Multi-Agent System with Artificial Intelligence

Rabia Engin

MSc Computer Animation and Visual Effects

Bournemouth University

August, 2016

Abstract

Multi-agent systems widely uses to produce or reproduce behavioural animation of crowds such as pedestrians or cars in traffic. In such situations, it is important to have decision making ability for each individual to represent a realistic scene. Thus, artificial intelligence is closely related to multi-agent systems since it provides imaginary consciousness for agents separately. In this project, an A* pathfinding algorithm has been used to calculate shortest route for each agent between given starting and end positions by avoiding the obstacles within the scene. Furthermore, a collision detection algorithm has been applied to individuals to avoid collision with agents each other. Application has been written by using C++ and OpenGL.

Keywords: Multi-agent System, Artificial Intelligence, A* Pathfinding, Maze, Collision

Acknowledgement

I want to thank to the every person that I have been taking this course together and have chance to work with. I met a lot of extraordinary people throughout this year and learned from every single of them. I also want to thank to Jon Macey to run this significant programme. Finally, my last and the most special thanks goes to my dearest colleagues;

George Bitiușca,
Yasser Ashraf Mohsen,
Fannar Traustason &
Rebecca Kuo

for their patience. Their precious support ensured me to survive and complete this project.

Contents

List of Figures

1 Introduction

Multi-agent systems widely used by researchers to produce or reproduce behavioural animation of crowds such as pedestrians or cars in traffic. In such situations, it is important to have decision making ability for each individual to represent a realistic scene. Thus, artificial intelligence is closely related to multi-agent systems since it provides imaginary consciousness for agents.

In this project, an A* path finding algorithm has been used to calculate shortest route for each agent between given starting and end positions by avoiding the obstacles within the scene. Furthermore, a collision detection algorithm has been applied to individuals to avoid collision with each other. There are 3 basic collision types among agents; towards, glancing and away collisions. Towards collision occurs while agents came across and tries to move to each other's current position. Glancing collision occurs when agents needs to be in the same position in a certain time step. Third and last collision type named away collision occurs when an agent wants to move to another agent's current position from its behind.

There are different scenarios in the program to simulate different situations that agents can experience. First scenario includes a maze and 2 agents moves from different sides of the maze and moves through the maze in opposing directions. Second and third scenarios simulates towards and glancing collisions respectively with multiple agents. Last scenario includes a wall that separates the scene into 2 pieces with only one gate to simulate bottleneck situations with multiple individuals.

Various type of obstacles has been presented to user to create different kind of scenarios. One of the obstacle types is wall as mentioned above that sets obstacles to form a wall throughout a row or column between given points. Another type of obstacle is cottage that sets obstacles to all neighbours of a given point to form a cottage like geometry. It is also available to set obstacles to individual points.

Application has been written by using C++ and OpenGL. Implementation details will be given in next chapters.

2 Related Works

Many studies has been done in the field of multi-agent systems and artificial intelligence so far. Artificial intelligence is a group of machines that pretends human brain by making decisions based on circumstances. One of the very first use of AI in computer animation and games was in 1950s by creating a game that plays chess.

Even though, the artificial intelligent subject is such a massive are, in this particular project, 2 of its subtitles has been focused on. One of these subjects is path finding algorithms with A* in main focus. The other one is behaviours of agents in case of collision among each other.

Path finding problem first mentioned in the beginning of 19th century by W.R. Hamilton with a name of "Travelling Salesman". This problem were enquiring a solution for a salesman who will travel to several cities by visiting each of them ones and for all. However, "Travelling Salesman" problem did not take attention until 20th century. One of the very first algorithm has been developed by C.P. Tremaux. Breadth First Search, Bellman-Ford's, Dijkra's and A* path finding algorithms followed Tremaux' Depth First Search.

A* shortest path finding algorithm has been developed by Hart et al. (1972) and occupies the first place of most popular path finding algorithms rank. Numerious of new methods such as D* (Likhachev et al. 2005) and Theta* (Nash 2010) have been developed based on A* algorithm.

On the other hand many studies in group movement and collision detection has been done by researchers throughout the years. The flocking system concept has introduced for the first time by Reynolds in 1987. In this work Reynolds was using local rules for the individuals in the flocks. After a decade, Reynolds extended his work to an autonomous reactive behaviour. In this approach, Reynolds displayed a method for the flocks to deform themselves to avoid collision with other units while keeping the collision detection among individuals that forms the flocks (Reynolds 1999).

A study that combines flocking techniques with probabilistic roadmap approach has been done. The flocks were using the roads that created by probabilistic roadmap approach to reach their targets (Bayazit et al. 2004). On the other hand, dynamic structuring of flocks like centralized planning of motion has been improved by a new approach (Li and Chou 2003).

A part of crowd simulation that investigates units' movements in a virtual environment has received large amounts of attention over years around the new millennium. Feurtley used a space-time approach to predict collisions among agents

(Feurtley 2000) while Helbing and Molnr using social force model to simulate movements (Helbing and Molnr 1995). Other works in this field has used cellular automata model (Blue and Adler 2000) or focused on path finding that also uses behaviour simulation (Lamarche and Donikian 2004). Simulating human behaviour that concentrating particularly on collision avoidance also has been done (Rymill and Dodgson 2005). Their algorithms mostly based on psychology research.

Emulating the complexity of pedestrians in urban environments has been studied by Shao and Terzapoulos. Their approach integrated behavioural, motor and perceptual components. They have displayed the environment using data structures (Shao and Terzapoulos 2005).

Pelechano studied a multi-agent model named HIDAC to simulate flow of crowd in dynamic environments (Pelechano et al. 2007). After, he has classified existing methods in two general categories (Pelechano et al. 2008); continuum macroscopic models (Hughes 2002, Huang et al. 2009) and individual based microscopic methods (Helbing et al. 2000, Kirchner et al. 2002). Studies mostly has been done to simulate aspects of crowd movements and dynamics (Zheng et al. 2009).

Van den Berg introduced two level navigation method that uses model human interactions. These works are often based on Reciprocal Velocity Obstacles (Van den Berg et al. 2008).

3 Technical Background

3.1 Multi-agent Systems

A multi-agent system is a network of imagery individuals that interact each other to visualize circumstances. Since a large amount of situations includes multiple agents it can be challenging for individual capacities of problem solvers. Thus, researchers are using these systems to solve problems that requires multiple perspectives.

In this case, a multi-agent system has been used to reproduce certain environments that includes multiple individuals and their interaction to each other.

3.2 Artificial Intelligence

Artificial Intelligence is a subject that has been studied in various fields including robotics and games. One of the very first applications of artificial intelligence in games industry is well known game named Pac-Man that has very basic but interesting version of artificial intelligence (Pitmann 2011).

In the field of robotics artificial intelligence has been used to reach real time solutions in different kind of circumstances. This unique subject also found an application opportunity in multi-agent systems to give agents unique decision making process in given situations that also includes interacting of agents to each other.

Artificial Intelligence itself basically a group of machines that provides solution to problems. In this work, Finite State Machine with trivial "true" or "false" answers to basic questions used to create a smooth artificial intelligence for the agents within the system. By using this machine, a decision making opportunity has been provided to each of the individuals in situations such as they interact with each other or the environment.

3.3 A* Path finding Algorithm

A* path finding algorithm is a heuristic algorithm that uses a computational method towards optimality. To begin explaining A*, the environment has to be stated first. As a basic problem, an environment with a wall between start and end positions has been considered (**Figure 1**).

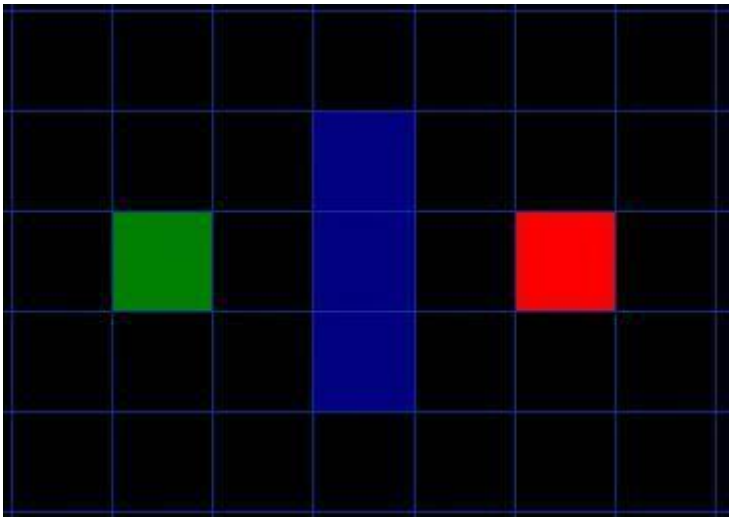


Figure1: Start position (green), end position (red), wall (blue). (Lester 2005)

It is clear to see that, the environment has been divided to squares and each opponent occupies one of the nodes. Using such a grid based approach is enormously handy in calculating the path between 2 given points.

According to A* path finding algorithm, each node in the grid has 3 different scores named G score, H score and F score. G score is the movement price to reach the current node from the start node (green). H score is the distance between current node and end node (red). Finally, F score is basically adding G score and H score together. The most important difference between G score and H score is that to calculate the G score obstacles has to be considered while it is unnecessary for calculating H score.

Algorithm begins with the start node and checks all of neighbour nodes to confirm if they are occupied by an obstacle. There are 8 neighbours for each node unless the node is at the edges or corners of the grid. If there is no obstacle in the neighbour node than the node gets marked as open. Algorithm also calculates the G, H and F scores of these nodes in this step and current node gets assigned as parent node of this neighbour nodes. Program calculates G score of a node by getting G score of the node's parent node and adding the distance between parent node and neighbour node to it. This distance is 10 for vertical or horizontal movements and 14 for diagonal movements. This measures come from unit distances 1 and 1.4 (approximate value of $\sqrt{2}$ the length of hypotenuse in Pythagoras' Theorem) multiplying by 10 to reduce the calculation expense. To calculate H score, program uses the same distance measures by counting the diagonal and vertical/horizontal steps and multiplying this numbers vith 14 and 10 respectively. As last step, program adds these G score and H score to get F score which will be used to pick the next node in the potential path. Neighbours of start node that represents the first step of the path has given below (**Figure 2**).

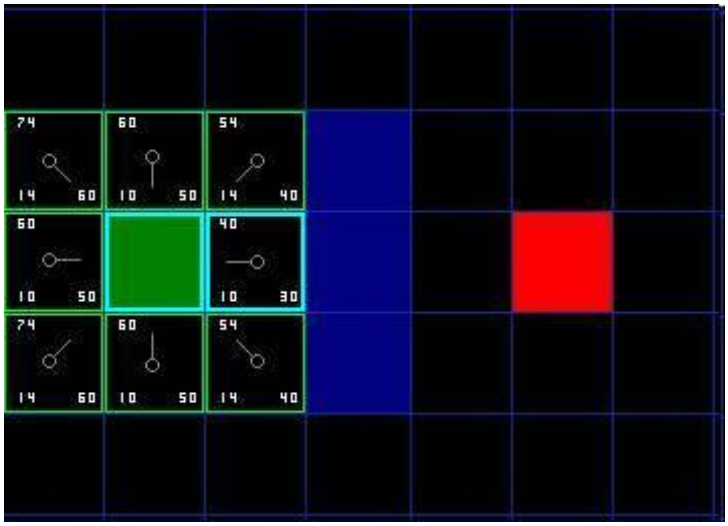


Figure 2: Neighbours of start node and scores of each node. F score (top-left), G score (bottom-left) and H score (bottom-right). (Lester 2005)

Next step in the process is finding the lowest F score in open nodes. After finding the node with lowest F score, this node getting marked as a closed node. Next, finding neighbours and lowest F score process starts again until reach the target, in this case, end node (red). Pseudo code of the algorithm has given below (**Figure 3**).

```
OPEN //the set of nodes to be evaluated
CLOSED //the set of nodes already evaluated
add the start node to OPEN

loop
 current = node in OPEN with the lowest f_cost
 remove current from OPEN
 add current to CLOSED

 if current is the target node //path has been found
  return

 foreach neighbour of the current node
  if neighbour is not traversable or neighbour is in CLOSED
   skip to the next neighbour

  if new path to neighbour is shorter OR neighbour is not in OPEN
   set f_cost of neighbour
   set parent of neighbour to current
   if neighbour is not in OPEN
    add neighbour to OPEN
```

Figure 3: A pseudo code for A* path finding algorithm. (Lague 2014)

When the next node with lowest F score is equals to target, algorithm goes out of loop and creates path by getting the parent node of current node until reach back to the start node from end node. Final look and the path of the example given above has given in **Figure 4**.

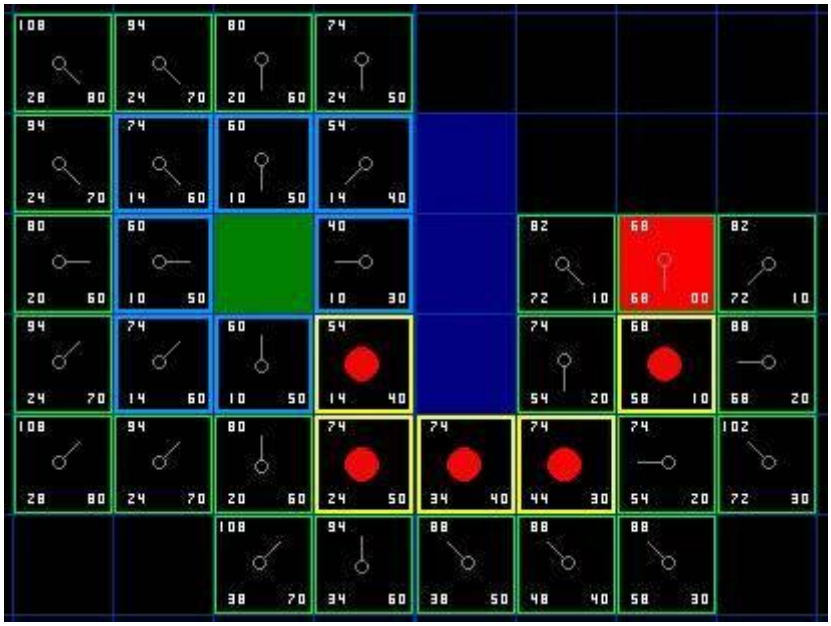


Figure 4: Final look and path (nodes that includes red dotes). (Lester 2005)

With this final step the A* path finding algorithm gets finalized and the result can be used in any application. In this work, an A* path finding algorithm that outlined above has been used to calculate paths of agents individually between given start and end positions.

3.4 Collision Avoidance

In this work, a multi-agent system has been created based on human behaviour to simulate the navigation process. One of the main points is collision detection among agents to create more accurate dynamic scenes. Thus, every agent within the system should check if there is any collision occurs in oncoming time steps. To avoid the collision it is also important to predict the type of collision. In real life, there is 3 type of collisions named towards, glancing and away (Foudil 2006). Furthermore, a bottleneck type of collision has been added to the case. Collision avoidance between agents can cause many problems when there is many agents interacting. Solutions for only 2 agents might not work for situations that involves more agents. These collision types detail's described below.

3.4.1 Towards Collision

Towards collision occurs when agents moving into each others' space (**Figure 5**). In such situations general human behaviour is moving side to give way to each other. Studies show that there is 3 options for agents to avoid towards collision. These are

changing direction, changing speed or changing both of them. If none of these solutions avoids collision then agent simply stops to wait other one make the movement for avoidance. In this work, when a towards collision occurs, one of the agents move side to give way while other carry on its original path.

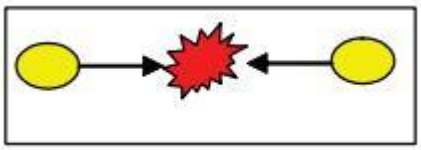


Figure 5: Towards collision. (Foudil 2006)

3.4.2 Glancing Collision

Glancing collision occurs when agents move into same direction. If 2 agents are in the same position in certain time step that means there is collision (**Figure 6**). Solution for this kind of collision is similar to towards collision. Agents can change direction or speed to avoid the collision. In this work, when glancing collision occurs between agents one of them simply stops until the other one pass the position that has been predicted to cause collision.

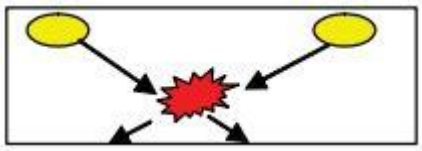


Figure 6: Glancing collision. (Foudil 2006)

3.4.3 Away Collision

Away collision occurs when an agents move towards another agents' position from behind (**Figure 7**). To avoid away collision the agent that locates behind of other can stop, slow down or move faster to overtake the agent in front of it. In this work, since all of the agents have same speed, the agent that behind the other one waits until other agent move away from its current position.

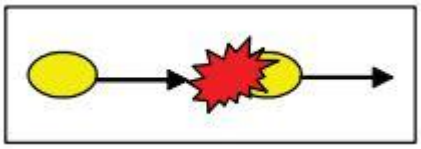


Figure 7: Away collision. (Foudil 2006)

3.4.4 Bottleneck Collision

Bottleneck collision occurs when several agents tries to move towards a small gate such as entrance into subways or doors (**Figure 8**). In such situations, the priority to take the next step is given to the agent that closest to the centre of bottleneck. To avoid collision around bottlenecks, agents wait for the ones closest to the gate so they can have space to move forward.

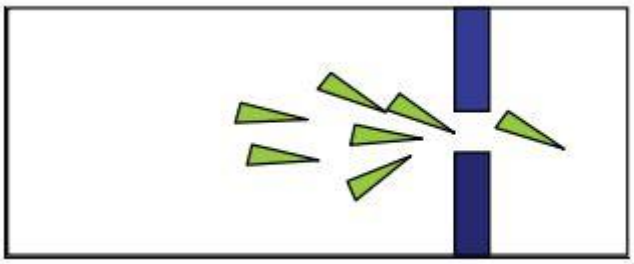


Figure 8: Bottleneck collision. (Foudil 2006)

4 Implementation

In this work, a multi-agent system has been used to reproduce certain environments that includes multiple individuals and their interaction to each other and environment. Main focus of this project is finding shortest path between 2 given points in an environment that also includes obstacles and other agents. Second, avoiding the collision among agents within the system. A* path finding algorithm already handles the collision detection of agents with obstacles that located within the system. Thus, collision detection among agents is the most important case in this work after finding the shortest path. The A* path finding algorithm that outlined above has been used in this work.

A principal artificial intelligence has been created for the agents within the system to be able to apply the observed general behaviours of human instead of using complex imaginary brain systems. A trivial decision making opportunity has been provided to each individual for the cases of path finding and collision detection.

There are 4 type of collisions in this work to observe agents behaviours. These collisions types are towards, glancing, away and bottleneck collisions. There are also several versions of collision avoidance method for mentioned collision types. Agents can give way to others by moving side or stopping and simply waiting for the other ones to pass the point that predicted for the collision. These methods work for the towards, glancing and away collisions. Bottleneck collision is slightly different since in such situations general approach is giving the priority to the agent that stands closest to the centre of the gate. Thus, this certain agent has the right to pass through the bottleneck first.

4.1 Class Structure

Class diagram for this project has given in **Appendix A**. There is 5 major classes in the work named Node Class, Grid Class, Agent Class, Crowd Class and finally Scene Class. These classes has described below respectively.

Node Class: Includes the node attributes that is necessary to calculate the path of each agent using A* path finding algorithm. However, it cannot calculate the path by itself since A* uses a grid based approach that uses multiple nodes in each time step. Thus, this class only provides information that grid class needs.

Grid Class: Includes the attributes, minor and major methods to calculate the path. It also uses the attributes of nodes and has access to them. Thus, this class can be considered as a parent class to Node Class. This class can only calculate the path

for only one agent in a single time step. As a result, it serves its results to Crowd Class to be able to calculate the paths for all of the agents in the system.

Agent Class: Similar to Node Class it includes the attributes of agents and provides them to Crowd Class so Crowd Class can have the knowledge of each agents' condition regarding to the others.

Crowd Class: This class has access to all of the classes given below. It takes the start and end positions data from agent and pass them to the grid. After, grid finishes calculating the path, Crowd Class take the path and considers if there is any collision prediction. After making the necessary changes to avoid any possible collision assigns the path to the related agent.

Scene Class: Simply draws the scene. It has access to entire system to be able to get data that will be shown in the window. Scene Class also have the ability to assign values to create new and unique scenarios within the system.

4.2 Scenarios

There are 4 different scenarios in this work to represent different kind of situations that can happen to individuals. These scenarios named as maze, towards, glancing and bottleneck. Details of these environments given below.

4.2.1 Maze

In maze scenario, the main focus is finding the shortest and optimal path for multiple individuals. Thus, there are 2 agents in the scene using the same maze but different begin and final positions. These agents are calculating their path within the maze individually and when they came across to each other avoiding the collision (**Figure 9**).

Another method to find the way within the maze could be one of the most classic methods; touching one wall during the entire journey. However, this approach requires agent to go into various dead ends on the way to exit. Thus, it does not uses the shortest path. Thus, it is not the optimal solution and we choose to use A* instead of this blind search method.

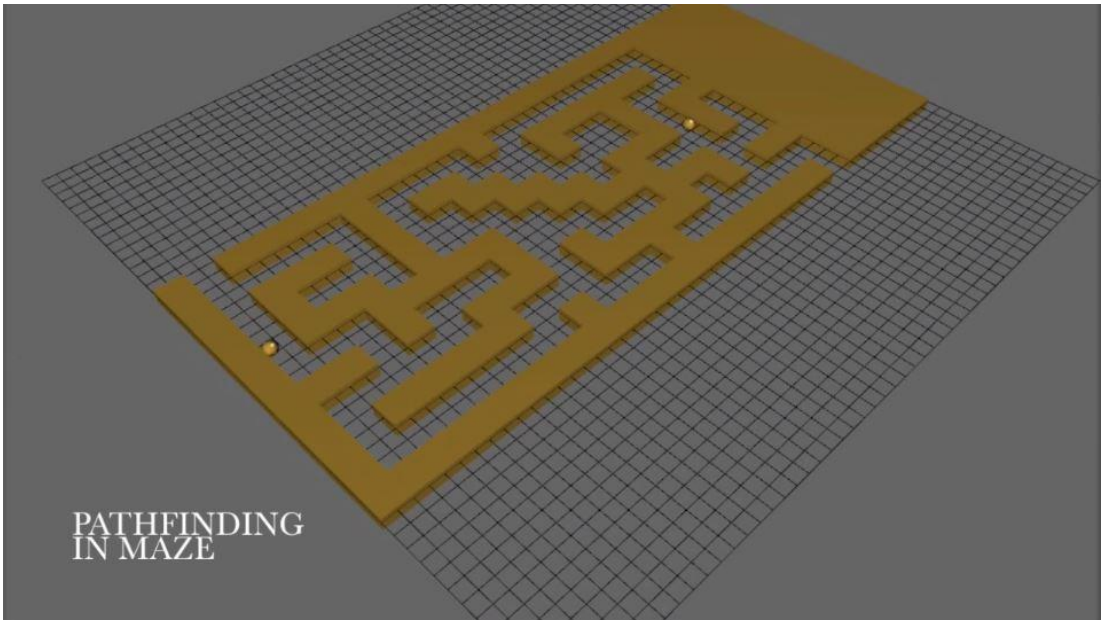


Figure 9: Maze.

4.2.2 Collisions

There are 2 collision scenarios representing towards and glancing collisions. These scenes can be seen in **Figure 10** and **Figure 11** respectively.

In towards collision scene, there are 6 agents located to form a line like structure and moving towards to each other. When a collision has predicted, one of the agents that will interact moves side to give way the other one. After the other agent's passed through the node that they supposed to interact, first agent moves back to its original position and follows remaining of its route (**Figure 10**).

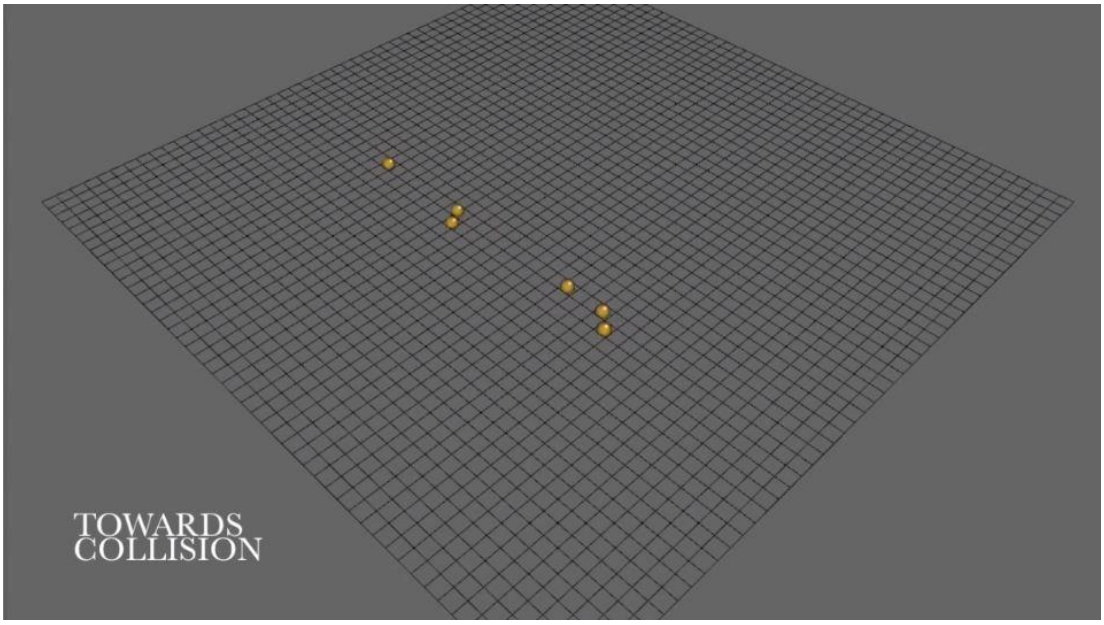


Figure 10: Towards collision.

In glancing collision scene, there are 4 agents in the system to interact each other. One of the agents moves vertically while the rest of them moves horizontally. First agent interacts with all other agents in the system. Thus, there are 3 interactions and when an agent came across with one another one of them waits for the other one to pass the position that predicted for collision (**Figure 11**).

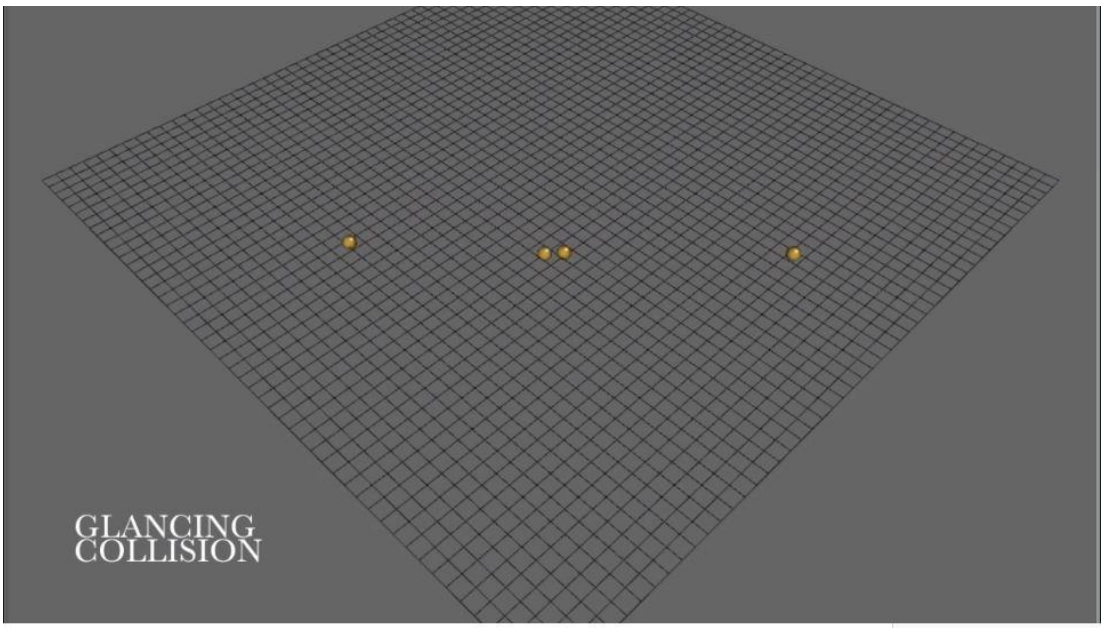


Figure 11: Glancing collision.

4.2.3 Bottleneck

In bottleneck scenario, there is a wall that separates the scene into 2 pieces. The wall has only one gate that can let only one agent in a time step. The reason of this wall is for creating a scene that can cause a bottleneck situation. To achieve that situation, there are 5 agents in one side of the wall and their target positions are in the other side of the wall. Thus, this agents has to across the scene and pass the wall by using the gate. Furthermore, the agents have been located in places where they can reach the gate approximately at the same time (**Figure 12**).

This scene has a significant identity considering to other scenes that involves other types of collisions. The reason of this significance is involving a bottleneck environment. In such situations, there is a priority rule within the system to avoid the collision. The agents that closest to the centre of the group, as a result closest to the gate, has the priority to move through the bottleneck. In other cases, any agent could give way to the other one but in this specific case one certain agent at a time can move through and the rest of the agents has to wait until there is sufficient space for them to move.

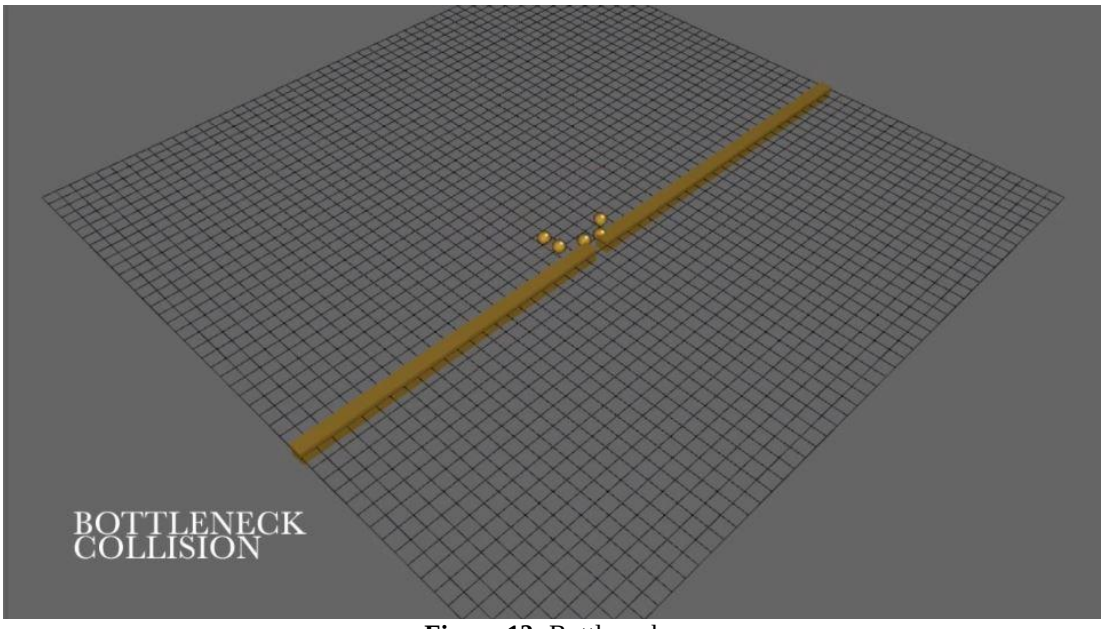


Figure 12: Bottleneck.

4 Conclusion & Future Works

Multi-agent systems mainly used in studies to simulate behaviours of crowds such as pedestrians in a traffic, guests in an exhibition or competitors in a triathlon. In such situations, for the agents it is important to be able to move as an individual and make decisions considering their own and environmental conditions. Thus, multi-agent systems has to be considered as a subject that related to artificial intelligence.

In this project, a multi-agent system with a basic artificial intelligence has been used to simulate different type of scenarios in various environments. In the scenarios, there are mazes, collisions and bottlenecks to create various type of effects.

One of the main focuses of this work was producing scenes that involves agents that can find their optimal path between 2 given point and throughout obstacles that forms complex shapes such as mazes. To achieve that purpose A* path finding algorithm has been implemented since it is one of the most popular solution for path finding algorithms. Another point of view to solve a path finding problem would be a blind search algorithm that literally uses one of traditional methods to find way with in a maze by touching on of the walls in the beginning and follows it until reach the exit of the maze. However, this would not be the optimal solution for path finding problem since it requires to go in number of unnecessary dead ends.

Second part of the project was collision detection among agents in various type of situations. There are 3 main type of collisions named towards, glancing and away collision. To avoid collision there are several choices such as waiting or moving aside to give way. Agents can decide what to do considering the situation that they are in. Furthermore, a bottleneck collision has been added to the system to display there can be more complex situations that involves collision detection.

The system can be extended to more complex scenes with larger number of agents and obstacles. There are also many ideas for future works. First of all, this work requires agents to move individually. However, it can be extended to move in small groups. Second, agents are travelling between pre-assigned start and end positions. They can be given duties to visit certain places throughout their journey. Third, decision making mechanism for agents is quite limited. This can be extended to achieve more complex and relatively realistic simulations. Finally, a vector field approach can be used to navigate agents within the system to reach their target. Those vector fields can be assigned as direction labels in a city or exhibit that displays way to go to reach the desired target.

References

Arun, A. 2014. Pathfinding Algorithms in Multi Agent Systems. *Dissertation. Bournemouth University.* https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc15/01Amitha/thesis.pdf [Accessed 20.08.2016]

Bayazit, O. B., Lien J. M., Amato, N., M., 2004. Better flocking behaviours using rule based roadmaps. *In Algorithmic Foundations of Robotics V, Springer Tracts in Advanced Robotics 7, Springer-Verlag Berlin Heidelberg, pp. 95-111.*

Blue, V., Adler, J., 2000. Cellular automata model of emergent collective bi-directional pedestrian dynamics. *In Artificial Life VI, the Seventh International Conference on the Simulation and Synthesisof Living Systems.*

Cogne, V., 2010. A Star Pathfinder C++ Implementation. http://xpac27.github.io/a-star-pathfinder-c%2B%2B-implementation.html [Accessed 20.08.2016]

Daniel K., Nash A., Koenig S. and Felner A., 2010. Theta*: Any-angle path planning on grids. J. Artif. Intell. Res. (JAIR), 533–579

Feigenbaum, E., Cohen, P., Barr, A., 1981. The Handbook of Artificial Intelligence. *Stanford Calif. : HeuricTech Press; Los Altos, Calif.: William Kaufmann.*

Ferber, J., 1999. Multi-Agent Systems An Introduction to Distributed Artificial Intelligence *Essex: Longman.*

Feurtley, F., 2000. Simulating the Collision Avoidance Behaviour of Pedestrians. *Master's thesis. Department of Electronic Engineering, University of Tokyo.*

Foudil, C., Noureddine, D., 2006. Collision Avoidance in Crowd Simulation with Priority Rules. *European Journal of Scientific Research, ISSN 1450-216X Vol.15 No.1, p. 6-17.*

Hart P. E., Nilsson N. J. and Raphael B., December 1972. ucorrection/u to ”a formal basis for the heuristic determination of minimum cost paths”. *SIGART Bull., (37), 28–29.*

Helbing, D., Molnr, P., 1995. Social force model for pedestrian dynamics. *Physical Review, 51, pp. 4282-4286.*

Helbing, D., Farkas, T.V., 2000. Simulating dynamical features of escape panic. *Nature, vol. 407(6803), pp. 487-490.*

Huang, L., Wong, S., Zhang, C., Shu, W., Lam, W.H., 2009. Revisiting Hughes' dynamic continuum model for pedestrian flow and the development of an efficient

solution algorithm. *Transportation Research Part B: Methodogical, vol.43(1), pp. 127-141.*

Hughes, R.L., A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodogical, Vol. 36(6). pp. 507-535.*

Kirchner A., Shadschneider, A., 2002. Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. *Phsica A: Statistical Mechanics and its Applications, vol.312(1), pp. 260-276.*

Koenig S., Likhachev M. and Furcy D., May 2004. Lifelong planning a*. *Artif. Intell., 155(1-2), 93–146.*

Korf R. E., March 1990. Real-time heuristic search. *Artif. Intell., 42 (2-3), 189–211*.

Lague, S., 2014. A* Pathfinding (E01: Algorithm Explanation). https://www.youtube.com/watch?v=-L-WgKMFuhE [Accessed 20.08.2016].

Lamarche, F., Donikian, S., 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *In Computer Graphics Forum, vol. 23, pp. 509-518.*

Lester, P., 2005. A* Pathfinding for Beginners. http://www.policyalmanac.org/games/aStarTutorial.htm [Accessed 20.08.2016].

Lester, P., 2005. Using Binary Heaps in A* Pathfinding. http://www.policyalmanac.org/games/binaryHeaps.htm [Accessed 20.08.2016].

Li, T., Y., Chou, H., C., 2003. Motion planning for a crowd of robots. *In International Conference on Robotics and Automation (ICRA). IEEE Press, San Diego, CA.*

Likhachev M., Ferguson D., Gordon G., Stentz A. T. and Thrun S., June 2005. Anytime dynamic a*: An anytime, replanning algorithm. *In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS).*

Nash A., 2010. Theta* any angle paths. http://aigamedev.com/open/ tutorials/theta-star-any-angle-paths/ [Accessed 20.08.2016].

Patel A., 2010. Introduction to a*. http://theory.stanford.edu/~amitp/GameProgramming/ [Accessed 20.08.2016].

Pelechano, N., Allbeck, J., Badler, N., I., 2007. Controlling individual agents in high-density crowd simulation. *In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation Eurographics Association, San Diego, CA, USA, pp. 108-118.*

Pelechano, N., Allbeck, J., Badler, N., I., 2008. Virtual Crowds: Methods, Simulation and Control. *Synthesis Lectures on Computer Graphics and Animation, vol.3.*

Pittman, J., 2011.The Pac-Man Dossier. Available http://home.comcast.net/~jpittman2/pacman/pacmandossier.html#Table Of Contents [Accessed 20.08.2016].

Raghunandan, S. 2013. Multi Agent Systems with Artificial Intelligence. Dissertation. Bournemouth University. https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc13/05/thesis/Multi%20Agent%20System%20with%20Artificial%20Intelligence_Thesis.pdf [Accessed 20.08.2016].

Reynolds, C. W., 1987. Flocks, herds and schools: A distributed behavioral model. *In Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '87, New York, NY, USA. ACM,* 25–34.

Reynolds, C. W., 1999. Steering behaviours for autonomous characters. *In Game Developers Conference.*

Rouse M., 2009. Flocking like it's 1999. http://whatis.techtarget. com/definition/heuristic [Accessed 20.08.2016].

Rymill, S., J., Dodgson, N., A., 2005. A pyschological-based simulation of human behaviour. *Theory and Practice of Computer Graphics, EG UK, 2005, pp 229-236.*

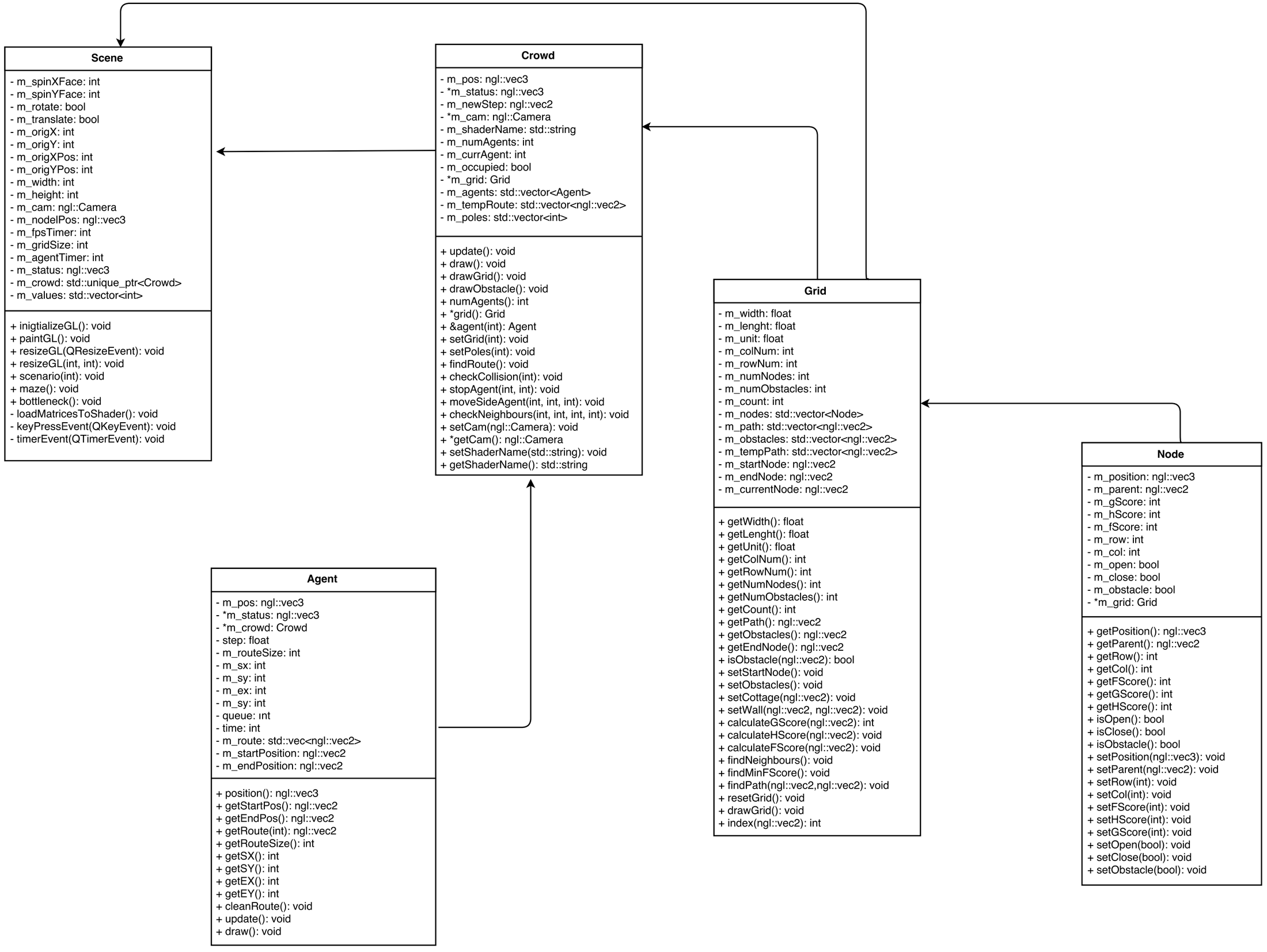
Shao, W., Terzopoulos, D., 2005. Autonomous Pedestrians. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation. USA. pp. 19-28.*

Stentz A., 1995. The focussed d* algorithm for real-time replanning. *In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 1652–1659.*

Woolridge M. and Wooldridge M. J., 2001. Introduction to Multiagent Systems. *John Wiley & Sons, Inc., New York, NY, USA.*

Van den Berg, J., Patil, S., Seawall, J., Manocha, D., Lin, M.C., 2008. Interactive navigation of individual agents in crowded environments. *In Proceedings of the 2008 Symposium on Interactive 3D graphics and Games, ACM, Redwood Shores, CA, USA, pp. 139-147.*

Zheng, X., Zhong, T., Liu, M., 2009. Modelling crowd evacuation of a building based on seven methodological approaches. *Building Environment, vol44(3), pp. 437-445.*


Scene
- m_spinXFace: int
- m_spinYFace: int
- m_rotate: bool
- m_translate: bool
- m_origX: int
- m_origY: int
- m_origXPos: int
- m_origYPos: int
- m_width: int
- m_height: int
- m_cam: ngl::Camera
- m_nodeIPos: ngl::vec3
- m_fpsTimer: int
- m_gridSize: int
- m_agentTimer: int
- m_status: ngl::vec3
- m_crowd: std::unique_ptr<Crowd>
- m_values: std::vector<int>
+ inigtializeGL(): void
+ paintGL(): void
+ resizeGL(QResizeEvent): void
+ resizeGL(int, int): void
+ scenario(int): void
+ maze(): void
+ bottleneck(): void
- loadMatricesToShader(): void
- keyPressEvent(QKeyEvent): void
- timerEvent(QTimerEvent): void
Crowd
- m_pos: ngl::vec3
- *m_status: ngl::vec3
- m_newStep: ngl::vec2
- *m_cam: ngl::Camera
- m_shaderName: std::string
- m_numAgents: int
- m_currAgent: int
- m_occupied: bool
- *m_grid: Grid
- m_agents: std::vector<Agent>
- m_tempRoute: std::vector<ngl::vec2>
- m_poles: std::vector<int>
+ update(): void
+ draw(): void
+ drawGrid(): void
+ drawObstacle(): void
+ numAgents(): int
+ *grid(): Grid
+ &agent(int): Agent
+ setGrid(int): void
+ setPoles(int): void
+ findRoute(): void
+ checkCollision(int): void
+ stopAgent(int, int): void
+ moveSideAgent(int, int, int): void
+ checkNeighbours(int, int, int, int): void
+ setCam(ngl::Camera): void
+ *getCam(): ngl::Camera
+ setShaderName(std::string): void
+ getShaderName(): std::string
Grid
- m_width: float
- m_lenght: float
- m_unit: float
- m_colNum: int
- m_rowNum: int
- m_numNodes: int
- m_numObstacles: int
- m_count: int
- m_nodes: std::vector<Node>
- m_path: std::vector<ngl::vec2>
- m_obstacles: std::vector<ngl::vec2>
- m_tempPath: std::vector<ngl::vec2>
- m_startNode: ngl::vec2
- m_endNode: ngl::vec2
- m_currentNode: ngl::vec2
+ getWidth(): float
+ getLenght(): float
+ getUnit(): float
+ getColNum(): int
+ getRowNum(): int
+ getNumNodes(): int
+ getNumObstacles(): int
+ getCount(): int
+ getPath(): ngl::vec2
+ getObstacles(): ngl::vec2
+ getEndNode(): ngl::vec2
+ isObstacle(ngl::vec2): bool
+ setStartNode(): void
+ setObstacles(): void
+ setCottage(ngl::vec2): void
+ setWall(ngl::vec2, ngl::vec2): void
+ calculateGScore(ngl::vec2): int
+ calculateHScore(ngl::vec2): void
+ calculateFScore(ngl::vec2): void
+ findNeighbours(): void
+ findMinFScore(): void
+ findPath(ngl::vec2,ngl::vec2): void
+ resetGrid(): void
+ drawGrid(): void
+ index(ngl::vec2): int
Node
- m_position: ngl::vec3
- m_parent: ngl::vec2
- m_gScore: int
- m_hScore: int
- m_fScore: int
- m_row: int
- m_col: int
- m_open: bool
- m_close: bool
- m_obstacle: bool
- *m_grid: Grid
+ getPosition(): ngl::vec3
+ getParent(): ngl::vec2
+ getRow(): int
+ getCol(): int
+ getFScore(): int
+ getGScore(): int
+ getHScore(): int
+ isOpen(): bool
+ isClose(): bool
+ isObstacle(): bool
+ setPosition(ngl::vec3): void
+ setParent(ngl::vec2): void
+ setRow(int): void
+ setCol(int): void
+ setFScore(int): void
+ setHScore(int): void
+ setGScore(int): void
+ setOpen(bool): void
+ setClose(bool): void
+ setObstacle(bool): void
Agent
- m_pos: ngl::vec3
- *m_status: ngl::vec3
- *m_crowd: Crowd
- step: float
- m_routeSize: int
- m_sx: int
- m_sy: int
- m_ex: int
- m_sy: int
- queue: ınt
- time: int
- m_route: std::vec<ngl::vec2>
- m_startPosition: ngl::vec2
- m_endPosition: ngl::vec2
+ position(): ngl::vec3
+ getStartPos(): ngl::vec2
+ getEndPos(): ngl::vec2
+ getRoute(int): ngl::vec2
+ getRouteSize(): int
+ getSX(): int
+ getSY(): int
+ getEX(): int
+ getEY(): int
+ cleanRoute(): void
+ update(): void
+ draw(): void