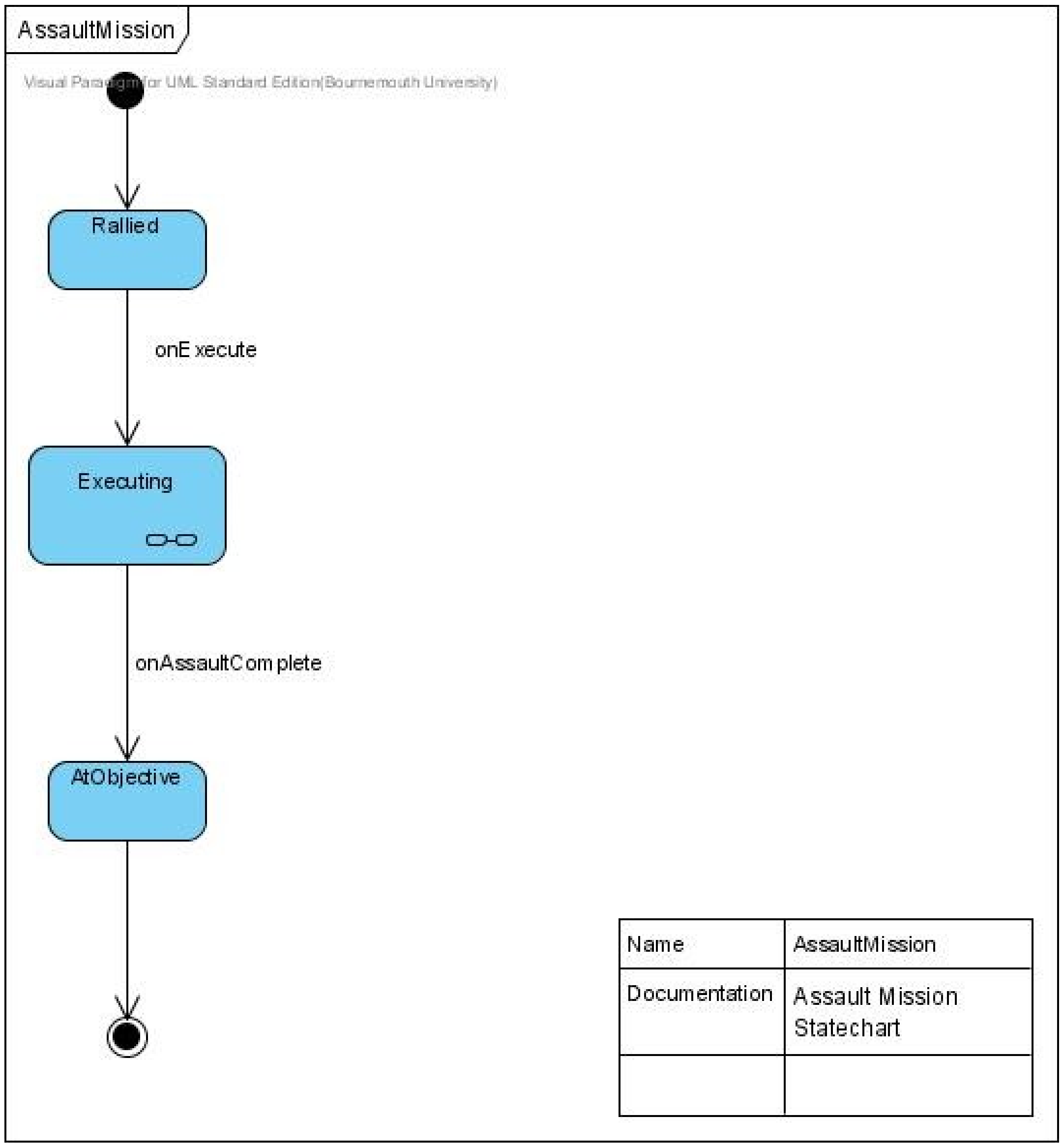

sd Cover
Visual Paradigm for UML Standard Edition(...University)
GameEngine
cover
teamLead
scout
missionCombatState
coverCombatState
1: spawn
2: register
3: idle
4: spawn
5: register
6: returned
7: go(point)
8: moveUp(point)
9: movingUp
10: movingUp
11: atNextPoint
12: atNextPoint
13: engageEnemy
14: canMoveUp?
15: canMoveUp?
16: false
17: false
18: canMoveUp?
19: canMoveUp?
20: true
21: true
22: moveUp(point)
23: movingUp
24: movingUp
25: atNextPoint
26: atNextPoint
27: inPosition
Name
Cover
Documentation
Cover event trace


sd Breach
GameEngine
teamLead
cover
breach
scout
missionCombatState
breachCombatState
coverCombatState
1: spawn
2: register
3: idle
4: spawn
5: register
6: returned
7: go(point)
8: approachObjective
9: inPosition
10: inPosition
11: getBreachInPosition
12: getBreachInPosition
13: true
14: true
15: getAllCoversInPosition
16: getAllCoversInPosition
17: false
18: false
19: inPosition
20: inPosition
21: getAllCoversInPosition
22: getAllCoversInPosition
23: true
24: true
25: attack
26: attack
Name
Breach
Documentation
Breach Event Trace

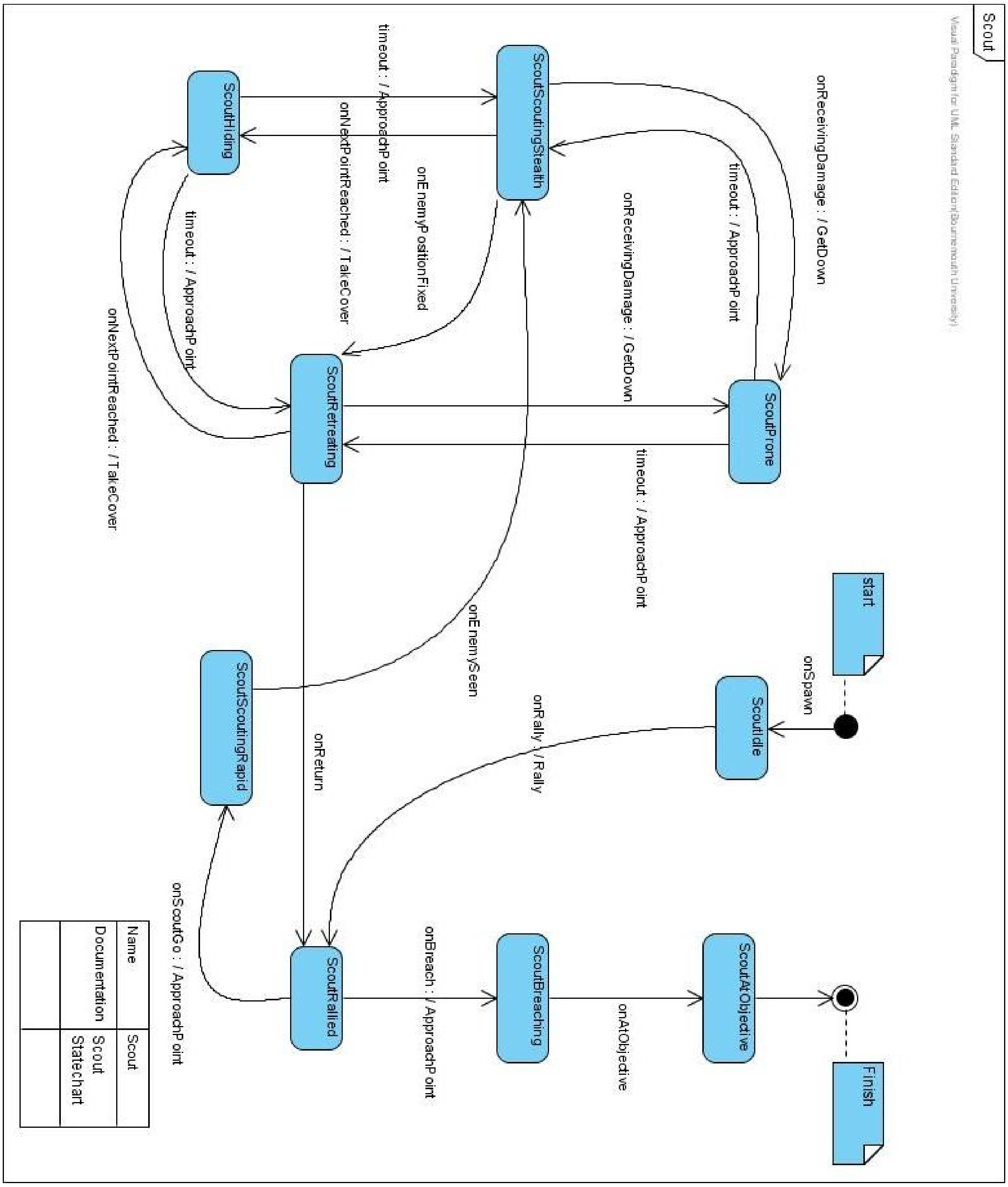

sd TeamLead
teamLead : TeamLead
scout : Scout
missionCombatState
cover : Cover
breach : Breach
1: go(point)
2: returned
3: go(point)
4: go(point)
5: getAllCoversInPosition
6: true
7: getBreachInPosition
8: true
9: attack
10: targetDestroyed
11: approachObjective
12: approachObjective
13: approachObjective
14: approachObjective
Name
TeamLead
Documentation
Team Leader Event Trace

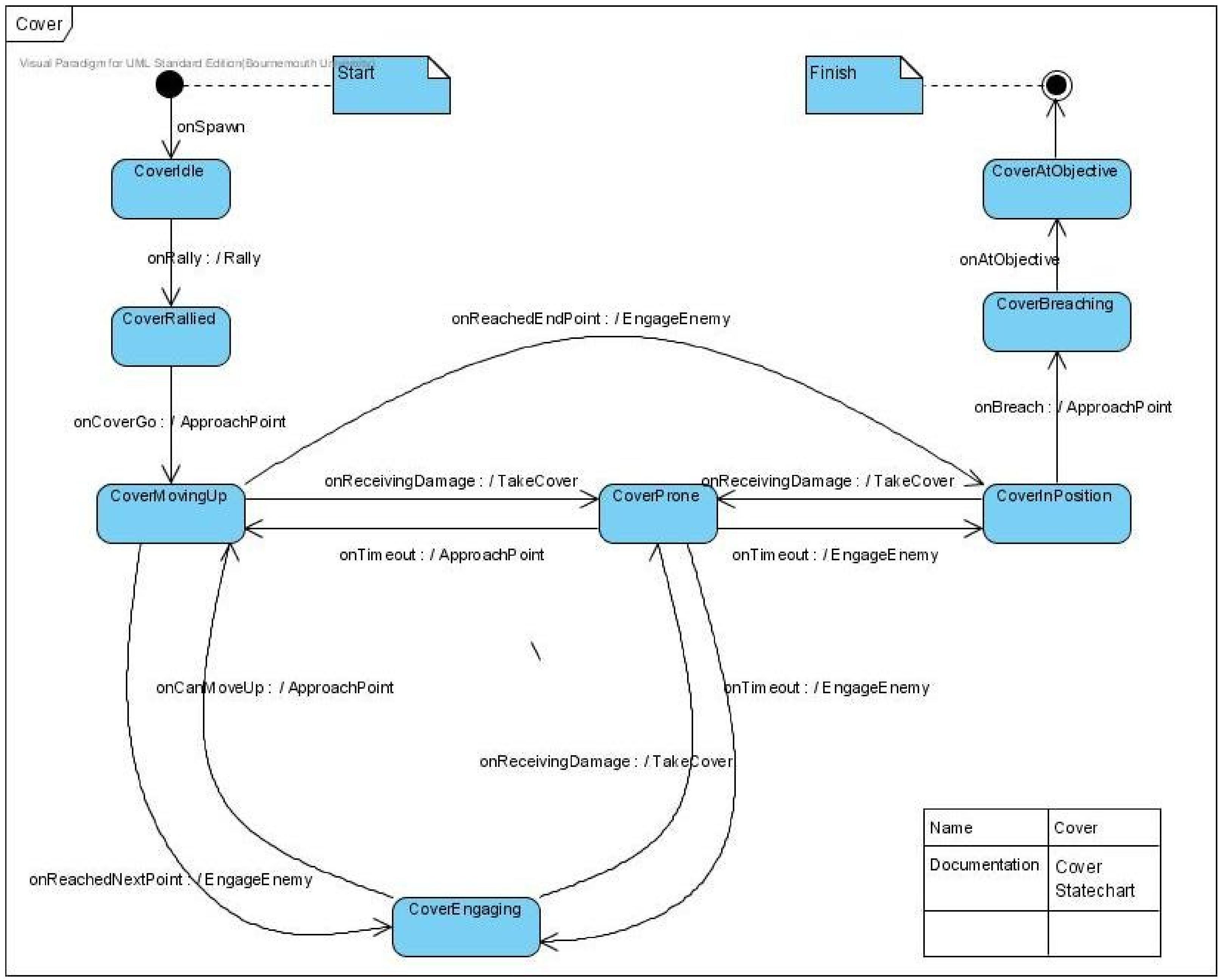

sd EnemyGunner
Visual Paradigm for UML Standard Edition
GameEngine
enemyGunner : EnemyGunner
missionCombatState : MissionCombatState
enemyMGCombatState : EnemyMGCombatSt...
1: spawn
2: register
3: register
4: idle
5: enemySeen
6: engageEnemy
7: getMountedGunManned
8: getMountedGunManned
9: false
10: false
11: setMountedGunner(true)
12: setMountedGunner(true)
13: incomingFire
14: getDown
15: timeout
16: engageEnemy
17: noHealth
18: setMountedGunner(false)
19: setMountedGunner(false)
20: die
Name
EnemyGunner
Documentation
Enemy Gunner Event Trace

Appendix D: State Transition Diagrams

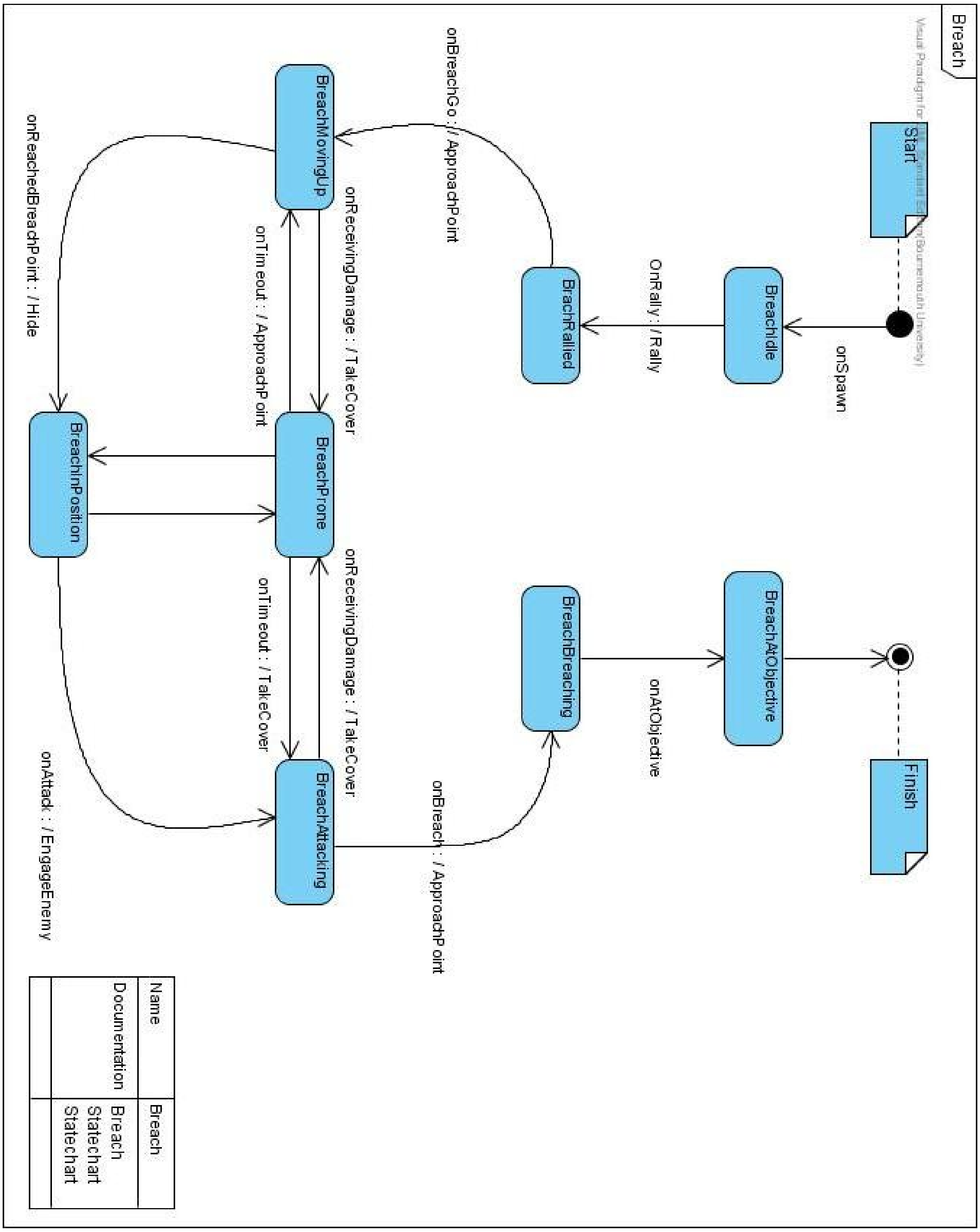


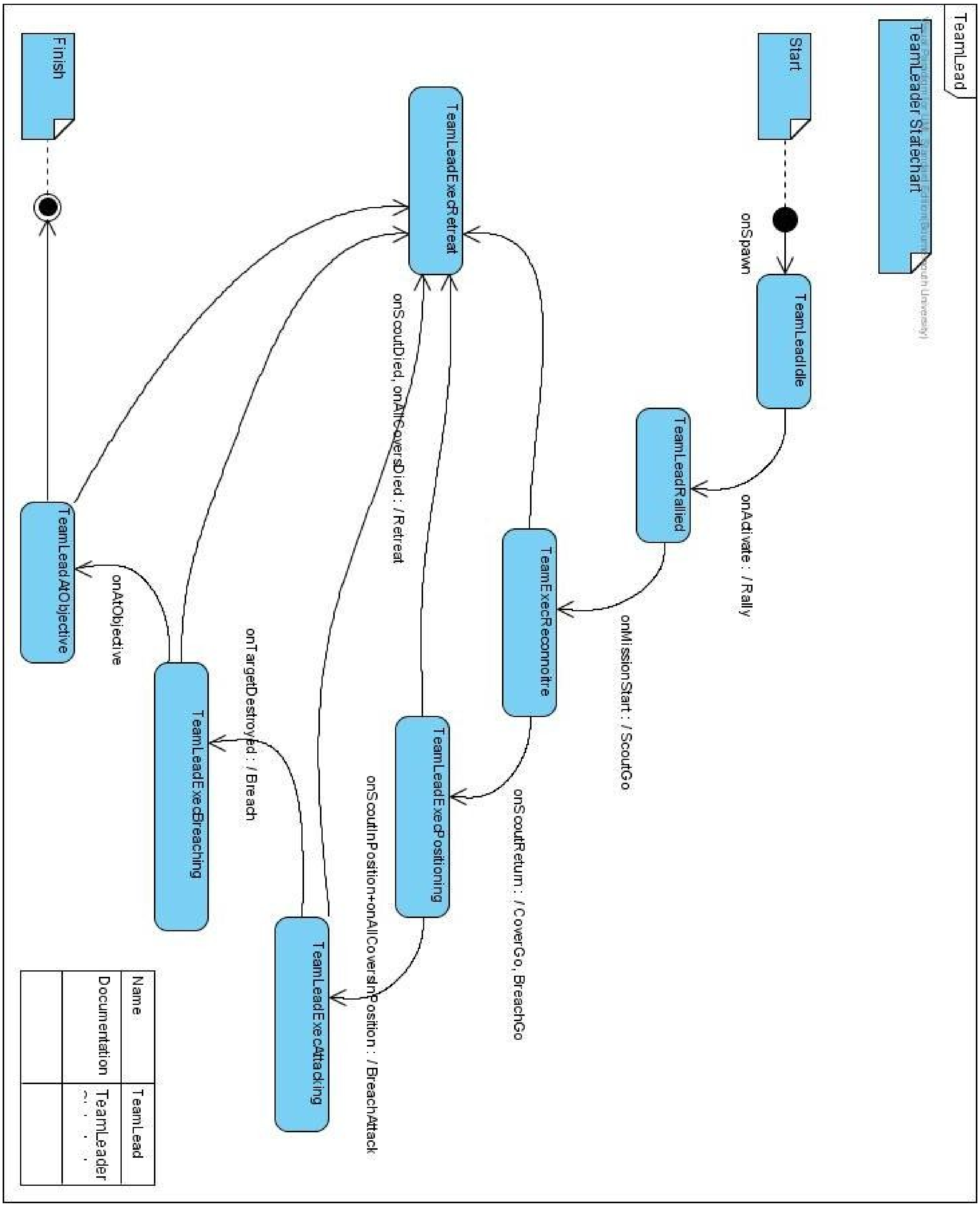
Name	AssaultMission
Documentation	Assault Mission Statechart


Scout
Visual Paradigm for UML Standard Edition(Bournemouth University)
start
onSpawn
ScoutIdle
onRally : / Rally
ScoutRallied
onBreach : / ApproachPoint
ScoutBreaching
onAtObjective
ScoutAtObjective
Finish
onScoutGo : / ApproachPoint
ScoutScoutingRapid
onEnemySeen
ScoutScoutingStealth
onReceivingDamage : / GetDown
timeout : / ApproachPoint
ScoutProne
onReceivingDamage : / GetDown
timeout : / ApproachPoint
ScoutRetreating
onReturn
onEnemyPositionFixed
ScoutHiding
timeout : / ApproachPoint
onNextPointReached : / TakeCover
timeout : / ApproachPoint
onNextPointReached : / TakeCover
Name
Scout
Documentation
Scout Statechart


Cover
Visual Paradigm for UML Standard Edition(Bournemouth University)
Start
Finish
onSpawn
CoverIdle
onRally : / Rally
CoverRallied
onCoverGo : / ApproachPoint
CoverMovingUp
onReachedEndPoint : / EngageEnemy
onReceivingDamage : / TakeCover
onTimeout : / ApproachPoint
CoverProne
onReceivingDamage : / TakeCover
onTimeout : / EngageEnemy
CoverInPosition
onBreach : / ApproachPoint
CoverBreaching
onAtObjective
CoverAtObjective
onCanMoveUp : / ApproachPoint
onReachedNextPoint : / EngageEnemy
onReceivingDamage : / TakeCover
onTimeout : / EngageEnemy
CoverEngaging

Name	Cover
Documentation	Cover Statechart


Breach
Start
onSpawn
BreachIdle
OnRally : / Rally
BrachRallied
onBreachGo : / ApproachPoint
BreachMovingUp
onReceivingDamage : / TakeCover
onTimeout : / ApproachPoint
onReachedBreachPoint : / Hide
BreachProne
BreachinPosition
onReceivingDamage : / TakeCover
onTimeout : / TakeCover
onAttack : / EngageEnemy
BreachAttacking
onBreach : / ApproachPoint
BreachBreaching
onAtObjective
BreachAtObjective
Finish
Name
Breach
Documentation
Breach Statechart Statechart


TeamLead
TeamLeader Statechart
Start
onSpawn
TeamLeadIdle
onActivate : / Rally
TeamLeadRallied
onMissionStart : / ScoutGo
TeamExecReconnoitre
onScoutReturn : / CoverGo, BreachGo
TeamLeadExecPositioning
onScoutInPosition+onAllCoversInPosition : / BreachAttack
TeamLeadExecAttacking
onTargetDestroyed : / Breach
TeamLeadExecBreaching
onAtObjective
TeamLeadAtObjective
TeamLeadExecRetreat
onScoutDied, onAllCoversDied : / Retreat
Finish
Name
TeamLead
Documentation
TeamLeader

Appendix E: C++ Source Code

```
//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: ScriptObjectBlackboard.h
//  Description: Script wrapper for blackbaord system
//
//      History:
//          25July2007       : File created
//          18August2007     : Added agent combat state management methods
//
//////////////////////////////////////////////////////////////////////

#ifndef _SCRIPTOBJECTBLACKBOARD_H_
#define _SCRIPTOBJECTBLACKBOARD_H_

#include <IScriptSystem.h>
#include <_ScriptableEx.h>
#include "AgentCombatState.h"
#include "EnemyMGState.h"
#include "BreachCombatState.h"
#include "ScoutCombatState.h"
#include <map>

/*!
    All Blackboard system related script functions are implemented in
    this class

        All functions are script-exclusive, they aren't called from the
        game C code
*/

class CScriptObjectBlackboard : public _ScriptableEx<CScriptObjectBlackboard>
{
public:

        //! Constructor
        CScriptObjectBlackboard( void ) {m_bAgentIsMovingUp = false;}

        //! Destructor
        virtual ~CScriptObjectBlackboard( void );

        //! Registers functions and global vars with LUA
        static void InitializeTemplate(IScriptSystem* pSS);

        //! Sets up this object as being accessible from LUA
        void Init(IScriptSystem* pScriptSystem, CXGame* pGame);

        //! Debug method - just prints a string to the console and the log file
        int PrintTestMessage(IFunctionHandler* pH);

    //********************************
    // Cover Combat State Management
    //********************************

    //! Add a new cover combat state
    int addCoverCombatState(IFunctionHandler* pH);

    //! Remove an existing cover combat state
    int removeCoverCombatState(IFunctionHandler* pH);

    //! Set the elapsed time of an cover combat state
    int setCoverElapsedTime(IFunctionHandler* pH);

    //! Get the elapsed time of an cover combat state
    int getCoverElapsedTime(IFunctionHandler* pH);

    //! Call to determine whether cover with specified id is OK to begin moving up
    int canCoverMoveUp(IFunctionHandler* pH);
```

```
//! Call to inform that the cover with specified id has finished moving up
int coverFinishedMovingUp(IFunctionHandler* pH);

//! Call to determine the number of cover agents registered
int getNumCovers(IFunctionHandler* pH);

//! Call to set the cover 'in position' state
int setCoverInPosn(IFunctionHandler* pH);

//! Call to get the cover 'in position' state
int getCoverInPosn(IFunctionHandler* pH);

//! Call to determine if all covers are in position
int getAllCoversInPosn(IFunctionHandler* pH);

//! Adds to the list of enemies that have been spotted
int addCoverEnemySighting(IFunctionHandler* pH);

//! Sets the 'alive' state of an enemy
int setCoverEnemyAlive(IFunctionHandler* pH);

//! Returns a list of enemy IDs
int getCoverEnemyIDs(IFunctionHandler* pH);

//! sets the Cover Destination
int setCoverDestination(IFunctionHandler* pH);

//!  gets the Cover Destination
int getCoverDestination(IFunctionHandler* pH);

//***************************************
// Enemy Machine Gunner State Management
//***************************************

//! Sets the id of the current gunner
int setGunnerID(IFunctionHandler* pH);

   //! Gets the id of the current gunner
int getGunnerID(IFunctionHandler* pH);

   //! Resets the gunner state
int gunnerDied(IFunctionHandler* pH);

   //! Gets whether a gunner is currently assigned
int gunnerAssigned(IFunctionHandler* pH);

//*********************************
// Breach Combat State Management
//*********************************

//! Add a new breach combat state
int addBreachCombatState(IFunctionHandler* pH);

//! Remove an existing breach combat state
int removeBreachCombatState(IFunctionHandler* pH);

//! Call to set the breach 'in position' state
int setBreachInPosn(IFunctionHandler* pH);

//! Call to get the breach 'in position' state
int getBreachInPosn(IFunctionHandler* pH);

//! Call to determine if all breaches are in position
int getAllBreachesInPosn(IFunctionHandler* pH);

//!  sets Breach Destination
int setBreachDestination(IFunctionHandler* pH);

//! gets the Breach Destination
int getBreachDestination(IFunctionHandler* pH);

//*********************************
// Scout Combat State Management
//*********************************

//! Add a new breach combat state
int addScoutCombatState(IFunctionHandler* pH);
```

```
    //! Remove an existing breach combat state
    int removeScoutCombatState(IFunctionHandler* pH);

    //! Adds to the list of enemies that have been spotted
    int addScoutEnemySighting(IFunctionHandler* pH);;

    //! Adds details of main target sighting
    int addScoutTargetSighting(IFunctionHandler* pH);

    //! Gets the position of the main target sighting
    int getScoutTargetSighting(IFunctionHandler* pH);

    //! Sets the destination of the scout
    int setScoutDestination(IFunctionHandler* pH);

    //! gets the destination of the scout
    int getScoutDestination(IFunctionHandler* pH);

private:

    //! The game object itself
    CXGame *m_pGame;

    //! Map of agent combat states
    std::map<int, CAgentCombatState*> m_AgentCombatStates;

    //! Map of breach combat states
    std::map<int, CBreachCombatState*> m_BreachCombatStates;

    //! Map of scout combat states
    std::map<int, CScoutCombatState*> m_ScoutCombatStates;

    //! Flag to indicate whether a cover agent is currently moving up
    bool m_bAgentIsMovingUp;

    //! Handle to the enemy machine gunner state
    CEnemyMGState* m_pEnemyMGState;


};

#endif //_SCRIPTOBJECTBLACKBOARD_H_


//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: ScriptObjectBlackboard.cpp
//  Description: Script wrapper for blackbaord system
//
//      History:
//          04August2007    : File created by Spencer Drayton
//          18August2007    : Added agent combat state management methods
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "IAgent.h"
#include "IAISystem.h"
#include "ScriptObjectBlackboard.h"

_DECLARE_SCRIPTABLEEX(CScriptObjectBlackboard)

CScriptObjectBlackboard::~CScriptObjectBlackboard( void )
{
        // Release the agent (cover) combat state objects from the heap
        if (!m_AgentCombatStates.empty())
        {
                std::map<int, CAgentCombatState*>::iterator iter;
                for (iter = m_AgentCombatStates.begin(); iter != m_AgentCombatStates.end(); iter++)
                        delete iter->second;
        }

        // Release the breach combat state objects from the heap
        if (!m_BreachCombatStates.empty())
        {
                std::map<int, CBreachCombatState*>::iterator iter;
                for (iter = m_BreachCombatStates.begin(); iter != m_BreachCombatStates.end(); iter++)
                        delete iter->second;
        }

        // Release the scout combat state objects from the heap
        if (!m_ScoutCombatStates.empty())
        {
                std::map<int, CScoutCombatState*>::iterator iter;
                for (iter = m_ScoutCombatStates.begin(); iter != m_ScoutCombatStates.end(); iter++)
                        delete iter->second;
        }

        // Release enemy machine gunner state
        if (m_pEnemyMGState != NULL)
         delete m_pEnemyMGState;
}

void CScriptObjectBlackboard::Init(IScriptSystem *pScriptSystem, CXGame* pGame)
{
    m_pGame = pGame;

    //! Enemy machine gunner state
    m_pEnemyMGState = new CEnemyMGState(0);

    // Causes this script object to be globally accessible in all LUA scripts using the
    // namespace BlackboardObject
    InitGlobal(pScriptSystem, "BlackboardObject", this);
}

void CScriptObjectBlackboard::InitializeTemplate(IScriptSystem *pSS)
{
    _ScriptableEx<CScriptObjectBlackboard>::InitializeTemplate(pSS);

    // Register the functions we want to be accessible from LUA
    REG_FUNC(CScriptObjectBlackboard, PrintTestMessage);

    // Covers:
    REG_FUNC(CScriptObjectBlackboard, addCoverCombatState);
    REG_FUNC(CScriptObjectBlackboard, removeCoverCombatState);
    REG_FUNC(CScriptObjectBlackboard, setCoverElapsedTime);
    REG_FUNC(CScriptObjectBlackboard, getCoverElapsedTime);
    REG_FUNC(CScriptObjectBlackboard, canCoverMoveUp);
    REG_FUNC(CScriptObjectBlackboard, coverFinishedMovingUp);
    REG_FUNC(CScriptObjectBlackboard, getNumCovers);
    REG_FUNC(CScriptObjectBlackboard, getAllCoversInPosn);
    REG_FUNC(CScriptObjectBlackboard, getCoverInPosn);
    REG_FUNC(CScriptObjectBlackboard, setCoverInPosn);
    REG_FUNC(CScriptObjectBlackboard, addCoverEnemySighting);
    REG_FUNC(CScriptObjectBlackboard, setCoverEnemyAlive);
    REG_FUNC(CScriptObjectBlackboard, getCoverEnemyIDs);
    REG_FUNC(CScriptObjectBlackboard, setCoverDestination);
    REG_FUNC(CScriptObjectBlackboard, getCoverDestination);

    // Scouts:
    REG_FUNC(CScriptObjectBlackboard, addScoutCombatState);
    REG_FUNC(CScriptObjectBlackboard, removeScoutCombatState);
    REG_FUNC(CScriptObjectBlackboard, addScoutEnemySighting);
    REG_FUNC(CScriptObjectBlackboard, addScoutEnemySighting);
    REG_FUNC(CScriptObjectBlackboard, addScoutTargetSighting);
    REG_FUNC(CScriptObjectBlackboard, getScoutTargetSighting);
    REG_FUNC(CScriptObjectBlackboard, setScoutDestination);
    REG_FUNC(CScriptObjectBlackboard, getScoutDestination);

    // Enemy gunners:
    REG_FUNC(CScriptObjectBlackboard, setGunnerID);
    REG_FUNC(CScriptObjectBlackboard, getGunnerID);
    REG_FUNC(CScriptObjectBlackboard, gunnerDied);
    REG_FUNC(CScriptObjectBlackboard, gunnerAssigned);

    // Breaches:
    REG_FUNC(CScriptObjectBlackboard, addBreachCombatState);
    REG_FUNC(CScriptObjectBlackboard, removeBreachCombatState);
    REG_FUNC(CScriptObjectBlackboard, setBreachInPosn);
    REG_FUNC(CScriptObjectBlackboard, getBreachInPosn);
    REG_FUNC(CScriptObjectBlackboard, getAllBreachesInPosn);
    REG_FUNC(CScriptObjectBlackboard, setBreachDestination);
    REG_FUNC(CScriptObjectBlackboard, getBreachDestination);

    // Also initialise any global values
    pSS->SetGlobalValue("GVAR_TEST", 0);
}
// script functions

int CScriptObjectBlackboard::PrintTestMessage(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // NB: to check number of parameters passed, use:
    // pH->GetParamCount()
    // To check that the paramater passed is a string , use:
    // pH->GetParamType(1) == svtString

    // Get the string
    const char *szMessage = 0;
    pH->GetParam(1, szMessage);

    if (!szMessage)
        return pH->EndFunction();

    // Print the string to the console
    char szMsgLine[64] = {0};
    sprintf(szMsgLine, "Message = \"%s\"", szMessage);
    m_pGame->GetSystem()->GetIConsole()->PrintLine(szMsgLine);

    // Print the string to the log
    m_pGame->m_pLog->Log("Message= \"%s\"", szMessage);

    // NB: To call this function from LUA, use:
    // BlackboardObject:PrintTestMessage("This is a test message");

    // To build a table, set some values then return it to the script, use:
    // _SmartScriptObject pObj(m_pScriptSystem);
    // pObj->SetValue("TestNumber", nValueID);
    // return pH->EndFunction(*pObj);

    return pH->EndFunction();

}

//************************************
// Cover Agent Combat State Management
//************************************

int CScriptObjectBlackboard::setCoverInPosn(IFunctionHandler* pH)
{
    // Check that only two parameters were passed in
    CHECK_PARAMETERS(2);

    // Get the id
    int coverID;
    pH->GetParam(1, coverID);

    // Get the flag
    bool flag;
    pH->GetParam(2, flag);

    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->setInPosn(flag);

        // Log the call
        m_pGame->m_pLog->Log("Updated combat state (in posn) for cover id: \"%d\"", coverID);

    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getCoverInPosn(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);
    bool result = false;

    // Get the id
    int coverID;
    pH->GetParam(1, coverID);

    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        result = iter->second->getInPosn();

        // Log the call
        m_pGame->m_pLog->Log("Got combat state (in Posn) for cover id: \"%d\"", coverID);

    }
    return pH->EndFunction(result);
}

int CScriptObjectBlackboard::getAllCoversInPosn(IFunctionHandler* pH)
{
    bool allInPosn = false;

    // Iterate through all agents
    std::map<int, CAgentCombatState*>::iterator iter;
    if (!(m_AgentCombatStates.empty()))
    {
        int numInPosn = 0;
        for (iter = m_AgentCombatStates.begin(); iter != m_AgentCombatStates.end(); iter++)
        {
            if (iter->second->getInPosn())
            {
                numInPosn++;
            }
        }
        allInPosn = (numInPosn == m_AgentCombatStates.size());
    }
    int retVal = allInPosn ? 1 : 0;
    m_pGame->m_pLog->Log("Returning getAllCoversInPosn: \"%d\"", retVal);
    return pH->EndFunction(allInPosn);
}

int CScriptObjectBlackboard::addCoverCombatState(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int agentID;
    pH->GetParam(1, agentID);

    // See if a combat state with this key already exists; if so, erase it
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        delete iter->second;
        m_AgentCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Added combat state for cover id: \"%d\"", agentID);
    }

    // Add the newly created combat state
    CAgentCombatState* pCombatState = new CAgentCombatState(agentID, 0.0, false);
    m_AgentCombatStates.insert(std::make_pair(agentID, pCombatState));

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::removeCoverCombatState(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int agentID;
    pH->GetParam(1, agentID);
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        delete iter->second;
        m_AgentCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Deleted combat state for cover id: \"%d\"", agentID);

    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::setCoverElapsedTime(IFunctionHandler* pH)
{
    // Check that only two parameters were passed in
    CHECK_PARAMETERS(2);

    // Get the id
    int agentID;
    pH->GetParam(1, agentID);

    // Get the time
    float time;
    pH->GetParam(2, time);

    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->setElapsedTime(time);

        // Log the call
        m_pGame->m_pLog->Log("Updated combat state (elapsed time) for cover id: \"%d\"", agentID);

    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getCoverElapsedTime(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);
    float result = 0.0;

    // Get the id
    int agentID;
    pH->GetParam(1, agentID);

    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        result = iter->second->getElapsedTime();

        // Log the call
        m_pGame->m_pLog->Log("Got combat state (elapsed time) for cover id: \"%d\"", agentID);

    }
    return pH->EndFunction(result);
}

int CScriptObjectBlackboard::canCoverMoveUp(IFunctionHandler* pH)
{
    bool canMoveUp = false;

    // Check that only two parameters were passed in
    CHECK_PARAMETERS(2);

    // Get the id
    int agentID;
    pH->GetParam(1, agentID);

    // Get the time
    float time;
    pH->GetParam(2, time);

    // Set the calling agents elapased time
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->setElapsedTime(time);
    }

    // Can't move up if an agent is already moving up, or if we have no data
    if ((m_bAgentIsMovingUp) || (m_AgentCombatStates.empty()))
        return pH->EndFunction(canMoveUp);


    float maxTime = 0.0;
    int maxID = 0;

    for (iter = m_AgentCombatStates.begin(); iter != m_AgentCombatStates.end(); iter++)
       {
        float currTime = iter->second->getElapsedTime();
        m_pGame->m_pLog->Log("Current time is: \"%f\"", currTime);
               if ( currTime > maxTime)
               {
            maxTime = currTime;
            maxID = iter->first;
        }
    }
    m_pGame->m_pLog->Log("Max time is: \"%f\"", maxTime);

    // If the agent with the longest elapsed time made the request, then
    // let it move up, otherwise update is elapsed time.
    if (maxID == agentID)
    {
        m_bAgentIsMovingUp = true;
        canMoveUp = true;

        // Log the call
        m_pGame->m_pLog->Log("Letting cover move up, id: \"%d\"", agentID);
    }

    return pH->EndFunction(canMoveUp);
}

int CScriptObjectBlackboard::coverFinishedMovingUp(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);
    float result = 0.0;

    // Get the id
    int agentID;
    pH->GetParam(1, agentID);

    // Reset the elapsed time
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(agentID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->setElapsedTime(0.0);

        // Log the call
        m_pGame->m_pLog->Log("Cover finished moving up, id: \"%d\"", agentID);
    }

    // Reset the flag
    m_bAgentIsMovingUp = false;
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getNumCovers(IFunctionHandler* pH)
{
    int result = m_AgentCombatStates.size();
    return pH->EndFunction(result);
}

int CScriptObjectBlackboard::addCoverEnemySighting(IFunctionHandler* pH)
{
    // expecting : int coverID, const int enemyID, const Vec3 vPosition, const bool alive
    CHECK_PARAMETERS(4);

    // Get the coverID
    int coverID;
    pH->GetParam(1, coverID);

    // Get the enemyID
    int enemyID;
    pH->GetParam(2, enemyID);

    // get the enemy position
    Vec3 vPosn(0,0,0);
    CScriptObjectVector vPosition(m_pScriptSystem,true);
    if(pH->GetParam(3, *vPosition))
    {
        vPosn = vPosition.Get();
    }

    // get the enemy 'alive' flag
    int bAlive;
    pH->GetParam(4, bAlive);

    // Find the specified cover, then add a new enemy sighting
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->addEnemySighting(enemyID, vPosn, bAlive);

        // Log the call
        m_pGame->m_pLog->Log("Added enemy sighting for cover, id: \"%d\"", coverID);
    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::setCoverEnemyAlive(IFunctionHandler* pH)
{
    // int coverID, const int enemyID, const bool alive
    CHECK_PARAMETERS(3);

    // Get the coverID
    int coverID;
    pH->GetParam(1, coverID);

    // Get the enemyID
    int enemyID;
    pH->GetParam(2, enemyID);

    // get the enemy 'alive' flag
    int bAlive;
    pH->GetParam(3, bAlive);

    // Find the specified cover, then add a new enemy sighting
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->setEnemyAlive(enemyID, bAlive);

        // Log the call
        m_pGame->m_pLog->Log("Updated enemy sighting for cover, id: \"%d\"", coverID);
    }

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getCoverEnemyIDs(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(1);

    // Get the coverID
    int coverID;
    pH->GetParam(1, coverID);


    // Find the specified cover, then get the enemy IDs
    std::vector<int> enemyIDs;
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->getEnemyIDs(enemyIDs);
        // Log the call
        m_pGame->m_pLog->Log("Got enemy IDs for cover, id: \"%d\"", coverID);
    }

   // Convert the vector of tagpoint names and return a LUA table
    _SmartScriptObject cList(m_pGame->m_pSystem->GetIScriptSystem(), false);
       for (std::vector<int>::iterator it = enemyIDs.begin(); it != enemyIDs.end(); ++it)
       {
              cList->SetAt(cList->Count()+1, *it);
       }

    return pH->EndFunction(cList);
}

int CScriptObjectBlackboard::setCoverDestination(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(2);

    // Get the coverID
    int coverID;
    pH->GetParam(1, coverID);

    // Get the destination string
    const char *szDest = 0;
    pH->GetParam(2, szDest);
    string strDest(szDest);

    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        iter->second->setDestination(strDest);

        // Log the call
        m_pGame->m_pLog->Log("Set destination for cover, id: \"%d\"", coverID);
    }

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getCoverDestination(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(1);

    // Get the coverID
    int coverID;
    pH->GetParam(1, coverID);

    // Get the destination string
    string strDest;
    std::map<int, CAgentCombatState*>::iterator iter = m_AgentCombatStates.find(coverID);
    if (iter != m_AgentCombatStates.end())
    {
        strDest = iter->second->getDestination();

        // Log the call
        m_pGame->m_pLog->Log("Got destination for cover, id: \"%d\"", coverID);
    }

    return pH->EndFunction(strDest.c_str());
}

//********************************
// Scout Combat State Management
//********************************

//! Add a new breach combat state
int CScriptObjectBlackboard::addScoutCombatState(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int scoutID;
    pH->GetParam(1, scoutID);

    // See if a combat state with this key already exists; if so, erase it
    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        delete iter->second;
        m_ScoutCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Added combat state for scout id: \"%d\"", scoutID);
    }

    // Add the newly created combat state
    CScoutCombatState* pCombatState = new CScoutCombatState(scoutID, false);
    m_ScoutCombatStates.insert(std::make_pair(scoutID, pCombatState));

    return pH->EndFunction(1);
}

//! Remove an existing breach combat state
int CScriptObjectBlackboard::removeScoutCombatState(IFunctionHandler* pH)
{
   // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int scoutID;
    pH->GetParam(1, scoutID);

    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        delete iter->second;
        m_ScoutCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Deleted combat state for scout id: \"%d\"", scoutID);

    }
    return pH->EndFunction(1);
}

//! Adds to the list of enemies that have been spotted
int CScriptObjectBlackboard::addScoutEnemySighting(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(4);

    // Get the scoutID
    int scoutID;
    pH->GetParam(1, scoutID);

    // Get the enemyID
    int enemyID;
    pH->GetParam(2, enemyID);

    // get the enemy position
    Vec3 vPosn(0,0,0);
    CScriptObjectVector vPosition(m_pScriptSystem,true);
    if(pH->GetParam(3, *vPosition))
    {
        vPosn = vPosition.Get();
    }

    // get the enemy 'alive' flag
    int bAlive;
    pH->GetParam(4, bAlive);

    // Find the specified scout, then add a new enemy sighting
    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        iter->second->addEnemySighting(enemyID, vPosn, bAlive);

        // Log the call
        m_pGame->m_pLog->Log("Added enemy sighting for scout, id: \"%d\"", scoutID);
    }
    return pH->EndFunction(1);
}

//! Adds details of main target sighting
```

```
int CScriptObjectBlackboard::addScoutTargetSighting(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(2);

    // Get the scoutID
    int scoutID;
    pH->GetParam(1, scoutID);

    // get the target position
    Vec3 vPosn(0,0,0);
    CScriptObjectVector vPosition(m_pScriptSystem,true);
    if(pH->GetParam(2, *vPosition))
    {
        vPosn = vPosition.Get();
    }

    // Find the specified scout, then get the target sighting
    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        iter->second->addTargetSighting(vPosn);

        // Log the call
        m_pGame->m_pLog->Log("Set target sighting for scout, id: \"%d\"", scoutID);
    }

    return pH->EndFunction(1);
}

//! Gets the position of the main target sighting
int CScriptObjectBlackboard::getScoutTargetSighting(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(1);

    // Get the scoutID
    int scoutID;
    pH->GetParam(1, scoutID);

    // Find the specified scout, then get the target sighting
    CScriptObjectVector vPosition(m_pScriptSystem,true);
    Vec3 targetPosn;
    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        targetPosn = iter->second->getTargetSighting();
        vPosition.Set(targetPosn);

        // Log the call
        m_pGame->m_pLog->Log("Got target sighting from scout, id: \"%d\"", scoutID);
    }

    return pH->EndFunction(vPosition);

}

int CScriptObjectBlackboard::setScoutDestination(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(2);

    // Get the scoutID
    int scoutID;
    pH->GetParam(1, scoutID);

    // Get the destination string
    const char *szDest = 0;
    pH->GetParam(2, szDest);
    string strDest(szDest);

    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        iter->second->setDestination(strDest);

        // Log the call
        m_pGame->m_pLog->Log("Set destination for scout, id: \"%d\"", scoutID);
    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getScoutDestination(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(1);

    // Get the scoutID
    int scoutID;
    pH->GetParam(1, scoutID);

    // Get the destination string
    string strDest;
    std::map<int, CScoutCombatState*>::iterator iter = m_ScoutCombatStates.find(scoutID);
    if (iter != m_ScoutCombatStates.end())
    {
        strDest = iter->second->getDestination();

        // Log the call
        m_pGame->m_pLog->Log("Got destination for scout, id: \"%d\"", scoutID);
    }

    return pH->EndFunction(strDest.c_str());
}

//**************************************
// Enemy Machine Gunner State Management
//**************************************

int CScriptObjectBlackboard::setGunnerID(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the id
    int gunnerID;
    pH->GetParam(1, gunnerID);

    m_pEnemyMGState->setGunnerID(gunnerID);

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getGunnerID(IFunctionHandler* pH)
{
    return pH->EndFunction(m_pEnemyMGState->getGunnerID());
}

int CScriptObjectBlackboard::gunnerDied(IFunctionHandler* pH)
{
    m_pEnemyMGState->gunnerDied();

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::gunnerAssigned(IFunctionHandler* pH)
{
    bool ga = m_pEnemyMGState->gunnerAssigned();
    return pH->EndFunction(ga);
}

//********************************
// Breach State Management
//********************************

int CScriptObjectBlackboard::setBreachDestination(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(2);

    // Get the breachID
    int breachID;
    pH->GetParam(1, breachID);

    // Get the destination string
    const char *szDest = 0;
    pH->GetParam(2, szDest);
    string strDest(szDest);
    std::map<int, CBreachCombatState*>::iterator iter = m_BreachCombatStates.find(breachID);
    if (iter != m_BreachCombatStates.end())
    {
        iter->second->setDestination(strDest);

        // Log the call
        m_pGame->m_pLog->Log("Set destination for breach, id: \"%d\"", breachID);
    }

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getBreachDestination(IFunctionHandler* pH)
{
    CHECK_PARAMETERS(1);

    // Get the breachID
    int breachID;
    pH->GetParam(1, breachID);

    // Get the destination string
    string strDest;
    std::map<int, CBreachCombatState*>::iterator iter = m_BreachCombatStates.find(breachID);
    if (iter != m_BreachCombatStates.end())
    {
        strDest = iter->second->getDestination();

        // Log the call
        m_pGame->m_pLog->Log("Got destination for breach, id: \"%d\"", breachID);
    }

    return pH->EndFunction(strDest.c_str());
}

int CScriptObjectBlackboard::getAllBreachesInPosn(IFunctionHandler* pH)
{
    bool allInPosn = false;

    // Iterate through all agents
    std::map<int, CBreachCombatState*>::iterator iter;
    if (!(m_BreachCombatStates.empty()))
    {
        int numInPosn = 0;
        for (iter = m_BreachCombatStates.begin(); iter != m_BreachCombatStates.end(); iter++)
        {
            if (iter->second->getInPosn())
            {
                numInPosn++;
            }
        }
        allInPosn = (numInPosn == m_BreachCombatStates.size());
    }
    // Log the call
    int retVal = allInPosn ? 1 : 0;
    m_pGame->m_pLog->Log("Returning getAllBreachesInPosn: \"%d\"", retVal);
    return pH->EndFunction(allInPosn);
}

int CScriptObjectBlackboard::addBreachCombatState(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int agentID;
    pH->GetParam(1, agentID);

    // See if a combat state with this key already exists; if so, erase it
    std::map<int, CBreachCombatState*>::iterator iter = m_BreachCombatStates.find(agentID);
    if (iter != m_BreachCombatStates.end())
    {
        delete iter->second;
        m_BreachCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Added combat state for breach id: \"%d\"", agentID);
    }

    // Add the newly created combat state
    CBreachCombatState* pCombatState = new CBreachCombatState(agentID, false);
    m_BreachCombatStates.insert(std::make_pair(agentID, pCombatState));

    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::removeBreachCombatState(IFunctionHandler* pH)
{
    // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);

    // Get the integer
    int agentID;
    pH->GetParam(1, agentID);

    std::map<int, CBreachCombatState*>::iterator iter = m_BreachCombatStates.find(agentID);
    if (iter != m_BreachCombatStates.end())
    {
        delete iter->second;
        m_BreachCombatStates.erase(iter);

        // Log the call
        m_pGame->m_pLog->Log("Deleted combat state for breach id: \"%d\"", agentID);

    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::setBreachInPosn(IFunctionHandler* pH)
{
    // Check that only two parameters were passed in
    CHECK_PARAMETERS(2);

    // Get the id
    int breachID;
    pH->GetParam(1, breachID);

    // Get the flag
    bool flag;
    pH->GetParam(2, flag);

    std::map<int, CBreachCombatState*>::iterator iter = m_BreachCombatStates.find(breachID);
    if (iter != m_BreachCombatStates.end())
    {
        iter->second->setInPosn(flag);

        // Log the call
        m_pGame->m_pLog->Log("Updated combat state (in posn) for breach id: \"%d\"", breachID);

    }
    return pH->EndFunction(1);
}

int CScriptObjectBlackboard::getBreachInPosn(IFunctionHandler* pH)
{
   // Check that only one parameter was passed in
    CHECK_PARAMETERS(1);
    bool result = false;

    // Get the id
    int breachID;
    pH->GetParam(1, breachID);

    std::map<int, CBreachCombatState*>::iterator iter = m_BreachCombatStates.find(breachID);
    if (iter != m_BreachCombatStates.end())
    {
        result = iter->second->getInPosn();

        // Log the call
        m_pGame->m_pLog->Log("Got combat state (in Posn) for breach id: \"%d\"", breachID);

    }
    return pH->EndFunction(result);
}
```

```
//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: AgentCombatState.h
//  Description: Abstraction of the state of a single agent
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#ifndef _AGENTCOMBATSTATE_H_
#define _AGENTCOMBATSTATE_H_

#include <map>
#include <vector>
#include "EnemySighting.h"

class CAgentCombatState
{
public:

        //! Constructors
        CAgentCombatState( void ) : m_nId(0), m_fElapsedTime(0.0), m_bInPosn(false)  {}

    CAgentCombatState( int id, float time, bool flag ) : m_nId(id), m_fElapsedTime(time),
m_bInPosn(flag) {}

        //! Destructor
        virtual ~CAgentCombatState( void );

        //! Sets the id of the agent associated with this state object
    void setID(const int id);

        //! Gets the id of the agent associated with this state object
    int getID();

        //! Sets the elapsed time in position of this agent
    void setElapsedTime(const float time);

        //! Gets the elapsed time in position of this agent
    float getElapsedTime();

    //! Sets the 'in position' state of the agent
    void setInPosn(const bool flag);

    //! Gets the 'in position' state of the agent
    int getInPosn();

    //! Adds to the list of enemies that have been spotted
    void addEnemySighting(const int id, const Vec3 vPosition, const bool alive);

    //! Sets the 'alive' state of an enemy
    void setEnemyAlive(const int id, const bool alive);

    //! Returns a list of enemy IDs
    void getEnemyIDs(std::vector<int>& enemyIDList);

    //! Set this agents current destination (tagpoint name)
    void setDestination(const string name);

    //! Get this agents current destination (tagpoint name)
    string getDestination();

private:

    //! The agent id
    int m_nId;

    //! The elapsed time
    float m_fElapsedTime;

    //! The 'in position' state
    bool m_bInPosn;
    //! The list of enemy sightings
    std::map<int, CEnemySighting*> m_enemySightings;

    //! The name of the current detsination (tagpoint)
    string m_destination;
};


#endif //_AGENTCOMBATSTATE_H_

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: AgentCombatState.h
//  Description: Abstraction of the state of a single agent
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "AgentCombatState.h"
#include "EnemySighting.h"

CAgentCombatState::~CAgentCombatState( void )
{
    // Release the enemy sightings objects from the heap
    if (!m_enemySightings.empty())
        {
                std::map<int, CEnemySighting*>::iterator iter;
                for (iter = m_enemySightings.begin(); iter != m_enemySightings.end(); iter++)
                        delete iter->second;
        }
}

void CAgentCombatState::setID(const int id)
{
    this->m_nId = id;
}

int CAgentCombatState::getID()
{
    return this->m_nId;
}

void CAgentCombatState::setElapsedTime(const float time)
{
    this->m_fElapsedTime = time;
}

float CAgentCombatState::getElapsedTime()
{
   return this->m_fElapsedTime;
}

void CAgentCombatState::setInPosn(const bool flag)
{
    this->m_bInPosn = flag;
}

int CAgentCombatState::getInPosn()
{
    return this->m_bInPosn;
}

void CAgentCombatState::addEnemySighting(const int id, const Vec3 vPosition, const bool alive)
{
    // See if an enemy sighting with this key already exists; if so, erase it
    std::map<int, CEnemySighting*>::iterator iter = m_enemySightings.find(id);
    if (iter != m_enemySightings.end())
    {
        delete iter->second;
        m_enemySightings.erase(iter);
    }

    // Add the newly created enemy sighting
    CEnemySighting* pEnemySighting = new CEnemySighting(id, vPosition, alive);
    m_enemySightings.insert(std::make_pair(id, pEnemySighting));
}

void CAgentCombatState::setEnemyAlive(const int id, const bool alive)
{
    // See if an enemy sighting with this key already exists; if so, erase it
    std::map<int, CEnemySighting*>::iterator iter = m_enemySightings.find(id);
    if (iter != m_enemySightings.end())
    {
        iter->second->setEnemyAlive(alive);
    }
}

void CAgentCombatState::getEnemyIDs(std::vector<int>& enemyIDList)
{
    enemyIDList.clear();
    std::map<int, CEnemySighting*>::iterator iter;
    for (iter = m_enemySightings.begin(); iter != m_enemySightings.end(); iter++)
    {
        enemyIDList.push_back(iter->first);
    }
}

void CAgentCombatState::setDestination(const string destn)
{
    m_destination = destn;
}

string CAgentCombatState::getDestination()
{
    return m_destination;
}

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: AgentCombatState.h
//  Description: Abstraction of the state of a single agent
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "AgentCombatState.h"
#include "EnemySighting.h"

CAgentCombatState::~CAgentCombatState( void )
{
    // Release the enemy sightings objects from the heap
    if (!m_enemySightings.empty())
        {
                std::map<int, CEnemySighting*>::iterator iter;
                for (iter = m_enemySightings.begin(); iter != m_enemySightings.end(); iter++)
                        delete iter->second;
        }
}

void CAgentCombatState::setID(const int id)
{
    this->m_nId = id;
}

int CAgentCombatState::getID()
{
    return this->m_nId;
}

void CAgentCombatState::setElapsedTime(const float time)
{
    this->m_fElapsedTime = time;
```

```
}

float CAgentCombatState::getElapsedTime()
{
   return this->m_fElapsedTime;
}

void CAgentCombatState::setInPosn(const bool flag)
{
    this->m_bInPosn = flag;
}

int CAgentCombatState::getInPosn()
{
    return this->m_bInPosn;
}

void CAgentCombatState::addEnemySighting(const int id, const Vec3 vPosition, const bool alive)
{
    // See if an enemy sighting with this key already exists; if so, erase it
    std::map<int, CEnemySighting*>::iterator iter = m_enemySightings.find(id);
    if (iter != m_enemySightings.end())
    {
        delete iter->second;
        m_enemySightings.erase(iter);

    }

    // Add the newly created enemy sighting
    CEnemySighting* pEnemySighting = new CEnemySighting(id, vPosition, alive);
    m_enemySightings.insert(std::make_pair(id, pEnemySighting));
}

void CAgentCombatState::setEnemyAlive(const int id, const bool alive)
{
    // See if an enemy sighting with this key already exists; if so, erase it
    std::map<int, CEnemySighting*>::iterator iter = m_enemySightings.find(id);
    if (iter != m_enemySightings.end())
    {
        iter->second->setEnemyAlive(alive);
    }
}

void CAgentCombatState::getEnemyIDs(std::vector<int>& enemyIDList)
{
    enemyIDList.clear();
    std::map<int, CEnemySighting*>::iterator iter;
    for (iter = m_enemySightings.begin(); iter != m_enemySightings.end(); iter++)
    {
        enemyIDList.push_back(iter->first);
    }
}

void CAgentCombatState::setDestination(const string destn)
{
    m_destination = destn;
}

string CAgentCombatState::getDestination()
{
    return m_destination;
}

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: BreachCombatState.cpp
//  Description: Abstraction of the state of a breach agent
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "BreachCombatState.h"
```

```
void CBreachCombatState::setBreachID(const int id)
{
    this->m_nBreachId = id;
}

int CBreachCombatState::getBreachID()
{
    return this->m_nBreachId;
}

void CBreachCombatState::setInPosn(const bool flag)
{
    this->m_bInPosn = flag;
}

int CBreachCombatState::getInPosn()
{
    return this->m_bInPosn;
}

void CBreachCombatState::setDestination(const string destn)
{
    m_destination = destn;
}

string CBreachCombatState::getDestination()
{
    return m_destination;
}

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: EnemyMGState.h
//  Description: Abstraction of the state of an enemy machine gunner
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#ifndef _ENEMYMGSTATE_H_
#define _ENEMYMGSTATE_H_

class CEnemyMGState
{
public:

        //! Constructors
        CEnemyMGState( void ) : m_nCurrentGunnerId(0) {}

    CEnemyMGState( int id) : m_nCurrentGunnerId(id){}

        //! Destructor
        virtual ~CEnemyMGState( void ) {}

        //! Sets the id of the current gunner
    void setGunnerID(const int id);

        //! Gets the id of the current gunner
    int getGunnerID();

        //! Resets the gunner state
    void gunnerDied();

        //! Gets whether a gunner is currently assigned
    bool gunnerAssigned();

private:

    //! The agent id
    int m_nCurrentGunnerId;
};
```

```
#endif //_ENEMYMGSTATE_H_

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: EnemyMGState.cpp
//  Description: Abstraction of the state of an enemy machine gunner
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "EnemyMGState.h"

void CEnemyMGState::setGunnerID(const int id)
{
    this->m_nCurrentGunnerId = id;
}

int CEnemyMGState::getGunnerID()
{
    return this->m_nCurrentGunnerId;
}

void CEnemyMGState::gunnerDied()
{
    this->m_nCurrentGunnerId = 0;
}

bool CEnemyMGState::gunnerAssigned()
{
    return this->m_nCurrentGunnerId != 0;
}

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: EnemySighting.h
//  Description: Abstraction of an enemy sighting record
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#ifndef _ENEMYSIGHTING_H_
#define _ENEMYSIGHTING_H_

class CEnemySighting
{
public:

        //! Constructors
        CEnemySighting( void ) : m_nEnemyId(0), m_position(Vec3(0.0, 0.0, 0.0)), m_bAlive(false) {}

    CEnemySighting( int id, Vec3 posn, bool alive) : m_nEnemyId(id), m_position(posn),
m_bAlive(alive) {}

        //! Destructor
        virtual ~CEnemySighting( void ) {}

    //! Sets the 'alive' state of an enemy
    void setEnemyAlive(const bool alive);

    //! Gets the 'alive' state of an enemy
    bool getEnemyAlive();

    //! Sets the position of an enemy
    void setPosition(const Vec3 posn);

    //! gets the position of an enemy
    Vec3 getPosition();
private:

    //! The enemy system id
    int m_nEnemyId;

    //! The last known position
    Vec3 m_position;

    //! The last known 'alive' status
    bool m_bAlive;
};


#endif //_ENEMYSIGHTING_H_

//////////////////////////////////////////////////////////////////////
//
//      SJD Source code
//
//      File: EnemySighting.cpp
//  Description: Abstraction of an enemy sighting record
//
//      History:
//          18August2007: File created
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "EnemySighting.h"

void CEnemySighting::setEnemyAlive(const bool alive)
{
    m_bAlive = alive;
}

bool CEnemySighting::getEnemyAlive()
{
    return m_bAlive;
}

void CEnemySighting::setPosition(const Vec3 posn)
{
    m_position = posn;
}

Vec3 CEnemySighting::getPosition()
{
    return m_position;
}

//////////////////////////////////////////////////////////////////////
//
//      Crytek Source code
//      Copyright (c) Crytek 2001-2004
//
//      File: Game.h
//  Description: Interface and definitions for the game class.
//
//      History:
//      - August 02,2001: Created by Marco Corbetta and Alberto Demichelis
//      - Sep 24,2001 : Modified by Petar Kotevski
//      - February 2005: Modified by Marco Corbetta for SDK release
//      - October 2006: Modified by Marco Corbetta for SDK 1.4 release
//      - August 2007: Modified by S J Drayton for Msc  project
//
//////////////////////////////////////////////////////////////////////

...

// SJD MOD -->
void GetTagPointsInRadius(std::vector<string>& tagPointNames, Vec3& position, float radius, float
tolerance);

void GetHighestTagPointInRadius(string& tagPointName, Vec3& position, float radius, float
tolerance);
// --> SJD MOD
```

```
...

//////////////////////////////////////////////////////////////////////
//
//      Crytek Source code
//      Copyright (c) Crytek 2001-2004
//
//      File: Game.cpp
//  Description: Implementation of general game functions
//
//  History:
//  - 08/08/2001: Created by Marco Corbetta and Alberto Demichelis
//  - 09/24/2001: Modified by Petar Kotevski
//  - 12/14/2003: Martin Mittring made ClassID from unsigned char(8bit) to EntityClassId (16bit))
//  - 27/04/2004: cleanup by Mathieu Pinard
//      - February 2005: Modified by Marco Corbetta for SDK release
//      - August 2007: Modified by S J Drayton for Msc  project
//
//////////////////////////////////////////////////////////////////////

bool CXGame::Init(struct ISystem *pSystem,bool bDedicatedSrv,bool bInEditor,const char *szGameMod)
{

...

// SJD MOD->
        m_pScriptObjectBlackboard = new CScriptObjectBlackboard;
        CScriptObjectBlackboard::InitializeTemplate(m_pScriptSystem);
        m_pScriptObjectBlackboard->Init(m_pScriptSystem, this);
// ->SJD MOD-

...

}

//////////////////////////////////////////////////////////////////////
//
//      Crytek Source code
//      Copyright (c) Crytek 2001-2004
//
//      File: GameTagPoints.cpp
//      Description: Editor/Game tag points.
//
//      History:
//      - December 11,2001: File created
//      - October      31,2003: Merged from Game.cpp and other files
//      - February 2005: Modified by Marco Corbetta for SDK release
//      - August 2007: Modified by S J Drayton for Msc  project
//
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// SJD MOD -->

void CXGame::GetTagPointsInRadius(std::vector<string>& tagPointNames, Vec3& position, float radius,
float tolerance)
{
    // Iterate through all tagpoints and find the ones which match the
    // supplied criteria
        if (!m_mapTagPoints.empty())
        {
            tagPointNames.clear();
            float halfTol = tolerance/2.0;
                TagPointMap::iterator ti;
                for (ti=m_mapTagPoints.begin(); ti!=m_mapTagPoints.end(); ti++)
         {
             // Get the position of the current tagpoint
                       Vec3 currTagPosn;
                       ti->second->GetPos(currTagPosn);

                       // Compute distance from target to current tagpoint
                       float distance = GetDistance(position, currTagPosn);
                       if ((distance > (radius-halfTol)) && (distance < (radius+halfTol)))
                       {
                  tagPointNames.push_back(ti->first);
                       }
         }
        }
}

void CXGame::GetHighestTagPointInRadius(string& tagPointName, Vec3& position, float radius, float
tolerance)
{
    // Iterate through all tagpoints and find the ones which match the
    // supplied criteria
    string highestTagPointName;
    float maxHeight = 0.0;
        if (!m_mapTagPoints.empty())
        {
            float halfTol = tolerance/2.0;
                TagPointMap::iterator ti;
                for (ti=m_mapTagPoints.begin(); ti!=m_mapTagPoints.end(); ti++)
         {
             // Get the position of the current tagpoint
                         Vec3 currTagPosn;
                         ti->second->GetPos(currTagPosn);

                         // Compute distance from target to current tagpoint
                         float distance = GetDistance(position, currTagPosn);
                         if ((distance > (radius-halfTol)) && (distance < (radius+halfTol)))
                         {
                             if (currTagPosn.z > maxHeight)
                             {
                                 maxHeight = currTagPosn.z;
                                 highestTagPointName = ti->first;
                             }
                         }
         }
         tagPointName = highestTagPointName;
        }

}
//  --> SJD MOD
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
//
//      Crytek Source code
//      Copyright (c) Crytek 2001-2004
//
//      File: ScriptObjectGame.h
//  Description:
//              Interface of the CScriptObjectGame script wrapper.
//              This class implements script-functions for exposing the Game functionalities
//              REMARKS:
//              After initialization of the script-object it will be globally accessable through
//              scripts using the namespace "Game".
//
//              Example:
//              local players=Game.GetPlayers();
//
//              IMPLEMENTATIONS NOTES:
//              These function will never be called from C-Code. They're script-exclusive.
//
//      History:
//      - March 2001: File created
//      - February 2005: Modified by Marco Corbetta for SDK release
//      - October 2006: Modified by Marco Corbetta for SDK 1.4 release
//      - August 2007: Modified by S J Drayton for Msc  project
//
//////////////////////////////////////////////////////////////////////

// SJD MOD -->
    int CreateTagPoint(IFunctionHandler *pH);
    int GetHighestTagPointInRadius(IFunctionHandler *pH);
    int GetTagPointsInRadius(IFunctionHandler *pH);
//  --> SJD MOD

//////////////////////////////////////////////////////////////////////
//
//      Crytek Source code
//      Copyright (c) Crytek 2001-2004
//
//      File: ScriptObjectGame.cpp
//  Description:
//             Implementation of many misc single and multi player game
//             script functions. Heavily modified by all programmers during
//             FC development.
//
//      History:
//      - February 2005: Modified by Marco Corbetta for SDK release
//      - October 2006: Modified by Marco Corbetta for SDK 1.4 release
//
///////////////////////////////////////////////////////////////////////

void CScriptObjectGame::InitializeTemplate(IScriptSystem *pSS)
{

...
       // SJD MOD -->
       REG_FUNC(CScriptObjectGame,CreateTagPoint);
       REG_FUNC(CScriptObjectGame,GetHighestTagPointInRadius);
       REG_FUNC(CScriptObjectGame,GetTagPointsInRadius);
       //  --> SJD MOD
...

}

///////////////////////////////////////////////////////////////////////
// SJD MOD -->
int CScriptObjectGame::CreateTagPoint(IFunctionHandler *pH)
{
    int bSuccess = false;
       CHECK_PARAMETERS(3);

       const char *sTPName;
       Vec3 vPosn(0,0,0);
    Vec3 vAng(0,0,0);
    ITagPoint *pTP=NULL;

    // Extract the tagpoint name
       if(pH->GetParam(1,sTPName))
       {
        bSuccess = true;
       }

    // Extract the tagpoint position
    if (bSuccess)
    {
        CScriptObjectVector vPosition(m_pScriptSystem,true);
        if(pH->GetParam(2, *vPosition))
        {
            vPosn = vPosition.Get();
        }
        else
        {
            bSuccess = false;
        }
    }

    // Extract the tagpoint angle
    if (bSuccess)
    {
        CScriptObjectVector vAngle(m_pScriptSystem,true);
        if(pH->GetParam(3, *vAngle))
        {
            vAng = vAngle.Get();
        }
        else
        {
            bSuccess = false;
        }
    }
    string strName(sTPName);
    m_pSystem->GetILog()->Log("Tag point name :%s", strName.c_str());
    m_pSystem->GetILog()->Log("Tag point x :%f", vPosn.x);
    m_pSystem->GetILog()->Log("Tag point y :%f", vPosn.y);
    m_pSystem->GetILog()->Log("Tag point z :%f", vPosn.z);

    // Create the tagpoint
    if (bSuccess)
    {
        pTP =  m_pGame->CreateTagPoint(strName, vPosn, vAng);
    }

    if (pTP)
    {
        bSuccess = true;
        m_pSystem->GetILog()->Log("Created new tag point");
    }
    else
    {
        bSuccess = false;
        m_pSystem->GetILog()->Log("FAILED to create new tag point");
    }
    return pH->EndFunction(bSuccess);
}

//  --> SJD MOD
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// SJD MOD -->
int CScriptObjectGame::GetTagPointsInRadius(IFunctionHandler *pH)
{
    // We're expecting: pos (Vec3), radius (float), tolerance (float)
    CHECK_PARAMETERS(3);

    // Get the position
    Vec3 vPosn(0,0,0);
    CScriptObjectVector vPosition(m_pScriptSystem,true);
    if(pH->GetParam(1, *vPosition))
    {
        vPosn = vPosition.Get();
    }

    // Get the radius
    float radius;
    pH->GetParam(2, radius);

    // Get the tolerance
    float tolerance;
    pH->GetParam(3, tolerance);

    std::vector<string> tagPointNames;
    m_pGame->GetTagPointsInRadius(tagPointNames, vPosn, radius, tolerance);

    // Convert the vector of tagpoint names and return a LUA table
    _SmartScriptObject cList(m_pSystem->GetIScriptSystem(), false);
        for (std::vector<string>::iterator it = tagPointNames.begin(); it != tagPointNames.end();
++it)
        {
                cList->SetAt(cList->Count()+1, it->c_str());
        }

    return pH->EndFunction(cList);
}
//  --> SJD MOD
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// SJD MOD -->
int CScriptObjectGame::GetHighestTagPointInRadius(IFunctionHandler *pH)
{
    // We're expecting: pos (Vec3), radius (float), tolerance (float)
    CHECK_PARAMETERS(3);

    // Get the position
    Vec3 vPosn(0,0,0);
    CScriptObjectVector vPosition(m_pScriptSystem,true);
    if(pH->GetParam(1, *vPosition))
    {
        vPosn = vPosition.Get();
    }

    // Get the radius
    float radius;
    pH->GetParam(2, radius);
    // Get the tolerance
    float tolerance;
    pH->GetParam(3, tolerance);

    string tagPointName;
    m_pGame->GetHighestTagPointInRadius(tagPointName, vPosn, radius, tolerance);

    // Return the tag point name
    return pH->EndFunction(tagPointName.c_str());
}
//  --> SJD MOD
//////////////////////////////////////////////////////////////////////
```

Appendix F: Lua Behaviour Scripts

```
-----------------------------------------------------------------------
-- AssaultScoutAtObjective
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of scout team in a squad conducting an assault,
-- in 'at objective' state.
-----------------------------------------------------------------------

AIBehaviour.AssaultScoutAtObjective = {
	Name = "AssaultScoutAtObjective",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultScoutAtObjective::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultScoutAtObjective::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultScoutAtObjective::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultScoutAtObjective::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultScoutAtObjective::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultScoutAtObjective::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultScoutAtObjective::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultScoutAtObjective::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultScoutAtObjective::OnEnemyMemory");
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultScoutAtObjective::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultScoutAtObjective::OnThreateningSoundHeard");
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultScoutAtObjective::OnReload");
	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultScoutAtObjective::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultScoutAtObjective::");
	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultScoutAtObjective::OnNoHidingPlace");
	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutAtObjective::OnReceivingDamage");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultScoutAtObjective::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultScoutAtObjective::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutAtObjective::OnDeath");
	end,
	--------------------------------------------------

}--------------------------------------------------------------------
-- AssaultScoutBreaching
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of scout team in a squad conducting an assault,
-- in breaching state.
--------------------------------------------------------------------

AIBehaviour.AssaultScoutBreaching = {
	Name = "AssaultScoutBreaching",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultScoutBreaching::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultScoutBreaching::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultScoutBreaching::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultScoutBreaching::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultScoutBreaching::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultScoutBreaching::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultScoutBreaching::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultScoutBreaching::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultScoutBreaching::OnEnemyMemory");
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultScoutBreaching::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultScoutBreaching::OnThreateningSoundHeard");
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultScoutBreaching::OnReload");
	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultScoutBreaching::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultScoutBreaching::");
	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultScoutBreaching::OnNoHidingPlace");
	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutBreaching::OnReceivingDamage");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultScoutBreaching::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultScoutBreaching::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutBreaching::OnDeath");
	end,
	--------------------------------------------------
}-----------------------------------------------------------------------
-- AssaultScoutIdle
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of scout team in a squad conducting an assault,
-- in idle state.
----------------------------------------------------------------------

AIBehaviour.AssaultScoutIdle = {
	Name = "AssaultScoutIdle",
	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultScoutIdle::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultScoutIdle::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultScoutIdle::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultScoutIdle::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultScoutIdle::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultScoutIdle::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultScoutIdle::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultScoutIdle::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultScoutIdle::OnEnemyMemory");
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultScoutIdle::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultScoutIdle::OnThreateningSoundHeard");
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultScoutIdle::OnReload");
	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultScoutIdle::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultScoutIdle::");
	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultScoutIdle::OnNoHidingPlace");
	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutIdle::OnReceivingDamage");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultScoutIdle::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultScoutIdle::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutIdle::OnDeath");
	end,
	--------------------------------------------------
}-----------------------------------------------------------------------
-- AssaultScoutRallied
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of scout team in a squad conducting an assault,
-- in rallied state.
----------------------------------------------------------------------

AIBehaviour.AssaultScoutRallied = {
	Name = "AssaultScoutRallied",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultScoutRallied::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultScoutRallied::OnSpawn");

		BlackboardObject:addScoutCombatState(entity.id)

		-- Select default behaviour
		entity:SelectPipe(0, "SJDGruntStealthIdle");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultScoutRallied::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultScoutRallied::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultScoutRallied::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultScoutRallied::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultScoutRallied::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultScoutRallied::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultScoutRallied::OnEnemyMemory");
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultScoutRallied::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultScoutRallied::OnThreateningSoundHeard");
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultScoutRallied::OnReload");
	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultScoutRallied::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultScoutRallied::");
	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultScoutRallied::OnNoHidingPlace");
	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutRallied::OnReceivingDamage");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultScoutRallied::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultScoutRallied::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutRallied::OnDeath");
	end,
	--------------------------------------------------

	SJD_SCOUT_GO = function (self, entity, sender)
		local name = "objective";
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			AI:Signal(SIGNALID_READIBILITY, 2, "ORDER_RECEIVED",entity.id);
			entity:SelectPipe(0, "SJDScoutJobApproachObjectiveRapid", name);
		end
	end,

	SJD_ALL_MOVE_TO_OBJECTIVE = function (self, entity, sender)
		local name = "objective";
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:SelectPipe(0, "SJDGruntStealthIdle", name);
			entity:InsertSubpipe(0, "SJDTeamLeadJobBreach", name);
		end
	end,
}
-----------------------------------------------------------------------
-- AssaultScoutScouting
--
-- S J Drayton 14/08/2007
```

```
--
-- Behaviour for member of scout team in a squad conducting an assault,
-- in scouting state.
--------------------------------------------------------------------

AIBehaviour.AssaultScoutScouting = {
	Name = "AssaultScoutScouting",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultScoutScouting::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultScoutScouting::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultScoutScouting::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultScoutScouting::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultScoutScouting::OnPlayerSeen");
		if (fDistance < 40) then
			entity:InsertSubpipe(0, "just_shoot");
		else
			entity:SelectPipe(0, "SJDScoutJobApproachObjectiveStealth", name);
		end
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultScoutScouting::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultScoutScouting::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultScoutScouting::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultScoutScouting::OnEnemyMemory");
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultScoutScouting::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultScoutScouting::OnThreateningSoundHeard");
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultScoutScouting::OnReload");
	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultScoutScouting::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultScoutScouting::OnNoHidingPlace");
	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultScoutScouting::OnNoHidingPlace");
	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutScouting::OnReceivingDamage");
		if(sender.current_mounted_weapon) then
			-- Got a fix on the machine gunner so we can retreat
			System:Log("Got a fix on the machine gunner");
			-- Create a tagpoint at the enemy position
			Game:CreateTagPoint("ENEMY_HERE", sender:GetPos(), 
sender:GetDirectionVector());

			AI:Signal(0, 1, "SJD_SCOUT_ENEMY_CONTACT", entity.id);
		else
			entity:InsertSubpipe(0, "SJDScout_hide");
		end
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultScoutScouting::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultScoutScouting::OnBulletRain");
		if(sender.current_mounted_weapon) then
			-- Got a fix on the machine gunner so we can retreat
			System:Log("Got a fix on the machine gunner");

			-- Parse game tagpoints
			local radius = 30.0;
			tolerance = 15.0;
			local tagPointNames = Game:GetTagPointsInRadius(sender:GetPos(),radius, 
tolerance);
			for idx, name in tagPointNames do
				System:Log("Tag Point in radius: "..name);
			end

			local hTagPointName = Game:GetHighestTagPointInRadius(sender:GetPos(),radius, 
tolerance);
			System:Log("Highest Tag Point in radius: "..hTagPointName);

			-- Create a tagpoint at the enemy position
			--Game:CreateTagPoint("ENEMY_HERE", sender:GetPos(), 
sender:GetDirectionVector());
			BlackboardObject:addScoutTargetSighting(entity.id, sender:GetPos());

			AI:Signal(0, 1, "SJD_SCOUT_ENEMY_CONTACT", entity.id);
		else
			entity:InsertSubpipe(0, "SJDScout_hide");
		end
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultScoutScouting::OnDeath");

		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_SCOUT_DIED", entity.id);
	end,
	--------------------------------------------------
	-- GROUP SIGNALS
	---------------------------------------------
	KEEP_FORMATION = function (self, entity, sender)
		-- the team leader wants everyone to keep formation
	end,
	---------------------------------------------
	BREAK_FORMATION = function (self, entity, sender)
```

```
		-- the team can split
	end,
	---------------------------------------------
	SINGLE_GO = function (self, entity, sender)
		-- the team leader has instructed this group member to approach the enemy
	end,
	---------------------------------------------
	GROUP_COVER = function (self, entity, sender)
		-- the team leader has instructed this group member to cover his friends
	end,
	---------------------------------------------
	IN_POSITION = function (self, entity, sender)
		-- some member of the group is safely in position
	end,
	---------------------------------------------
	GROUP_SPLIT = function (self, entity, sender)
		-- team leader instructs group to split
	end,
	---------------------------------------------
	PHASE_RED_ATTACK = function (self, entity, sender)
		-- team leader instructs red team to attack
	end,
	---------------------------------------------
	PHASE_BLACK_ATTACK = function (self, entity, sender)
		-- team leader instructs black team to attack
	end,
	---------------------------------------------
	GROUP_MERGE = function (self, entity, sender)
		-- team leader instructs groups to merge into a team again
	end,
	---------------------------------------------
	CLOSE_IN_PHASE = function (self, entity, sender)
		-- team leader instructs groups to initiate part one of assault fire maneuver
	end,
	---------------------------------------------
	ASSAULT_PHASE = function (self, entity, sender)
		-- team leader instructs groups to initiate part one of assault fire maneuver
	end,
	---------------------------------------------
	GROUP_NEUTRALISED = function (self, entity, sender)
		-- team leader instructs groups to initiate part one of assault fire maneuver
	end,

	SJD_SCOUT_ENEMY_CONTACT = function (self, entity, sender)

		System:Log("AssaultScoutScouting::SJD_SCOUT_ENEMY_CONTACT");

		-- Get the tagpoint back again
		local etp = Game:GetTagPoint("ENEMY_HERE");
		if (etp) then
			local dist = entity:GetDistanceFromPoint(etp);
			System:Log("Distance to enemy tagpoint is: "..dist);
		end

		-- Begin Retreat
		AI:MakePuppetIgnorant(entity.id, 1); -- make ignorant to events
		local name = "rallypoint";
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:SelectPipe(0, "SJDScout_retreat", name);
			entity:InsertSubpipe(0, "SJDScout_hide");
		end
		AI:MakePuppetIgnorant(entity.id, 0); -- make non-ignorant to events
	end,

	SJD_SCOUT_GOT_TO_NEXT_POINT = function (self, entity, sender)
		System:Log("AssaultScoutScouting::SJD_SCOUT_GOT_TO_NEXT_POINT");
		entity:InsertSubpipe(0, "SJDScout_pause");
	end,
}

-----------------------------------------------------------------------
-- AssaultCoverAtObjective
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- entity has reached enemy objective.
-----------------------------------------------------------------------

AIBehaviour.AssaultCoverAtObjective = {
	Name = "AssaultCoverAtObjective",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultCoverAtObjective::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultCoverAtObjective::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultCoverAtObjective::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultCoverAtObjective::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultCoverAtObjective::");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultCoverAtObjective::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultCoverAtObjective::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverAtObjective::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultCoverAtObjective::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverAtObjective::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverAtObjective::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverAtObjective::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverAtObjective::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
```

```
                -- called when no hiding place can be found with the specified parameters
                Hud:AddMessage("AssaultCoverAtObjective::");

        end,
        --------------------------------------------------
        OnNoFormationPoint = function ( self, entity, sender)
                -- called when the enemy found no formation point
                Hud:AddMessage("AssaultCoverAtObjective::OnNoHidingPlace");

        end,
        ---------------------------------------------
        OnReceivingDamage = function ( self, entity, sender)
                Hud:AddMessage("AssaultCoverAtObjective::OnReceivingDamage");

        end,
        ---------------------------------------------
        OnCoverRequested = function ( self, entity, sender)
                -- called when the enemy is damaged
                Hud:AddMessage("AssaultCoverAtObjective::OnCoverRequested");

        end,
        --------------------------------------------------
        OnBulletRain = function ( self, entity, sender)
                -- called when the enemy detects bullet trails around him
                Hud:AddMessage("AssaultCoverAtObjective::OnBulletRain");
        end,
        --------------------------------------------------

}
-----------------------------------------------------------------------
-- AssaultCoverBreaching
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- entity has been instructed to move through enemy breach point to objective.
-----------------------------------------------------------------------

AIBehaviour.AssaultCoverBreaching = {
        Name = "AssaultCoverBreaching",

        -- SYSTEM EVENTS                    -----
        ---------------------------------------------
        OnSelected = function( self, entity )
                Hud:AddMessage("AssaultCoverBreaching::OnSelected");
        end,
        ---------------------------------------------
        OnSpawn = function( self, entity )
                -- called when enemy spawned or reset
                Hud:AddMessage("AssaultCoverBreaching::OnSpawn");
        end,
        ---------------------------------------------
        OnActivate = function( self, entity )
                -- called when enemy receives an activate event (from a trigger, for example)
                Hud:AddMessage("AssaultCoverBreaching::OnActivate");
        end,
        ---------------------------------------------
        OnNoTarget = function( self, entity )
                -- called when the enemy stops having an attention target
                Hud:AddMessage("AssaultCoverBreaching::OnNoTarget");
        end,
        ---------------------------------------------
        OnPlayerSeen = function( self, entity, fDistance )
                -- called when the enemy sees a living player
                Hud:AddMessage("AssaultCoverBreaching::");
        end,
        ---------------------------------------------
        OnGrenadeSeen = function( self, entity, fDistance )
                -- called when the enemy sees a grenade
                Hud:AddMessage("AssaultCoverBreaching::OnGrenadeSeen");
        end,
        ---------------------------------------------
        OnEnemySeen = function( self, entity )
                -- called when the enemy sees a foe which is not a living player
                Hud:AddMessage("AssaultCoverBreaching::");
        end,
        ---------------------------------------------
```

```
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverBreaching::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultCoverBreaching::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverBreaching::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverBreaching::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverBreaching::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverBreaching::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultCoverBreaching::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultCoverBreaching::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultCoverBreaching::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultCoverBreaching::OnCoverRequested");

	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultCoverBreaching::OnBulletRain");
	end,
	--------------------------------------------------

}
-----------------------------------------------------------------------
-- AssaultCoverEngaging
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- entity is engaging a target.
-----------------------------------------------------------------------

AIBehaviour.AssaultCoverEngaging = {
	Name = "AssaultCoverEngaging",

	-- SYSTEM EVENTS			-----
```

```
        --------------------------------------------
        OnSelected = function( self, entity )
                Hud:AddMessage("AssaultCoverEngaging::OnSelected");
        end,
        --------------------------------------------
        OnSpawn = function( self, entity )
                -- called when enemy spawned or reset
                Hud:AddMessage("AssaultCoverEngaging::OnSpawn");
        end,
        --------------------------------------------
        OnActivate = function( self, entity )
                -- called when enemy receives an activate event (from a trigger, for example)
                Hud:AddMessage("AssaultCoverEngaging::OnActivate");
        end,
        --------------------------------------------
        OnNoTarget = function( self, entity )
                -- called when the enemy stops having an attention target
                Hud:AddMessage("AssaultCoverEngaging::OnNoTarget");
        end,
        --------------------------------------------
        OnPlayerSeen = function( self, entity, fDistance )
        -- called when the enemy sees a living player

                Hud:AddMessage("AssaultCoverEngaging::OnPlayerSeen");
                if (fDistance < 15) then
                        entity:InsertSubpipe(0, "just_shoot");
                end

                -- Pass the ray-description-table (shooter, pos, angles, dir, distance) and
                -- receive the target-description-table (objtype (0=entity, 1=stat-obj, 2=terrain),
                -- pos, normal, dir, target (nil if objtype!=0))

                local trace={};
                local firePos={x=0,y=0,z=0};
                local fireAng={x=0,y=0,z=0};
                entity.cnt:GetFirePosAngles(firePos, fireAng);
                ClampAngle(fireAng);
                --System:Log("fireang ,"..fireAng.x..","..fireAng.y..","..fireAng.z.."## firePos
,"..firePos.x..","..firePos.y..","..firePos.z);
                trace.pos = firePos;
                trace.dir = entity:GetDirectionVector();
                trace.distance = 1000;
                trace.shooter = entity;
                local weapon = entity.cnt:GetCurrWeapon();
                local hits = weapon:GetInstantHit(trace);
                local iTargetAngle = 0;
                local lTargetSpot = {x=0,y=0,z=0};
                local TargetDescTable;
                local tPos = {x=0,y=0,z=0};
                if(hits)then
                        --get the first(unpierceble hit)
                        TargetDescTable = hits[0];
                        if (TargetDescTable.objtype == 0) then
                                ta = new(TargetDescTable.target);
                                tPos = TargetDescTable.pos;
                                System:Log("Target number is: "..ta);
                        end
                end

                -- Work out distances:
                local vectorTarget = DifferenceVectors(firePos, tPos);
                vectorDist = sqrt((vectorTarget.x*vectorTarget.x) + (vectorTarget.y*vectorTarget.y));
                System:Log("LUA distance: "..fDistance);
                System:Log("Vector distance: "..vectorDist);

        end,
        --------------------------------------------
        OnGrenadeSeen = function( self, entity, fDistance )
                -- called when the enemy sees a grenade
                Hud:AddMessage("AssaultCoverEngaging::OnGrenadeSeen");
        end,
        --------------------------------------------
        OnEnemySeen = function( self, entity )
                -- called when the enemy sees a foe which is not a living player
                Hud:AddMessage("AssaultCoverEngaging::OnEnemySeen");
        end,
        --------------------------------------------
```

```
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverEngaging::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it 
last
		System:Log("***AssaultCoverEngaging::OnEnemyMemory***");
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )

		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverEngaging::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverEngaging::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverEngaging::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverEngaging::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultCoverEngaging::OnNoHidingPlace");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultCoverEngaging::OnNoFormationPoint");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultCoverEngaging::OnReceivingDamage");
		entity:InsertSubpipe(0, "SJDget_down");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultCoverEngaging::OnCoverRequested");

	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultCoverEngaging::OnBulletRain");
		System:Log("Cover bullet Rain, shooter is: "..sender.id);
		entity:InsertSubpipe(0, "SJDget_down");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		-- we got killed, so issue a signal to release the others
		BlackboardObject:coverFinishedMovingUp(entity.id);

		-- ... and remove our combat state details
		BlackboardObject:removeCoverCombatState(entity.id);

		-- Tell team leader
		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_COVER_DIED", entity.id);
	end,
	--------------------------------------------------
	SJD_FINISHED_FIRING = function (self, entity, sender)

		-- called when entity has finished random firing period
		--System:Log("AssaultCoverEngaging::SJD_FINISHED_FIRING");

		-- Determine whether we can move up at this point in time
		local elaspedTime = _time-entity.lastTime;
		--System:Log("Time is: "..elaspedTime);

		if (BlackboardObject:canCoverMoveUp(entity.id, elaspedTime) == false) then
			-- We cannot move up at the moment
			AI:Signal(0, 1, "SJD_STAY_PUT", entity.id);
		else
			-- We can move up - run for it !
			AI:Signal(0, 1, "SJD_MOVE_UP", entity.id);
		end
	end,

	SJD_STAY_PUT = function (self, entity, sender)
		-- reacquire fire target and hence keep laying down cover fire)
		-- Get the tagpoint name
		local name = "SJDGruntPath_"..entity:GetName();
		--System:Log(name.." :Staying put and reacquiring target...");
		entity:InsertSubpipe(0, "SJDGruntStealthAcquireTarget", name);
	end,

	SJD_MOVE_UP = function (self, entity, sender)
		-- We can move up
		-- Get the tagpoint name
		local name = "SJDGruntPath_"..entity:GetName();
		local tagpoint = Game:GetTagPoint(name);
		System:Log("Tagpoint position is x: "..tagpoint.x..", y: "..tagpoint.y..", Z:
"..tagpoint.z);

		if (tagpoint) then
			-- Get our distance from the tagpoint
			local dist = entity:GetDistanceFromPoint(tagpoint);
			--System:Log("Distance from next point is: "..dist);

			-- If distance is > 20 metres then advance by 15%
			-- else go directly to the tagpoint
			if (dist > 20) then
				entity:InsertSubpipe(0, "SJDGruntJobPatrolPath", name);
				--System:Log(name.." :Running to next point");
			else
				entity:InsertSubpipe(0, "SJDGruntJobRunToTagPoint", name);
				--System:Log(name.." :Running to LAST point");
			end
		end
	end,


}
----------------------------------------------------------------------
-- AssaultCoverIdle
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- in idle state.
----------------------------------------------------------------------

AIBehaviour.AssaultCoverIdle = {
	Name = "AssaultCoverIdle",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultCoverIdle::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultCoverIdle::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
```

```
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultCoverIdle::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultCoverIdle::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultCoverIdle::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultCoverIdle::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultCoverIdle::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverIdle::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultCoverIdle::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverIdle::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverIdle::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverIdle::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverIdle::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultCoverIdle::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultCoverIdle::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultCoverIdle::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultCoverIdle::OnCoverRequested");

	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultCoverIdle::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		-- we got killed, so issue a signal to release the others
		BlackboardObject:coverFinishedMovingUp(entity.id);

		-- ... and remove our combat state details
		BlackboardObject:removeCoverCombatState(entity.id);
	end,
	--------------------------------------------------
}
-----------------------------------------------------------------------
-- AssaultCoverInPosn
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- entity has moved from rally point and is now ready at initial position.
-----------------------------------------------------------------------

AIBehaviour.AssaultCoverInPosn = {
	Name = "AssaultCoverInPosn",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultCoverInPosn::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultCoverInPosn::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultCoverInPosn::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultCoverInPosn::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultCoverInPosn::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultCoverInPosn::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultCoverInPosn::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverInPosn::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultCoverInPosn::OnEnemyMemory");

	end,
	--------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverInPosn::OnInterestingSoundHeard");
	end,
	--------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverInPosn::OnThreateningSoundHeard");

	end,
	--------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverInPosn::OnReload");

	end,
	--------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverInPosn::OnGroupMemberDied");
	end,
	--------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultCoverInPosn::OnNoHidingPlace");

	end,
	-------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultCoverInPosn::OnNoHidingPlace");

	end,
	--------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultCoverInPosn::OnReceivingDamage");
		entity:InsertSubpipe(0, "SJDget_down");
	end,
	--------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultCoverInPosn::OnCoverRequested");

	end,
	-------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultCoverInPosn::OnBulletRain");
		System:Log("Cover bullet Rain, shooter is: "..sender.id);
		entity:InsertSubpipe(0, "SJDget_down");
	end,
	-------------------------------------------------

	SJD_ALL_MOVE_TO_OBJECTIVE = function (self, entity, sender)
		local name = "objective";
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:SelectPipe(0, "SJDGruntStealthIdle", name);
			entity:InsertSubpipe(0, "SJDTeamLeadJobBreach", name);
		end
	end,
}
-----------------------------------------------------------------------
-- AssaultCoverMovingUp
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- entity is moving up towards a target.
-----------------------------------------------------------------------

AIBehaviour.AssaultCoverMovingUp = {
	Name = "AssaultCoverMovingUp",

	-- SYSTEM EVENTS			-----
	--------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultCoverMovingUp::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultCoverMovingUp::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultCoverMovingUp::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultCoverMovingUp::OnNoTarget");
		-- Reacquire fire target
		local name = "SJDGruntPath_"..entity:GetName();
		--Hud:AddMessage("Reacquiring target...");
		entity:InsertSubpipe(0, "SJDGruntStealthAcquireTarget", name);
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
	-- called when the enemy sees a living player
		Hud:AddMessage("AssaultCoverMovingUp::OnPlayerSeen");
		if (fDistance < 15) then
			entity:InsertSubpipe(0, "just_shoot");
		end
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultCoverMovingUp::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultCoverMovingUp::");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverMovingUp::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultCoverMovingUp::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverMovingUp::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverMovingUp::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverMovingUp::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverMovingUp::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultCoverMovingUp::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultCoverMovingUp::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultCoverMovingUp::OnReceivingDamage");
		local shooterPosn = sender:GetPos();
		System:Log("Cover bullet Rain, shooter is: "..sender.id..", posn. is x:
"..shooterPosn.x..", y: "..shooterPosn.y.." , z: "..shooterPosn.z);
		entity:InsertSubpipe(0, "SJDget_down");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultCoverMovingUp::OnCoverRequested");

	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultCoverMovingUp::OnBulletRain");
		local shooterPosn = sender:GetPos();
		System:Log("Cover bullet Rain, shooter is: "..sender.id..", posn. is x:
"..shooterPosn.x..", y: "..shooterPosn.y.." , z: "..shooterPosn.z);
		entity:InsertSubpipe(0, "SJDget_down");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		-- we got killed, so issue a signal to release the others
		BlackboardObject:coverFinishedMovingUp(entity.id);

		-- ... and remove our combat state details
		BlackboardObject:removeCoverCombatState(entity.id);

		-- Tell team leader
		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_COVER_DIED", entity.id);
	end,
	--------------------------------------------------

	SJD_GOT_TO_NEXT_POINT = function (self, entity, sender)
		--Hud:AddMessage(entity:GetName()..", AssaultCoverMovingUp::SJD_GOT_TO_NEXT_POINT");
		entity.lastTime = _time;

		-- Announce our arrival
		BlackboardObject:coverFinishedMovingUp(entity.id);

		-- Reacquire fire target
		local name = "SJDGruntPath_"..entity:GetName();
		--Hud:AddMessage("Reacquiring target...");
		entity:InsertSubpipe(0, "SJDGruntStealthAcquireTarget", name);
	end,

	SJD_LEAVING_FOR_NEXT_POINT = function (self, entity, sender)
		Hud:AddMessage(entity:GetName()..",
AssaultCoverMovingUp::SJD_LEAVING_FOR_NEXT_POINT");
	end,

	SJD_GOT_TO_END_POINT = function (self, entity, sender)

		-- Announce our arrival
		BlackboardObject:coverFinishedMovingUp(entity.id);

		-- Get the tagpoint name
		local name = "SJDGruntPath_"..entity:GetName();
		local tagpoint = Game:GetTagPoint(name);

		-- Inform the team leader of our status
		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_COVER_IN_POSITION", entity.id);

		--Hud:AddMessage(entity:GetName()..", AssaultCoverMovingUp::SJD_GOT_TO_END_POINT");
		entity:SelectPipe(0, "SJDGruntStealthStandAndShoot", name);
```

```
        end,

}
-----------------------------------------------------------------------
-- AssaultCoverRallied
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of cover team in a squad conducting an assault,
-- at rally point, ready to start mission.
-----------------------------------------------------------------------

AIBehaviour.AssaultCoverRallied = {
        Name = "AssaultCoverRallied",

        -- SYSTEM EVENTS                    -----
        ---------------------------------------------
        OnSelected = function( self, entity )
                Hud:AddMessage("AssaultCoverRallied::OnSelected");
        end,
        ---------------------------------------------
        OnSpawn = function( self, entity )
                -- called when enemy spawned or reset
                Hud:AddMessage("AssaultCoverRallied::OnSpawn");

                -- Register combat state
                BlackboardObject:addCoverCombatState(entity.id);

                -- Set elapsed time in position timer
                entity.lastTime = _time;

                -- Select default behaviour
                name = "SJDGruntPath_"..entity:GetName();
                Hud:AddMessage("Name is: "..name);
                entity:SelectPipe(0, "SJDGruntStealthIdle", name);
        end,
        ---------------------------------------------
        OnActivate = function( self, entity )
                -- called when enemy receives an activate event (from a trigger, for example)
                Hud:AddMessage("AssaultCoverRallied::OnActivate");
        end,
        ---------------------------------------------
        OnNoTarget = function( self, entity )
                -- called when the enemy stops having an attention target
                Hud:AddMessage("AssaultCoverRallied::OnNoTarget");
        end,
        ---------------------------------------------
        OnPlayerSeen = function( self, entity, fDistance )
                -- called when the enemy sees a living player
                Hud:AddMessage("AssaultCoverRallied::OnPlayerSeen");

                --entity:SelectPipe(0, "minimize_exposure");
        end,
        ---------------------------------------------
        OnGrenadeSeen = function( self, entity, fDistance )
                -- called when the enemy sees a grenade
                Hud:AddMessage("AssaultCoverRallied::OnGrenadeSeen");
        end,
        ---------------------------------------------
        OnEnemySeen = function( self, entity )
                -- called when the enemy sees a foe which is not a living player
                Hud:AddMessage("AssaultCoverRallied::");
        end,
        ---------------------------------------------
        OnSomethingSeen = function( self, entity )
                -- called when the enemy sees something that it cant identify
                Hud:AddMessage("AssaultCoverRallied::OnSomethingSeen");
        end,
        ---------------------------------------------
        OnEnemyMemory = function( self, entity )
                -- called when the enemy can no longer see its foe, but remembers where it saw it
last
                Hud:AddMessage("AssaultCoverRallied::OnEnemyMemory");

        end,
        ---------------------------------------------
        OnInterestingSoundHeard = function( self, entity )
```

```
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultCoverRallied::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultCoverRallied::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultCoverRallied::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultCoverRallied::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultCoverRallied::OnNoHidingPlace");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultCoverRallied::OnNoFormationPoint");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultCoverRallied::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultCoverRallied::OnCoverRequested");

	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultCoverRallied::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		-- we got killed, so issue a signal to release the others
		BlackboardObject:coverFinishedMovingUp(entity.id);

		-- ... and remove our combat state details
		BlackboardObject:removeCoverCombatState(entity.id);

		-- Tell team leader
		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_COVER_DIED", entity.id);
	end,

	SJD_COVER_GO = function (self, entity, sender)
		local name = "SJDGruntPath_"..entity:GetName();
		entity:SelectPipe(0, "SJDGruntStealthEngaging", name);
		entity:InsertSubpipe(0, "SJDGruntStealthAcquireTarget", name);
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:InsertSubpipe(0, "SJDGruntJobPatrolPath", name);
		end
	end,
}

----------------------------------------------------------------------
-- AssaultBreachAtObjective
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in 'at objective' state.
-----------------------------------------------------------------------

AIBehaviour.AssaultBreachAtObjective = {
	Name = "AssaultBreachAtObjective",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultBreachAtObjective::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultBreachAtObjective::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultBreachAtObjective::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultBreachAtObjective::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultBreachAtObjective::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultBreachAtObjective::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultBreachAtObjective::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultBreachAtObjective::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultBreachAtObjective::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultBreachAtObjective::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultBreachAtObjective::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultBreachAtObjective::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultBreachAtObjective::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
```

```
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultBreachAtObjective::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultBreachAtObjective::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachAtObjective::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultBreachAtObjective::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultBreachAtObjective::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachAtObjective::OnDeath");
	end,
}
-----------------------------------------------------------------------
-- AssaultBreachBreaching
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in breaching state.
-----------------------------------------------------------------------

AIBehaviour.AssaultBreachBreaching = {
	Name = "AssaultBreachBreaching",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultBreachBreaching::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultBreachBreaching::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultBreachBreaching::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultBreachBreaching::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultBreachBreaching::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultBreachBreaching::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultBreachBreaching::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultBreachBreaching::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultBreachBreaching::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultBreachBreaching::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultBreachBreaching::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultBreachBreaching::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultBreachBreaching::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultBreachBreaching::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultBreachBreaching::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachBreaching::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultBreachBreaching::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultBreachBreaching::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachBreaching::OnDeath");

		-- Remove our combat state details
		BlackboardObject:removeBreachCombatState(entity.id);
	end,
}
-----------------------------------------------------------------------
-- AssaultBreachEngaging
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in engaging state.
-----------------------------------------------------------------------
```

```
AIBehaviour.AssaultBreachEngaging = {
	Name = "AssaultBreachEngaging",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultBreachEngaging::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultBreachEngaging::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultBreachEngaging::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultBreachEngaging::OnNoTarget");

		-- Assume enemy was killed
		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_NO_TARGET", entity.id);
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultBreachEngaging::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultBreachEngaging::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultBreachEngaging::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultBreachEngaging::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultBreachEngaging::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultBreachEngaging::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultBreachEngaging::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultBreachEngaging::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultBreachEngaging::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultBreachEngaging::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultBreachEngaging::OnNoHidingPlace");

	end,
	--------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachEngaging::OnReceivingDamage");

	end,
	--------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultBreachEngaging::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultBreachEngaging::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachEngaging::OnDeath");

		-- Remove our combat state details
		BlackboardObject:removeBreachCombatState(entity.id);

		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_DIED", entity.id);
	end,
	GROUP_NEUTRALISED = function (self, entity, sender)
		-- team leader instructs groups to initiate part one of assault fire maneuver
	end,

	SJD_ALL_MOVE_TO_OBJECTIVE = function (self, entity, sender)
		local name = "objective";
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:SelectPipe(0, "SJDGruntStealthIdle", name);
			entity:InsertSubpipe(0, "SJDTeamLeadJobBreach", name);
		end
	end,
}
-----------------------------------------------------------------------
-- AssaultBreachIdle
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in idle state.
-----------------------------------------------------------------------

AIBehaviour.AssaultBreachIdle = {
	Name = "AssaultBreachIdle",

	-- SYSTEM EVENTS			-----
	--------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultBreachIdle::OnSelected");
	end,
	--------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultBreachIdle::OnSpawn");
	end,
	--------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultBreachIdle::OnActivate");
	end,
	--------------------------------------------
	OnNoTarget = function( self, entity )
```

```
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultBreachIdle::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultBreachIdle::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultBreachIdle::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultBreachIdle::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultBreachIdle::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultBreachIdle::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultBreachIdle::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultBreachIdle::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultBreachIdle::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultBreachIdle::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultBreachIdle::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultBreachIdle::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachIdle::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultBreachIdle::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
```

```
                Hud:AddMessage("AssaultBreachIdle::OnBulletRain");
        end,
        -------------------------------------------------
        OnDeath = function ( self, entity, sender)
                Hud:AddMessage("AssaultBreachIdle::OnDeath");

                -- Remove our combat state details
                BlackboardObject:removeBreachCombatState(entity.id);
        end,
}
----------------------------------------------------------------------
-- AssaultBreachInPosn
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in 'in position' state.
----------------------------------------------------------------------

AIBehaviour.AssaultBreachInPosn = {
        Name = "AssaultBreachInPosn",

        -- SYSTEM EVENTS                    -----
        ---------------------------------------------
        OnSelected = function( self, entity )
                Hud:AddMessage("AssaultBreachInPosn::OnSelected");
        end,
        ---------------------------------------------
        OnSpawn = function( self, entity )
                -- called when enemy spawned or reset
                Hud:AddMessage("AssaultBreachInPosn::OnSpawn");
        end,
        ---------------------------------------------
        OnActivate = function( self, entity )
                -- called when enemy receives an activate event (from a trigger, for example)
                Hud:AddMessage("AssaultBreachInPosn::OnActivate");
        end,
        ---------------------------------------------
        OnNoTarget = function( self, entity )
                -- called when the enemy stops having an attention target
                Hud:AddMessage("AssaultBreachInPosn::OnNoTarget");
        end,
        ---------------------------------------------
        OnPlayerSeen = function( self, entity, fDistance )
                -- called when the enemy sees a living player
                Hud:AddMessage("AssaultBreachInPosn::OnPlayerSeen");
        end,
        ---------------------------------------------
        OnGrenadeSeen = function( self, entity, fDistance )
                -- called when the enemy sees a grenade
                Hud:AddMessage("AssaultBreachInPosn::OnGrenadeSeen");
        end,
        ---------------------------------------------
        OnEnemySeen = function( self, entity )
                -- called when the enemy sees a foe which is not a living player
                Hud:AddMessage("AssaultBreachInPosn::OnEnemySeen");
        end,
        ---------------------------------------------
        OnSomethingSeen = function( self, entity )
                -- called when the enemy sees something that it cant identify
                Hud:AddMessage("AssaultBreachInPosn::OnSomethingSeen");
        end,
        ---------------------------------------------
        OnEnemyMemory = function( self, entity )
                -- called when the enemy can no longer see its foe, but remembers where it saw it
last
                Hud:AddMessage("AssaultBreachInPosn::OnEnemyMemory");

        end,
        ---------------------------------------------
        OnInterestingSoundHeard = function( self, entity )
                -- called when the enemy hears an interesting sound
                Hud:AddMessage("AssaultBreachInPosn::OnInterestingSoundHeard");
        end,
        ---------------------------------------------
        OnThreateningSoundHeard = function( self, entity )
                -- called when the enemy hears a scary sound
                Hud:AddMessage("AssaultBreachInPosn::OnThreateningSoundHeard");

        end,
        ---------------------------------------------
        OnReload = function( self, entity )
                -- called when the enemy goes into automatic reload after its clip is empty
                Hud:AddMessage("AssaultBreachInPosn::OnReload");

        end,
        ---------------------------------------------
        OnGroupMemberDied = function( self, entity )
                -- called when a member of the group dies
                Hud:AddMessage("AssaultBreachInPosn::OnGroupMemberDied");
        end,
        ---------------------------------------------
        OnNoHidingPlace = function( self, entity, sender )
                -- called when no hiding place can be found with the specified parameters
                Hud:AddMessage("AssaultBreachInPosn::");

        end,
        --------------------------------------------------
        OnNoFormationPoint = function ( self, entity, sender)
                -- called when the enemy found no formation point
                Hud:AddMessage("AssaultBreachInPosn::OnNoHidingPlace");

        end,
        ---------------------------------------------
        OnReceivingDamage = function ( self, entity, sender)
                Hud:AddMessage("AssaultBreachInPosn::OnReceivingDamage");

        end,
        ---------------------------------------------
        OnCoverRequested = function ( self, entity, sender)
                -- called when the enemy is damaged
                Hud:AddMessage("AssaultBreachInPosn::OnCoverRequested");
        end,
        --------------------------------------------------
        OnBulletRain = function ( self, entity, sender)
                -- called when the enemy detects bullet trails around him
                Hud:AddMessage("AssaultBreachInPosn::OnBulletRain");
        end,
        --------------------------------------------------
        OnDeath = function ( self, entity, sender)
                Hud:AddMessage("AssaultBreachInPosn::OnDeath");

                -- Remove our combat state details
                BlackboardObject:removeBreachCombatState(entity.id);

                AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_DIED", entity.id);
        end,

        SJD_BREACH_ATTACK = function (self, entity, sender)
                -- Team Leader has instructed us to attack
                -- Move to a spot with line-of-fire and fire at will
                System:Log("Breach copy: attacking...")
                entity:SelectPipe(0, "SJDBreachJobAttack", name);
        end,
}
----------------------------------------------------------------------
-- AssaultBreachMovingUp
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in 'moving up' state.
----------------------------------------------------------------------

AIBehaviour.AssaultBreachMovingUp = {
        Name = "AssaultBreachMovingUp",

        -- SYSTEM EVENTS                         -----
        ---------------------------------------------
        OnSelected = function( self, entity )
                Hud:AddMessage("AssaultBreachMovingUp::OnSelected");
        end,
        ---------------------------------------------
        OnSpawn = function( self, entity )
```

```
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultBreachMovingUp::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultBreachMovingUp::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultBreachMovingUp::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultBreachMovingUp::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultBreachMovingUp::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultBreachMovingUp::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultBreachMovingUp::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultBreachMovingUp::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultBreachMovingUp::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultBreachMovingUp::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultBreachMovingUp::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultBreachMovingUp::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultBreachMovingUp::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultBreachMovingUp::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachMovingUp::OnReceivingDamage");
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultBreachMovingUp::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultBreachMovingUp::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachMovingUp::OnDeath");

		-- Remove our combat state details
		BlackboardObject:removeBreachCombatState(entity.id);

		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_DIED", entity.id);
	end,

	SJD_BREACH_GOT_TO_NEXT_POINT = function (self, entity, sender)

		local name = "breachpoint0"..entity.breachPoint;
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			local dist = entity:GetDistanceFromPoint(tagpoint);
			if (dist < 0.5) then
				entity.breachPoint = entity.breachPoint+1;
			end
			System:Log("Heading for: "..name);
			entity:SelectPipe(0, "SJDBreachJobApproachBreachPoint", name);
		else
			AI:Signal(0, 1, "SJD_BREACH_DIG_IN", entity.id);
		end
	end,

	SJD_BREACH_DIG_IN = function (self, entity, sender)
			System:Log("Digging in...");
			AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_IN_POSITION", entity.id);
			entity:SelectPipe(0, "SJDBreach_Wait_For_Orders", name);
			entity:InsertSubpipe(0, "SJDBreach_Dig_In");
	end,
}
----------------------------------------------------------------------
-- AssaultBreachRallied
--
-- S J Drayton 14/08/2007
--
-- Behaviour for member of breach team in a squad conducting an assault,
-- in rallied state.
----------------------------------------------------------------------

AIBehaviour.AssaultBreachRallied = {
	Name = "AssaultBreachRallied",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultBreachRallied::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultBreachRallied::OnSpawn");

		entity.breachPoint = 1;

		-- Register combat state
		BlackboardObject:addBreachCombatState(entity.id);

		-- Select default behaviour
		entity:SelectPipe(0, "SJDGruntStealthIdle");

	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultBreachRallied::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultBreachRallied::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultBreachRallied::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultBreachRallied::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultBreachRallied::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultBreachRallied::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultBreachRallied::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultBreachRallied::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultBreachRallied::OnThreateningSoundHeard");
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultBreachRallied::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultBreachRallied::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultBreachRallied::");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultBreachRallied::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachRallied::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultBreachRallied::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultBreachRallied::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultBreachRallied::OnDeath");
	end,

	SJD_BREACH_GO = function (self, entity, sender)
		local name = "breachpoint0"..entity.breachPoint;
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:SelectPipe(0, "SJDBreachJobApproachBreachPoint", name);
		end
	end,
}

----------------------------------------------------------------------
-- AssaultLeadAtObjective
--
-- S J Drayton 14/08/2007
--
-- Behaviour for squad leader conducting an assault,
-- in 'at objective' state.
----------------------------------------------------------------------

AIBehaviour.AssaultLeadAtObjective = {
	Name = "AssaultLeadAtObjective",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultLeadAtObjective::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultLeadAtObjective::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultLeadAtObjective::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultLeadAtObjective::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultLeadAtObjective::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultLeadAtObjective::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultLeadAtObjective::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultLeadAtObjective::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
```

```
		Hud:AddMessage("AssaultLeadAtObjective::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultLeadAtObjective::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultLeadAtObjective::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultLeadAtObjective::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultLeadAtObjective::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultLeadAtObjective::OnNoHidingPlace");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultLeadAtObjective::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadAtObjective::OnReceivingDamage");

	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultLeadAtObjective::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultLeadAtObjective::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadAtObjective::OnDeath");
	end,
}
-----------------------------------------------------------------------
-- AssaultLeadBreaching
--
-- S J Drayton 14/08/2007
--
-- Behaviour for squad leader conducting an assault,
-- in breaching state.
-----------------------------------------------------------------------

AIBehaviour.AssaultLeadBreaching = {
	Name = "AssaultLeadBreaching",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultLeadBreaching::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultLeadBreaching::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultLeadBreaching::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultLeadBreaching::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultLeadBreaching::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultLeadBreaching::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultLeadBreaching::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultLeadBreaching::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it 
last
		Hud:AddMessage("AssaultLeadBreaching::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultLeadBreaching::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultLeadBreaching::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultLeadBreaching::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultLeadBreaching::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultLeadBreaching::OnNoHidingPlace");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultLeadBreaching::OnNoHidingPlace");

	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadBreaching::OnReceivingDamage");
```

```
        end,
        --------------------------------------------
        OnCoverRequested = function ( self, entity, sender)
                -- called when the enemy is damaged
                Hud:AddMessage("AssaultLeadBreaching::OnCoverRequested");
        end,
        -------------------------------------------------
        OnBulletRain = function ( self, entity, sender)
                -- called when the enemy detects bullet trails around him
                Hud:AddMessage("AssaultLeadBreaching::OnBulletRain");
        end,
        -------------------------------------------------
        OnDeath = function ( self, entity, sender)
                Hud:AddMessage("AssaultLeadBreaching::OnDeath");
        end,

}
----------------------------------------------------------------------
-- AssaultLeadExecuting
--
-- S J Drayton 14/08/2007
--
-- Behaviour for squad leader conducting an assault,
-- in executing state.
----------------------------------------------------------------------

AIBehaviour.AssaultLeadExecuting = {
        Name = "AssaultLeadExecuting",

        -- SYSTEM EVENTS                    -----
        --------------------------------------------
        OnSelected = function( self, entity )
                Hud:AddMessage("AssaultLeadExecuting::OnSelected");
        end,
        --------------------------------------------
        OnSpawn = function( self, entity )
                -- called when enemy spawned or reset
                Hud:AddMessage("AssaultLeadExecuting::OnSpawn");
        end,
        --------------------------------------------
        OnActivate = function( self, entity )
                -- called when enemy receives an activate event (from a trigger, for example)
                Hud:AddMessage("AssaultLeadExecuting::OnActivate");
        end,
        --------------------------------------------
        OnNoTarget = function( self, entity )
                -- called when the enemy stops having an attention target
                Hud:AddMessage("AssaultLeadExecuting::OnNoTarget");
        end,
        --------------------------------------------
        OnPlayerSeen = function( self, entity, fDistance )
                -- called when the enemy sees a living player
                Hud:AddMessage("AssaultLeadExecuting::OnPlayerSeen");
        end,
        --------------------------------------------
        OnGrenadeSeen = function( self, entity, fDistance )
                -- called when the enemy sees a grenade
                Hud:AddMessage("AssaultLeadExecuting::OnGrenadeSeen");
        end,
        --------------------------------------------
        OnEnemySeen = function( self, entity )
                -- called when the enemy sees a foe which is not a living player
                Hud:AddMessage("AssaultLeadExecuting::OnEnemySeen");
        end,
        --------------------------------------------
        OnSomethingSeen = function( self, entity )
                -- called when the enemy sees something that it cant identify
                Hud:AddMessage("AssaultLeadExecuting::OnSomethingSeen");
        end,
        --------------------------------------------
        OnEnemyMemory = function( self, entity )
                -- called when the enemy can no longer see its foe, but remembers where it saw it
last
                Hud:AddMessage("AssaultLeadExecuting::OnEnemyMemory");

        end,
        --------------------------------------------
        OnInterestingSoundHeard = function( self, entity )
                -- called when the enemy hears an interesting sound
                Hud:AddMessage("AssaultLeadExecuting::OnInterestingSoundHeard");
        end,
        ---------------------------------------------
        OnThreateningSoundHeard = function( self, entity )
                -- called when the enemy hears a scary sound
                Hud:AddMessage("AssaultLeadExecuting::OnThreateningSoundHeard");

        end,
        ---------------------------------------------
        OnReload = function( self, entity )
                -- called when the enemy goes into automatic reload after its clip is empty
                Hud:AddMessage("AssaultLeadExecuting::OnReload");

        end,
        ---------------------------------------------
        OnGroupMemberDied = function( self, entity )
                -- called when a member of the group dies
                Hud:AddMessage("AssaultLeadExecuting::OnGroupMemberDied");
        end,
        ---------------------------------------------
        OnNoHidingPlace = function( self, entity, sender )
                -- called when no hiding place can be found with the specified parameters
                Hud:AddMessage("AssaultLeadExecuting::OnNoHidingPlace");

        end,
        --------------------------------------------------
        OnNoFormationPoint = function ( self, entity, sender)
                -- called when the enemy found no formation point
                Hud:AddMessage("AssaultLeadExecuting::OnNoHidingPlace");

        end,
        ---------------------------------------------
        OnReceivingDamage = function ( self, entity, sender)
                Hud:AddMessage("AssaultLeadExecuting::OnReceivingDamage");

        end,
        ---------------------------------------------
        OnCoverRequested = function ( self, entity, sender)
                -- called when the enemy is damaged
                Hud:AddMessage("AssaultLeadExecuting::OnCoverRequested");
        end,
        --------------------------------------------------
        OnBulletRain = function ( self, entity, sender)
                -- called when the enemy detects bullet trails around him
                Hud:AddMessage("AssaultLeadExecuting::OnBulletRain");
        end,
        --------------------------------------------------
        OnDeath = function ( self, entity, sender)
                Hud:AddMessage("AssaultLeadExecuting::OnDeath");
        end,

        SJD_SCOUT_RETURNED = function (self, entity, sender) -- The scout got back OK
                System:Log("Team Lead copy: Scout returned");

                local targetPos = BlackboardObject:getScoutTargetSighting(sender.id);
                if (targetPos) then
                        System:Log("Scout reportd target, x: "..targetPos.x..", y: "..targetPos.y.." ,
z: "..targetPos.z);
                end

                -- Send out the cover
                AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_COVER_GO", entity.id);

                -- Send out the breach
                AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_GO", entity.id);
        end,

        SJD_COVER_IN_POSITION = function (self, entity, sender) -- a cover is in position
                System:Log("Team Lead copy: Cover in position");

                BlackboardObject:setCoverInPosn(sender.id, 1);

                -- Check if breach attack can be launched
                AI:Signal(0, 1, "SJD_BREACH_ATTACK_GONOGO", entity.id);
        end,
	SJD_BREACH_IN_POSITION = function (self, entity, sender)
		System:Log("Team Lead copy: Breach in position");

		BlackboardObject:setBreachInPosn(sender.id, 1);

		-- Check if breach attack can be launched
		AI:Signal(0, 1, "SJD_BREACH_ATTACK_GONOGO", entity.id);
	end,

	SJD_BREACH_ATTACK_GONOGO = function (self, entity, sender)

		local covers = BlackboardObject:getAllCoversInPosn();
		local breaches = BlackboardObject:getAllBreachesInPosn();

		if (covers ~= nil) then
			if (breaches ~= nil) then
				System:Log("Team Lead: Telling breach to attack")
				AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_BREACH_ATTACK", entity.id);
			else
				System:Log("Breaches not in position yet...")
			end
		else
			System:Log("Covers not in position yet...")
		end
	end,

	SJD_BREACH_NO_TARGET = function (self, entity, sender)

		-- Breach no longer has a target so assume enemy was destroyed; move
		-- everyone up to objective point
		System:Log("Team Lead copy: Target destroyed");

		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_ALL_MOVE_TO_OBJECTIVE", entity.id);
	end,

	SJD_ALL_MOVE_TO_OBJECTIVE = function (self, entity, sender)
		local name = "objective";
		local tagpoint = Game:GetTagPoint(name);
		if (tagpoint) then
			entity:SelectPipe(0, "SJDGruntStealthIdle", name);
			entity:InsertSubpipe(0, "SJDTeamLeadJobBreach", name);
		end
	end,

	-----------------------------------------------------------------------
	-- Use death events to determine whether we should abort the mission --
	-----------------------------------------------------------------------

	SJD_BREACH_DIED = function (self, entity, sender)
		System:Log("Team Lead copy: Breach died");

		-- AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_ABORT_MISSION", entity.id);
	end,

	SJD_COVER_DIED = function (self, entity, sender)
		System:Log("Team Lead copy: Cover died");

		-- AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_ABORT_MISSION", entity.id);
	end,

	SJD_SCOUT_DIED = function (self, entity, sender)
		System:Log("Team Lead copy: Scout died");

		-- AI:Signal(SIGNALFILTER_GROUPONLY, 1, "SJD_ABORT_MISSION", entity.id);
	end,

}
-----------------------------------------------------------------------
-- AssaultLeadIdle
--
-- S J Drayton 14/08/2007
--
-- Behaviour for squad leader conducting an assault,
-- in idle state.
-----------------------------------------------------------------------
```

```
AIBehaviour.AssaultLeadIdle = {
	Name = "AssaultLeadIdle",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultLeadIdle::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultLeadIdle::OnSpawn");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultLeadIdle::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultLeadIdle::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultLeadIdle::OnPlayerSeen");
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultLeadIdle::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultLeadIdle::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultLeadIdle::OnSomethingSeen");
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultLeadIdle::OnEnemyMemory");

	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultLeadIdle::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultLeadIdle::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultLeadIdle::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultLeadIdle::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultLeadIdle::OnNoHidingPlace");
	end,
	------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultLeadIdle::OnNoHidingPlace");

	end,
	--------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadIdle::OnReceivingDamage");

	end,
	--------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultLeadIdle::OnCoverRequested");
	end,
	------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultLeadIdle::OnBulletRain");
	end,
	------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadIdle::OnDeath");
	end,

}
----------------------------------------------------------------------
-- AssaultLeadRallied
--
-- S J Drayton 14/08/2007
--
-- Behaviour for squad leader conducting an assault,
-- in rallied state.
----------------------------------------------------------------------

AIBehaviour.AssaultLeadRallied = {
	Name = "AssaultLeadRallied",

	-- SYSTEM EVENTS                    -----
	--------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("AssaultLeadRallied::OnSelected");
	end,
	--------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("AssaultLeadRallied::OnSpawn");
		entity:SelectPipe(0, "SJDGruntStealthIdle");
	end,
	--------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("AssaultLeadRallied::OnActivate");
		entity.coversInPosn = 0;
		AI:Signal(0, 1, "SJD_LEAD_EXEC_MISSION", entity.id);
	end,
	--------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
		Hud:AddMessage("AssaultLeadRallied::OnNoTarget");
	end,
	--------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		Hud:AddMessage("AssaultLeadRallied::OnPlayerSeen");
	end,
	--------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("AssaultLeadRallied::OnGrenadeSeen");
	end,
	--------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("AssaultLeadRallied::OnEnemySeen");
	end,
	--------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultLeadRallied::OnSomethingSeen");
	end,
	--------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
		Hud:AddMessage("AssaultLeadRallied::OnEnemyMemory");

	end,
	--------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("AssaultLeadRallied::OnInterestingSoundHeard");
	end,
	--------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("AssaultLeadRallied::OnThreateningSoundHeard");

	end,
	--------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("AssaultLeadRallied::OnReload");

	end,
	--------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("AssaultLeadRallied::OnGroupMemberDied");
	end,
	--------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
		Hud:AddMessage("AssaultLeadRallied::OnNoHidingPlace");

	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
		Hud:AddMessage("AssaultLeadRallied::OnNoHidingPlace");

	end,
	--------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadRallied::OnReceivingDamage");

	end,
	--------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
		Hud:AddMessage("AssaultLeadRallied::OnCoverRequested");
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		Hud:AddMessage("AssaultLeadRallied::OnBulletRain");
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		Hud:AddMessage("AssaultLeadRallied::OnDeath");
	end,
}

-----------------------------------------------------------------------
-- EnemyMGEngaging
--
-- S J Drayton 14/08/2007
--
-- Behaviour for an enemy machine gunner that has an enagaged a target.
-----------------------------------------------------------------------

AIBehaviour.EnemyMGEngaging = {
	Name = "EnemyMGEngaging",
	NOPREVIOUS = 1,

	-- SYSTEM EVENTS			-----
	OnSpawn = function( self, entity )

	end,
	---------------------------------------------
	OnSelected = function( self, entity )
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
	end,
	--------------------------------------------
	OnNoTarget = function( self, entity )
		-- called when the enemy stops having an attention target
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		--Hud:AddMessage("EnemyMGEngaging::OnPlayerSeen");
		System:Log("Player seen at distance: "..fDistance);
		if (fDistance < entity.engageDistMin) then
			entity.engageDistMin = fDistance;
		end
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("EnemyMGEngaging::OnEnemySeen");
	end,
	---------------------------------------------
	OnFriendSeen = function( self, entity )
		-- called when the enemy sees a friendly target
	end,
	---------------------------------------------
	OnDeadBodySeen = function( self, entity )
		-- called when the enemy a dead body
	end,
	---------------------------------------------
	OnEnemyMemory = function( self, entity )
		-- called when the enemy can no longer see its foe, but remembers where it saw it
last
	end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		--if (entity.engageDistMin < 30) then
		--	AI:Signal(0,1,"SJD_DUCK",entity.id);
		--end
	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
	end,
	---------------------------------------------
	OnGroupMemberDiedNearest = function( self, entity )
		-- called when a member of the group dies
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
	end,
	--------------------------------------------------
	OnNoFormationPoint = function ( self, entity, sender)
		-- called when the enemy found no formation point
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("AssaultCoverEngaging::OnSomethingSeen");
	end,
	---------------------------------------------
	OnReceivingDamage = function ( self, entity, sender)
		-- called when the enemy is damaged
		System:Log("Enemy MG Receiving damage, shooter is: "..sender.id);
		AI:Signal(0,1,"SJD_DUCK",entity.id);
	end,
	---------------------------------------------
	OnCoverRequested = function ( self, entity, sender)
		-- called when the enemy is damaged
	end,
	--------------------------------------------------
	OnBulletRain = function ( self, entity, sender)
		-- called when the enemy detects bullet trails around him
		System:Log("Enemy MG bullet rain, shooter is: "..sender.id);
		AI:Signal(0,1,"SJD_DUCK",entity.id);
	end,
	--------------------------------------------------
	OnDeath = function ( self, entity, sender)
		-- we got killed, so issue a signal to release the others
		if (BlackboardObject:getGunnerID() == entity.id) then
			BlackboardObject:gunnerDied();
		end
		if (entity.current_mounted_weapon) then
			entity.current_mounted_weapon:OnReset(); -- release the gun
		end
	end,
	--------------------------------------------------
	HEADS_UP_GUYS = function ( self, entity, sender)
	end,
	--------------------------------------------------
	KEEP_FORMATION = function ( self, entity, sender)
	end,

	SJD_DUCK = function (self, entity, sender)
		System:Log("Ducking!");
		AI:MakePuppetIgnorant(entity.id, 1); -- make ignorant to events
		if (entity.current_mounted_weapon) then
			entity.current_mounted_weapon:OnReset(); -- release the gun
		end
		if (BlackboardObject:getGunnerID() == entity.id) then
			if (entity.engageDistMin < 35) then
				entity:InsertSubpipe(0,"SJDduck_stand_long"); -- duck for a longer
random time
			else
				entity:InsertSubpipe(0,"SJDduck_stand"); -- duck for a random time
			end
		else
			if (entity.engageDistMin < 30) then
				entity:InsertSubpipe(0,"SJDduck_crouch_long"); -- duck for a longer
random time
			else
				entity:InsertSubpipe(0,"SJDduck_crouch"); -- duck for a random time
			end
		end
		AI:MakePuppetIgnorant(entity.id, 0); -- make non-ignorant to events
	end,

	SJD_USE_MOUNTED_WEAPON_ENGAGING = function (self, entity, sender)
		System:Log("I'm at the Mounted Weapon tagpoint");
		local mounted = AI:FindObjectOfType(entity.id, 10, AIOBJECT_MOUNTEDWEAPON);
		if (mounted) then
			local gun = System:GetEntityByName(mounted);
			if (gun) then
				System:Log("Setting me as the gunner");
				gun:SetGunner( entity ); -- set this entity as the gunner
				entity.AI_AtWeapon = 1;
				entity:SelectPipe(0,"SJDuse_weapon");
			end
		else
			System:Log("NO MOUNTED WEAPON FOUND");
		end

	end,
	SJD_USE_NORMAL_WEAPON_ENGAGING = function (self, entity, sender)
			entity.AI_AtNormalWeapon = 1;
			System:Log("I'm at the normal weapon tagpoint");
			entity:SelectPipe(0,"SJDuse_weapon");
	end,

	SJD_USE_WEAPON_ENGAGING =  function (self, entity, sender)
		AI:MakePuppetIgnorant(entity.id, 0); -- make non-ignorant to events
		if (BlackboardObject:gunnerAssigned() == false) then
			BlackboardObject:setGunnerID(entity.id);
			-- designating ourself as the machine gunner
			System:Log("Setting myself as the designated MG");
			if (entity.AI_AtWeapon == nil) then
				entity:InsertSubpipe(0,"SJDgoto_mounted_weapon_engaging");
			end
			AI:Signal(0,1,"SJD_USE_MOUNTED_WEAPON_ENGAGING",entity.id);
		else
			if (BlackboardObject:getGunnerID() == entity.id) then
				-- we are already the designated machine gunner
				System:Log("I'm the designated MG");
				if (entity.AI_AtWeapon == nil) then
					entity:InsertSubpipe(0,"SJDgoto_mounted_weapon_engaging");
				end
				AI:Signal(0,1,"SJD_USE_MOUNTED_WEAPON_ENGAGING",entity.id);
			else
				-- we are just a normal gunner
				System:Log("I'm a normal gunner");
				AI:Signal(0,1,"SJD_USE_NORMAL_WEAPON_ENGAGING",entity.id);
			end
		end
		AI:MakePuppetIgnorant(entity.id, 0); -- make non-ignorant to events
	end,

	SJD_RETURN_TO_NORMAL_ENGAGING = function ( self, entity, sender)
		System:Log("EnemyMGEngaging::SJD_RETURN_TO_NORMAL");
		entity:TriggerEvent(AIEVENT_CLEAR);
		entity:SelectPipe(0,"just_shoot");
		if(entity.current_mounted_weapon) then
			entity.current_mounted_weapon:AbortUse();
		end
	end,

}
----------------------------------------------------------------------
-- EnemyMGIdle
--
-- S J Drayton 14/08/2007
--
-- Behaviour for enemy machine gunner, in idle state.
----------------------------------------------------------------------

AIBehaviour.EnemyMGIdle = {
	Name = "EnemyMGIdle",

	-- SYSTEM EVENTS			-----
	---------------------------------------------
	OnSelected = function( self, entity )
		Hud:AddMessage("EnemyMGIdle::OnSelected");
	end,
	---------------------------------------------
	OnSpawn = function( self, entity )
		-- called when enemy spawned or reset
		Hud:AddMessage("EnemyMGIdle::OnSpawn");
		entity.AI_AtWeapon = nil;
		entity.AI_AtNormalWeapon = nil;
		entity.engageDistMin = 999.0;
		-- Select default behaviour
		entity:SelectPipe(0, "SJDGruntStealthIdle");
	end,
	---------------------------------------------
	OnActivate = function( self, entity )
		-- called when enemy receives an activate event (from a trigger, for example)
		Hud:AddMessage("EnemyMGIdle::OnActivate");
	end,
	---------------------------------------------
	OnNoTarget = function( self, entity )
```

```
		-- called when the enemy stops having an attention target
		Hud:AddMessage("EnemyMGIdle::OnNoTarget");
	end,
	---------------------------------------------
	OnPlayerSeen = function( self, entity, fDistance )
		-- called when the enemy sees a living player
		System:Log("EnemyMGIdle::OnPlayerSeen");

		-- Tell the group
		AI:Signal(SIGNALFILTER_GROUPONLY, 1, "INCOMING_FIRE",entity.id);

		if ((entity.AI_AtWeapon) or (entity.AI_AtNormalWeapon)) then
			AI:Signal(0,1,"SJD_USE_WEAPON_IDLE",entity.id);
			do return end
		end

		if (BlackboardObject:gunnerAssigned() == false) then
			AI:MakePuppetIgnorant(entity.id, 1); -- make ignorant to events
			System:Log("I'm NOT at the Mounted Weapon tagpoint - going there now");
			--entity:TriggerEvent(AIEVENT_CLEAR); -- needed !
			entity:InsertSubpipe(0,"SJDgoto_mounted_weapon_idle");
			AI:MakePuppetIgnorant(entity.id, 0); -- make non-ignorant to events
		else
			AI:MakePuppetIgnorant(entity.id, 1); -- make ignorant to events
			System:Log("I'm NOT at the normal weapon tagpoint - going there now");
			--entity:TriggerEvent(AIEVENT_CLEAR); -- needed !
			entity:InsertSubpipe(0,"SJDgoto_normal_weapon");
			AI:MakePuppetIgnorant(entity.id, 0); -- make non-ignorant to events
		end
	end,
	---------------------------------------------
	OnGrenadeSeen = function( self, entity, fDistance )
		-- called when the enemy sees a grenade
		Hud:AddMessage("EnemyMGIdle::OnGrenadeSeen");
	end,
	---------------------------------------------
	OnEnemySeen = function( self, entity )
		-- called when the enemy sees a foe which is not a living player
		Hud:AddMessage("EnemyMGIdle::OnEnemySeen");
	end,
	---------------------------------------------
	OnSomethingSeen = function( self, entity )
		-- called when the enemy sees something that it cant identify
		Hud:AddMessage("EnemyMGIdle::OnSomethingSeen");
	end,
	---------------------------------------------
	--OnEnemyMemory = function( self, entity )
	--	-- called when the enemy can no longer see its foe, but remembers where it saw it
last
	--	Hud:AddMessage("EnemyMGIdle::OnEnemyMemory");

	--end,
	---------------------------------------------
	OnInterestingSoundHeard = function( self, entity )
		-- called when the enemy hears an interesting sound
		Hud:AddMessage("EnemyMGIdle::OnInterestingSoundHeard");
	end,
	---------------------------------------------
	OnThreateningSoundHeard = function( self, entity )
		-- called when the enemy hears a scary sound
		Hud:AddMessage("EnemyMGIdle::OnThreateningSoundHeard");

	end,
	---------------------------------------------
	OnReload = function( self, entity )
		-- called when the enemy goes into automatic reload after its clip is empty
		Hud:AddMessage("EnemyMGIdle::OnReload");

	end,
	---------------------------------------------
	OnGroupMemberDied = function( self, entity )
		-- called when a member of the group dies
		Hud:AddMessage("EnemyMGIdle::OnGroupMemberDied");
	end,
	---------------------------------------------
	OnNoHidingPlace = function( self, entity, sender )
		-- called when no hiding place can be found with the specified parameters
```

```
        Hud:AddMessage("EnemyMGIdle::");

end,
--------------------------------------------------
OnNoFormationPoint = function ( self, entity, sender)
        -- called when the enemy found no formation point
        Hud:AddMessage("EnemyMGIdle::OnNoHidingPlace");

end,
---------------------------------------------
OnReceivingDamage = function ( self, entity, sender)
        Hud:AddMessage("EnemyMGIdle::OnReceivingDamage");
        AI:Signal(0,1,"SJD_DUCK",entity.id);
end,
---------------------------------------------
OnCoverRequested = function ( self, entity, sender)
        -- called when the enemy is damaged
        Hud:AddMessage("EnemyMGIdle::OnCoverRequested");

end,
--------------------------------------------------
OnBulletRain = function ( self, entity, sender)
        -- called when the enemy detects bullet trails around him
        Hud:AddMessage("EnemyMGIdle::OnBulletRain");
        AI:Signal(0,1,"SJD_DUCK",entity.id);
end,
--------------------------------------------------
OnDeath = function ( self, entity, sender)
        -- we got killed, so issue a signal to release the others
        if (BlackboardObject:getGunnerID() == entity.id) then
                BlackboardObject:gunnerDied();
        end
end,
--------------------------------------------------

SJD_RETURN_TO_NORMAL_IDLE = function ( self, entity, sender)
        System:Log("EnemyMGIdle::SJD_RETURN_TO_NORMAL");
        entity:TriggerEvent(AIEVENT_CLEAR);
        entity:SelectPipe(0,"just_shoot");
        if(entity.current_mounted_weapon) then
                entity.current_mounted_weapon:AbortUse();
        end
end,

SJD_USE_MOUNTED_WEAPON_IDLE = function (self, entity, sender)
        System:Log("I'm at the Mounted Weapon tagpoint");
        local mounted = AI:FindObjectOfType(entity.id, 10, AIOBJECT_MOUNTEDWEAPON);
        if (mounted) then
                local gun = System:GetEntityByName(mounted);
                if (gun) then
                        System:Log("Setting me as the gunner");
                        gun:SetGunner( entity ); -- set this entity as the gunner
                        entity.AI_AtWeapon = 1;
                        entity:SelectPipe(0,"SJDuse_weapon");
                end
        end

end,

SJD_USE_NORMAL_WEAPON_IDLE = function (self, entity, sender)
                entity.AI_AtNormalWeapon = 1;
                System:Log("I'm at the normal weapon tagpoint");
                entity:SelectPipe(0,"SJDuse_weapon");
end,

SJD_USE_WEAPON_IDLE =  function (self, entity, sender)
        if (BlackboardObject:gunnerAssigned() == false) then
                BlackboardObject:setGunnerID(entity.id);
                -- we are the designated machine gunner
                AI:Signal(0,1,"SJD_USE_MOUNTED_WEAPON_IDLE",entity.id);
        else
                if (BlackboardObject:getGunnerID() == entity.id) then
                        -- we are the designated machine gunner
                        AI:Signal(0,1,"SJD_USE_MOUNTED_WEAPON_IDLE",entity.id);
                else
                        -- we are just a normal gunner
                        AI:Signal(0,1,"SJD_USE_NORMAL_WEAPON_IDLE",entity.id);
```

```
                              end
                         end
                    end,
               }
```

Appendix G: Lua Character Scripts

```
-----------------------------------------------------------------------
-- AssaultBreach
--
-- S J Drayton 14/08/2007
--
-- Character script for member of scout team in a squad conducting an
-- assault mission.
-----------------------------------------------------------------------

AICharacter.AssaultBreach =
{

	AssaultBreachRallied =
	{
		SJD_BREACH_GO				= "AssaultBreachMovingUp",
	},

	AssaultBreachMovingUp =
	{
		SJD_BREACH_DIG_IN		= "AssaultBreachInPosn",
	},

	AssaultBreachInPosn =
	{
		SJD_BREACH_ATTACK		= "AssaultBreachEngaging",
	},

	AssaultBreachEngaging =
	{
		SJD_ALL_MOVE_TO_OBJECTIVE	= "AssaultBreachBreaching",
	},

	AssaultBreachRetreating =
	{
	},


	AssaultBreachBreaching =
	{

	},

	AssaultBreachAtObjective =
	{

	},

	AssaultBreachIdle =
	{
	},
}

-----------------------------------------------------------------------
-- AssaultCover
--
-- S J Drayton 14/08/2007
--
-- Character script for member of cover team in a squad conducting an
-- assault mission.
-----------------------------------------------------------------------

AICharacter.AssaultCover =
{
	AssaultCoverRallied =
	{
		SJD_COVER_GO			= "AssaultCoverMovingUp",
	},

	AssaultCoverMovingIntoPosn =
	{
	},
        AssaultCoverInPosn =
        {
                SJD_ALL_MOVE_TO_OBJECTIVE     = "AssaultCoverBreaching",
        },


        AssaultCoverEngaging =
        {
                SJD_MOVE_UP                   = "AssaultCoverMovingUp",
        },

        AssaultCoverMovingUp =
        {
                SJD_GOT_TO_NEXT_POINT         = "AssaultCoverEngaging",
                SJD_GOT_TO_END_POINT          = "AssaultCoverInPosn",
        },

        AssaultCoverBreaching =
        {
        },

        AssaultCoverAtObjective =
        {
        },

        AssaultCoverRetreating =
        {
        },

        AssaultCoverIdle =
        {
        },
}

-----------------------------------------------------------------------
-- EnemyMG
--
-- S J Drayton 14/08/2007
--
-- Character script for an enemy machine gunner.
-----------------------------------------------------------------------

AICharacter.EnemyMG =
{
        EnemyMGIdle =
        {
                SJD_USE_WEAPON_IDLE    = "EnemyMGEngaging",
        },

        EnemyMGEngaging =
        {

        },
}

-----------------------------------------------------------------------
-- AssaultScout
--
-- S J Drayton 14/08/2007
--
-- Character script for member of scout team in a squad conducting an
-- assault mission.
-----------------------------------------------------------------------

AICharacter.AssaultScout =
{

        AssaultScoutRallied =
        {
                SJD_SCOUT_GO                  = "AssaultScoutScouting",
                SJD_ALL_MOVE_TO_OBJECTIVE     = "AssaultScoutBreaching",
        },

        AssaultScoutScouting =
        {
                SJD_SCOUT_ENEMY_CONTACT       = "AssaultScoutRetreating",
	},


	AssaultScoutRetreating =
	{
		SJD_SCOUT_GOT_TO_RALLY_POINT  = "AssaultScoutRallied",
	},


	AssaultScoutBreaching =
	{

	},

	AssaultScoutAtObjective =
	{

	},

	AssaultScoutIdle =
	{

	},
}

------------------------------------------------------------------------
-- AssaultLead
--
-- S J Drayton 14/08/2007
--
-- Character script for leader of a squad conducting an
-- assault mission.
------------------------------------------------------------------------

AICharacter.AssaultLead =
{

	AssaultLeadRallied =
	{
		SJD_LEAD_EXEC_MISSION			= "AssaultLeadExecuting",
	},

	AssaultLeadIdle =
	{

	},

	AssaultLeadExecuting =
	{
		SJD_ALL_MOVE_TO_OBJECTIVE		= "AssaultLeadBreaching",
	},

	AssaultLeadBreaching =
	{

	},

	AssaultLeadAtObjective =
	{

	},
}
```

Appendix H: Lua Goal Pipe Scripts

```
function PipeManager:OnInitSJD()

        System:Log("INIT SJD CALLED");

        -- Cover

        AI:CreateGoalPipe("SJDGruntJobPatrolPath");
        AI:PushGoal("SJDGruntJobPatrolPath", "signal", 0, 1, "SJD_LEAVING_FOR_NEXT_POINT", 0);
        AI:PushGoal("SJDGruntJobPatrolPath", "bodypos", 1, 0);
        AI:PushGoal("SJDGruntJobPatrolPath", "run", 1, 1);
        AI:PushGoal("SJDGruntJobPatrolPath","devalue",1,0);
        AI:PushGoal("SJDGruntJobPatrolPath","firecmd", 1, 0); -- stop firing
        AI:PushGoal("SJDGruntJobPatrolPath", "acqtarget", 1, "");
        AI:PushGoal("SJDGruntJobPatrolPath","lookat", 1, 0, 0);
        AI:PushGoal("SJDGruntJobPatrolPath", "approach", 1, 0.85, 1);
        AI:PushGoal("SJDGruntJobPatrolPath", "locate", 1, AIAnchor.AIANCHOR_SHOOTSPOTCROUCH);
        AI:PushGoal("SJDGruntJobPatrolPath", "pathfind",1,"");
        AI:PushGoal("SJDGruntJobPatrolPath", "trace",1,1);
        AI:PushGoal("SJDGruntJobPatrolPath", "run", 1, 0);
        AI:PushGoal("SJDGruntJobPatrolPath","firecmd", 1, 1); -- resume firing at will
        AI:PushGoal("SJDGruntJobPatrolPath", "signal", 1, 1, "SJD_GOT_TO_NEXT_POINT", 0);

        AI:CreateGoalPipe("SJDGruntJobRunToTagPoint");
        AI:PushGoal("SJDGruntJobRunToTagPoint", "signal", 0, 1, "SJD_LEAVING_FOR_NEXT_POINT", 0);
        AI:PushGoal("SJDGruntJobRunToTagPoint", "bodypos", 1, 0);
        AI:PushGoal("SJDGruntJobRunToTagPoint", "run", 1, 1);
        AI:PushGoal("SJDGruntJobRunToTagPoint","devalue",1,0);
        AI:PushGoal("SJDGruntJobRunToTagPoint","firecmd", 1, 0); -- stop firing
        AI:PushGoal("SJDGruntJobRunToTagPoint", "acqtarget", 1, "");
        AI:PushGoal("SJDGruntJobRunToTagPoint","lookat", 1, 0, 0);
        AI:PushGoal("SJDGruntJobRunToTagPoint", "approach", 1, 0.01, 1);
        AI:PushGoal("SJDGruntJobRunToTagPoint", "run", 1, 0);
        AI:PushGoal("SJDGruntJobRunToTagPoint","firecmd", 1, 1); -- resume firing at will
        AI:PushGoal("SJDGruntJobRunToTagPoint", "signal", 1, 1, "SJD_GOT_TO_END_POINT", 0);

        AI:CreateGoalPipe("SJDGruntJobHide");
        AI:PushGoal("SJDGruntJobHide", "hide", 1, 20, HM_NEAREST);

        AI:CreateGoalPipe("SJDGruntStealthIdle");
        AI:PushGoal("SJDGruntStealthIdle","firecmd", 1, 0); -- stop firing
        AI:PushGoal("SJDGruntStealthIdle","bodypos", 0, BODYPOS_STEALTH);

        AI:CreateGoalPipe("SJDGruntStealthStandAndShoot");
        AI:PushGoal("SJDGruntStealthStandAndShoot", "acqtarget", 1, "");
        AI:PushGoal("SJDGruntStealthStandAndShoot", "approach", 1, 0.01, 1);
        AI:PushGoal("SJDGruntStealthStandAndShoot","firecmd", 1, 1); -- keep firing
        AI:PushGoal("SJDGruntStealthStandAndShoot","bodypos", 0, BODYPOS_STAND);

        AI:CreateGoalPipe("SJDGruntStealthEngaging");
        AI:PushGoal("SJDGruntStealthEngaging","bodypos", 1, BODYPOS_CROUCH);
        AI:PushGoal("SJDGruntStealthEngaging","firecmd", 1, 1); -- fire at target
        AI:PushGoal("SJDGruntStealthEngaging","timeout", 1, 1, 2); -- for between 1 and 2 secs
        AI:PushGoal("SJDGruntStealthEngaging","firecmd", 0, 0); -- stop firing
        AI:PushGoal("SJDGruntStealthEngaging", "signal", 1, 1, "SJD_FINISHED_FIRING", 0);

        AI:CreateGoalPipe("SJDGruntStealthAcquireTarget");
        AI:PushGoal("SJDGruntStealthAcquireTarget","devalue",1,0);
        AI:PushGoal("SJDGruntStealthAcquireTarget", "acqtarget", 1, "");

        AI:CreateGoalPipe("SJDget_down");
        AI:PushGoal("SJDget_down", "bodypos", 1, BODYPOS_PRONE);
        AI:PushGoal("SJDget_down", "timeout", 1, 2, 3);
        AI:PushGoal("SJDget_down", "bodypos", 1, BODYPOS_STAND);


        -- Enemy MG
        AI:CreateGoalPipe("SJDgoto_mounted_weapon_idle");
        AI:PushGoal("SJDgoto_mounted_weapon_idle", "SJDgoto_mounted_weapon");
        AI:PushGoal("SJDgoto_mounted_weapon_idle","signal", 1, 1, "SJD_USE_WEAPON_IDLE", 0);

        AI:CreateGoalPipe("SJDgoto_mounted_weapon_engaging");
```

```
AI:PushGoal("SJDgoto_mounted_weapon_engaging", "SJDgoto_mounted_weapon");
AI:PushGoal("SJDgoto_mounted_weapon_engaging","signal", 1, 1, "SJD_USE_WEAPON_ENGAGING", 0);

AI:CreateGoalPipe("SJDgoto_mounted_weapon");
AI:PushGoal("SJDgoto_mounted_weapon","run", 1, 1);
AI:PushGoal("SJDgoto_mounted_weapon", "locate", 1, AIAnchor.USE_THIS_MOUNTED_WEAPON);
AI:PushGoal("SJDgoto_mounted_weapon", "pathfind", 1, "");
AI:PushGoal("SJDgoto_mounted_weapon", "trace", 1, 1);
AI:PushGoal("SJDgoto_mounted_weapon", "bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDgoto_mounted_weapon","run", 1, 0);
AI:PushGoal("SJDgoto_mounted_weapon","clear",1,1);

AI:CreateGoalPipe("SJDgoto_normal_weapon");
AI:PushGoal("SJDgoto_normal_weapon","run", 1, 1)
AI:PushGoal("SJDgoto_normal_weapon", "locate", 1, AIAnchor.AIANCHOR_SHOOTSPOTCROUCH);
AI:PushGoal("SJDgoto_normal_weapon", "pathfind", 1, "");
AI:PushGoal("SJDgoto_normal_weapon", "trace", 1, 1);
AI:PushGoal("SJDgoto_normal_weapon", "bodypos", 0, BODYPOS_CROUCH);
AI:PushGoal("SJDgoto_normal_weapon","run", 1, 0);
AI:PushGoal("SJDgoto_normal_weapon","clear",1,1);
AI:PushGoal("SJDgoto_normal_weapon","signal", 1, 1, "SJD_USE_WEAPON_IDLE", 0);

AI:CreateGoalPipe("SJDuse_weapon");
AI:PushGoal("SJDuse_weapon","firecmd",1,1);
AI:PushGoal("SJDuse_weapon","timeout",1,1,2);

AI:CreateGoalPipe("SJDduck_stand");
AI:PushGoal("SJDduck_stand","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDduck_stand","bodypos", 0, BODYPOS_PRONE);
AI:PushGoal("SJDduck_stand","timeout", 1, 2, 4);
AI:PushGoal("SJDduck_stand","bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDduck_stand","firecmd", 1, 2); -- resume firing
AI:PushGoal("SJDduck_stand","signal", 1, 1, "SJD_USE_WEAPON_ENGAGING", 0);

AI:CreateGoalPipe("SJDduck_stand_long");
AI:PushGoal("SJDduck_stand_long","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDduck_stand_long","bodypos", 0, BODYPOS_PRONE);
AI:PushGoal("SJDduck_stand_long","timeout", 1, 5, 7);
AI:PushGoal("SJDduck_stand_long","bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDduck_stand_long","firecmd", 1, 2); -- resume firing
AI:PushGoal("SJDduck_stand_long","signal", 1, 1, "SJD_USE_WEAPON_ENGAGING", 0);

AI:CreateGoalPipe("SJDduck_crouch");
AI:PushGoal("SJDduck_crouch","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDduck_crouch","bodypos", 0, BODYPOS_PRONE);
AI:PushGoal("SJDduck_crouch","timeout", 1, 2, 4);
AI:PushGoal("SJDduck_crouch","bodypos", 0, BODYPOS_CROUCH);
AI:PushGoal("SJDduck_crouch","firecmd", 1, 2); -- resume firing
AI:PushGoal("SJDduck_crouch","signal", 1, 1, "SJD_USE_WEAPON_ENGAGING", 0);

AI:CreateGoalPipe("SJDduck_crouch_long");
AI:PushGoal("SJDduck_crouch_long","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDduck_crouch_long","bodypos", 0, BODYPOS_PRONE);
AI:PushGoal("SJDduck_crouch_long","timeout", 1, 5, 7);
AI:PushGoal("SJDduck_crouch_long","bodypos", 0, BODYPOS_CROUCH);
AI:PushGoal("SJDduck_crouch_long","firecmd", 1, 2); -- resume firing
AI:PushGoal("SJDduck_crouch_long","signal", 1, 1, "SJD_USE_WEAPON_ENGAGING", 0);

-- Scout

AI:CreateGoalPipe("SJDScoutJobApproachObjectiveStealth");
AI:PushGoal("SJDScoutJobApproachObjectiveStealth", "bodypos", 0, BODYPOS_STEALTH);
AI:PushGoal("SJDScoutJobApproachObjectiveStealth", "run", 1, 0);
AI:PushGoal("SJDScoutJobApproachObjectiveStealth", "SJDScoutJobApproachObjective");

AI:CreateGoalPipe("SJDScoutJobApproachObjectiveRapid");
AI:PushGoal("SJDScoutJobApproachObjectiveRapid", "bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDScoutJobApproachObjectiveRapid", "run", 1, 1);
AI:PushGoal("SJDScoutJobApproachObjectiveRapid", "SJDScoutJobApproachObjective");

AI:CreateGoalPipe("SJDScoutJobApproachObjective");
AI:PushGoal("SJDScoutJobApproachObjective", "signal", 0, 1,
        "SJD_SCOUT_LEAVING_FOR_NEXT_POINT", 0);
AI:PushGoal("SJDScoutJobApproachObjective","devalue",1,0);
AI:PushGoal("SJDScoutJobApproachObjective","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDScoutJobApproachObjective", "acqtarget", 1, "");
AI:PushGoal("SJDScoutJobApproachObjective","lookat", 1, 0, 0);
```

```
AI:PushGoal("SJDScoutJobApproachObjective", "approach", 1, 0.85, 1);
AI:PushGoal("SJDScoutJobApproachObjective", "locate", 1, AIAnchor.AIANCHOR_SHOOTSPOTCROUCH);
AI:PushGoal("SJDScoutJobApproachObjective", "pathfind",1,"");
AI:PushGoal("SJDScoutJobApproachObjective", "trace",1,1);
AI:PushGoal("SJDScoutJobApproachObjective", "run", 1, 0);
AI:PushGoal("SJDScoutJobApproachObjective", "signal", 1, 1, "SJD_SCOUT_GOT_TO_NEXT_POINT",0);

AI:CreateGoalPipe("SJDScout_get_down");
AI:PushGoal("SJDScout_get_down", "bodypos", 1, BODYPOS_PRONE);
AI:PushGoal("SJDScout_get_down", "timeout", 1, 2, 3);
AI:PushGoal("SJDScout_get_down", "bodypos", 1, BODYPOS_STAND);
AI:PushGoal("SJDScout_get_down", "signal", 1, 1, "REACQUIRE_RALLY_POINT", 0);

AI:CreateGoalPipe("SJDScout_hide");
AI:PushGoal("SJDScout_hide", "bodypos", 1, BODYPOS_PRONE);
AI:PushGoal("SJDScout_hide", "timeout", 1, 2, 3);
AI:PushGoal("SJDScout_hide", "bodypos", 1, BODYPOS_STAND);
AI:PushGoal("SJDScout_hide", "run", 1, 1);
AI:PushGoal("SJDScout_hide", "locate", 1, AIAnchor.PLACEHOLDER);
AI:PushGoal("SJDScout_hide", "pathfind",1,"");
AI:PushGoal("SJDScout_hide", "trace",1,1);
AI:PushGoal("SJDScout_hide", "bodypos", 0, BODYPOS_CROUCH);
AI:PushGoal("SJDScout_hide","timeout",1, 2, 6);
AI:PushGoal("SJDScout_hide", "bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDScout_hide", "signal", 1, 1, "REACQUIRE_RALLY_POINT", 0);

AI:CreateGoalPipe("SJDScout_retreat");
AI:PushGoal("SJDScout_retreat", "signal", 0, 1, "SJD_SCOUT_LEAVING_FOR_RALLY_POINT", 0);
AI:PushGoal("SJDScout_retreat", "bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDScout_retreat", "run", 1, 1);
AI:PushGoal("SJDScout_retreat","devalue",1,0);
AI:PushGoal("SJDScout_retreat","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDScout_retreat", "acqtarget", 1, "");
AI:PushGoal("SJDScout_retreat", "pathfind",1,"");
AI:PushGoal("SJDScout_retreat", "trace",1,1);
AI:PushGoal("SJDScout_retreat", "signal", 1, 1, "SJD_SCOUT_GOT_TO_RALLY_POINT", 0);

AI:CreateGoalPipe("SJDScout_pause");
AI:PushGoal("SJDScout_pause", "bodypos", 0, BODYPOS_CROUCH);
AI:PushGoal("SJDScout_pause","lookat",1, -45, 45);
AI:PushGoal("SJDScout_pause","timeout",1, 2, 3);
AI:PushGoal("SJDScout_pause", "bodypos", 0, BODYPOS_STAND);

-- Breach

AI:CreateGoalPipe("SJDBreachJobApproachBreachPoint");
AI:PushGoal("SJDBreachJobApproachBreachPoint", "signal", 0, 1,
        "SJD_BREACH_LEAVING_FOR_NEXT_POINT", 0);
AI:PushGoal("SJDBreachJobApproachBreachPoint", "bodypos", 0, BODYPOS_STAND);
AI:PushGoal("SJDBreachJobApproachBreachPoint", "run", 1, 1);
--AI:PushGoal("SJDBreachJobApproachBreachPoint","devalue",1,0);
AI:PushGoal("SJDBreachJobApproachBreachPoint","firecmd", 1, 0); -- stop firing
AI:PushGoal("SJDBreachJobApproachBreachPoint", "acqtarget", 1, "");
AI:PushGoal("SJDBreachJobApproachBreachPoint","lookat", 1, 0, 0);
AI:PushGoal("SJDBreachJobApproachBreachPoint", "approach", 1, 0.01, 1);
AI:PushGoal("SJDBreachJobApproachBreachPoint", "run", 1, 0);
AI:PushGoal("SJDBreachJobApproachBreachPoint", "signal", 1, 1,
        "SJD_BREACH_GOT_TO_NEXT_POINT", 0);

AI:CreateGoalPipe("SJDBreach_Wait_For_Orders");
AI:PushGoal("SJDBreach_Wait_For_Orders", "firecmd", 1, 0);

AI:CreateGoalPipe("SJDBreach_Dig_In");
AI:PushGoal("SJDBreach_Dig_In", "locate", 1, AIAnchor.PLACEHOLDER);
AI:PushGoal("SJDBreach_Dig_In", "pathfind",1,"");
AI:PushGoal("SJDBreach_Dig_In", "trace",1,1);
AI:PushGoal("SJDBreach_Dig_In", "bodypos", 0, BODYPOS_CROUCH);

AI:CreateGoalPipe("SJDBreachJobAttack");
AI:PushGoal("SJDBreachJobAttack", "locate", 1, AIAnchor.AIANCHOR_SHOOTSPOTCROUCH);
AI:PushGoal("SJDBreachJobAttack", "pathfind",1,"");
AI:PushGoal("SJDBreachJobAttack", "trace",1,1);
AI:PushGoal("SJDBreachJobAttack","firecmd", 1, 1); -- fire at will

-- Shared
AI:CreateGoalPipe("SJDScoutJobBreach");
AI:PushGoal("SJDScoutJobBreach", "signal", 1, 1, "SJD_SCOUT_LEAVING_FOR_OBJECTIVE", 0);
```

```
        AI:PushGoal("SJDScoutJobBreach", "SJDTeamJobBreach");
        AI:PushGoal("SJDScoutJobBreach", "signal", 1, 1, "SJD_SCOUT_GOT_TO_OBJECTIVE", 0);

        AI:CreateGoalPipe("SJDCoverJobBreach");
        AI:PushGoal("SJDCoverJobBreach", "signal", 1, 1, "SJD_COVER_LEAVING_FOR_OBJECTIVE", 0);
        AI:PushGoal("SJDCoverJobBreach", "SJDTeamJobBreach");
        AI:PushGoal("SJDCoverJobBreach", "signal", 1, 1, "SJD_COVER_GOT_TO_OBJECTIVE", 0);

        AI:CreateGoalPipe("SJDBreachJobBreach");
        AI:PushGoal("SJDBreachJobBreach", "signal", 1, 1, "SJD_BREACH_LEAVING_FOR_OBJECTIVE", 0);
        AI:PushGoal("SJDBreachJobBreach", "SJDTeamJobBreach");
        AI:PushGoal("SJDBreachJobBreach", "signal", 1, 1, "SJD_BREACH_GOT_TO_OBJECTIVE", 0);

        AI:CreateGoalPipe("SJDTeamLeadJobBreach");
        AI:PushGoal("SJDTeamLeadJobBreach", "signal", 1, 1, "SJD_TEAMLEAD_LEAVING_FOR_OBJECTIVE", 0);
        AI:PushGoal("SJDTeamLeadJobBreach", "SJDTeamJobBreach");
        AI:PushGoal("SJDTeamLeadJobBreach", "signal", 1, 1, "SJD_TEAMLEAD_GOT_TO_OBJECTIVE", 0);

        AI:CreateGoalPipe("SJDTeamJobBreach");
        AI:PushGoal("SJDTeamJobBreach", "bodypos", 0, BODYPOS_STAND);
        AI:PushGoal("SJDTeamJobBreach", "run", 1, 1);
        AI:PushGoal("SJDTeamJobBreach","devalue",1,0);
        AI:PushGoal("SJDTeamJobBreach","firecmd", 1, 0); -- stop firing
        AI:PushGoal("SJDTeamJobBreach", "acqtarget", 1, "");
        AI:PushGoal("SJDTeamJobBreach","lookat", 1, 0, 0);
        AI:PushGoal("SJDTeamJobBreach", "approach", 1, 2, 1);
        AI:PushGoal("SJDTeamJobBreach", "run", 1, 0);
        AI:PushGoal("SJDTeamJobBreach", "bodypos", 0, BODYPOS_CROUCH);
end
```

Bibliography

[Fu99] John David Funge, 1999. *AI for games and animation: a cognitive modeling approach.* Natick, MA, USA: A. K. Peters, Ltd.

[St02] William van der Sterren, 2002. Squad Tactics: Team AI and Emergent Maneuvres. *In: Steve Rabin, ed. AI Game Programming Wisdom.* Rockland, MA, USA: Charles River Media, Inc., 233-246.

[SV04] Deniz T Sodiri, Venkat V S S Sastry, 2004. On the Interpretation of Gestures arising in Flight Deck Officers Training. *Simulation Interoperability Standards Organization*, *Proc. Behavior Representation in Modeling and Simulation Conference 2004,* http://www.sisostds.org/

[Li02] Lars Liden, 2002. Strategic and Tactical Reasoning with Waypoints. *In: Steve Rabin, ed. AI Game Programming Wisdom.* Rockland, MA, USA: Charles River Media, Inc., 211-220.

[KRO05] Arno Kamphuis, Michiel Rook, Mark H. Overmars, 2005. Tactical Path Finding in Urban Environments, *Proc. First International Workshop on Crowd Simulation 2005, http://vrlab.epfl.ch/v_crowds_05/*

[Ou99] Mark Goodwin, ed., et al, 1995. Object Oriented Analysis and Design, Book II. Milton Keynes, United Kingdom : The Open University.

[An72] P.W. Anderson, 1972. More is Different: Broken Symmetry and the Nature of the Hierarchical Structure of Science". *Science 177,* 393-396

[Le91] Robert Leonhard, 1991. *The Art of Maneuver.* Toronto, Canada: Presidio Press.

[Br03] Jason Brownlee, 2003. Finite State Machines (FSM), http://ai-depot.com/FiniteStateMachines/

[Bu05] Matt Buckland, 2005. *Programming Game AI by Example.* Texas, USA: Worldware Publishing.

[Co91] Daniel D. Corkill, 1991. Blackboard Systems. *AI Expert* 6(9):40-47.

[IB02] Damian Isla and Bruce Blumberg, 2002. Blackboard Architectures. *In: Steve Rabin, ed. AI Game Programming Wisdom.* Rockland, MA, USA: Charles River Media, Inc., 333-344

[Or03] Jeff Orkin, 2003. Applying Blackboard Systems to FPS. *Unpublished presentation UT Austin .* http://web.media.mit.edu/

[Bo96] John R. Boyd, 1996. The Essence of Winning and Losing. *Unpublished lecture notes,* http://www.d-n-i.net/

[La05] Christopher E Larsen, 2005. *Light Infantry Tactics for Small Teams*. Bloomignton, Indiana, USA: Authorhouse.

[Fc04] “Far Cry”, 2004. http://www.ubi.com/UK/Games/Info.aspx?pId=547

[UBI] Ubisoft Entertainment, http://www.ubi.com/UK/default.aspx

[CRY] CryTek GmbH, http://www.crytek.com/

[CRE] “CryEngine1.x”, http://www.crytek.com/technology/cryengine-1x/

[CRS] “CryEngine1.x Sandbox, FarCry Edition” http://www.crytek.com/technology/cryengine-1x/

[CBL]“Code::Blocks” Open source C++ IDE, http://www.codeblocks.org/

[TEX] “TextPad” Text Editor, Helios Software Solutions, Longridge, United Kingdom http://www.textpad.com/

[CRM] “CryMod” FarCry game modding community portal, http://www.crymod.com/

[Ab04] Aberot et al, 2004. CryEngine Sandbox FarCry Edition User Manual v1.1. *Coburg, Germany: CryTek GmbH.*

[AI04] FarCry SDK AI Manual, 2004. *Coburg, Germany: CryTek GmbH.*

Acknowledgements

The author of this document would like to thank the following people for their advice and guidance during the execution of this project:

- NCCA staff: Jon Macey, Ian Stephenson, Hammadi Nait-Charif and Jian Jun Zhang.
- The administrators and members of the "CryMod" FarCry modding community portal at http://www.crymod.com/.

Spencer Drayton
Msc Computer Animation '06-'07
National Centre for Computer Animation (NCCA)
Bournemouth University