PROTOPLAY

MASTERS THESIS

SUNDARARAJAN SRINIVASAKANNAN

N.C.C.A BOURNEMOUTH UNIVERSITY

August , 2009

Abstract:

This project aims at the creation of the functional prototypes of games and applications that can be used in the production of miniature two dimensional games. Seven different games and applications were produced during the course of the project. Game C++ , OpenGl, SDL and QT were used in the production of the games and the applications.

The fundamental objective of the development scenario was showcasing good coding and programming practices and creating the features of the game design within the available short time limit. Hence more concentration was provided towards the game development and coding rather than the game design. The applications were developed with good usability features while the games were created using the existing game design.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Document Overview

Any computer game is an amalgamation of the creativity, power of visualisation, programming sense and various other talents of any game developer .

Rapidly prototyping a game idea into a small version of a game provides an insight into the nature and feature of the proposed game. This prototyping bears many advantages on the evaluation of the game. This thesis is an outline of the production process and the overview of the hurdles faced during this process with an outline on the development of the game as such.

1.2 Introduction

'A game is a series of interesting choices' —Sid Meier

The Video game industry is one of the biggest and the fastest growing entertainment sector accumulating billions of dollars worldwide. In 2009, even during the period of economic recession faced by all industries world wide, The video game industry has showed a double digit growth rate (cnet news 2009)

This industry requires the best creative talent pool encompassing people around the globe in various job disciplines. But owing to the heavy competition every game that is being developed must include a wide array of highly innovative and creative features that are different from the other games.

One of the activities that is followed during many software development process is Software prototyping in which small incomplete versions of the software programs is produced in the quickest possible time. These small applications are called prototypes. These prototypes simulate only very few aspects and features of the original program but makes the evaluation of the idea of developer much more easier due to the fact that we can actually try out the features rather than interpreting the model. (wikipidea, 2009)

What has been tried through the course of this project is to extend the idea of software prototyping to games and try to create game prototypes- miniature versions of games and applications that can be used to create games that can showcase the core features which can be extended for the production of the game and the application.

Seven different games and application were produced during the course of the project . Their key features and development strategy are explained below

Applications :

1. *Cutey Cute:* This is dynamic particle system editor designed using QT, C++, Image Magick and Opengl . The purpose of this particle system editor was to help in the production of sprites. Output of the particle system editor can be rendered out as a tiff image and can be used as sprites.
2. *Pick-a-Path:* Pick a path is a little AI tool which allowed users to plot paths of a character on map and make it move along the path. A little pac-man game was coded using this prototype to show how this application in action.
3. *L-SysViz* : L-SysViz is a application for producing 2-d L-Systems which can be used in games. This was coded for using QT for GUI and Opengl for rendering data and C++ for the algorithms.

Games:

4. *Funky Blocks:* Funky blocks is a recreation of "Tower Blocks" by Mind Jolt games . What was tried is to re-create the flash game idea using OpenGl, C++ in the shortest possible time. This was built in 5 days.
5. *Golli:* Golli is a game where the basics of a graph structure were tried out . The game design is based on scaled down version of Chinese checkers where we remove coins from a board by hopping one coin over another until one final coin remained.
6. *Fire*: This game extended the functionalities of the particle systems into a little game where we destroy little objects by burning them down. This was the usual space shooter game with a little twist.
7. *Chicken Crossing:* This game was based on the game design of "Chicken Crossing the road "which is one of the popular flash games available online where we guide a chicken from one end of the road to another. This was built in 5 days using C++ and Open gl with SDL for interaction.

Chapter 2

Previous Work:

2.1 Software Prototyping

Software Prototyping is a activity where incomplete versions of the software are built. This initial version provides very few features of the original design but provides the user a hands on experience on the features that the developer has in mind(Wikipedia,2009). This process of software prototyping has many benefits. First and foremost it allows the user to have a hand on experience on the features that will be available on the final version of the product. Also it gives the developer an idea of the milestones that needs to be met and the time that would be required to complete the original product. Also this will help the developers get feed backs from others early in the project and helps them to have a design to work with and compare through the course of development. The brief idea of Prototyping would include three steps.

1. Identify basic requirements
2. Develop and Prototype
3. Review of the Prototype
4. Revise and Enhance the Prototype (Wikipedia,2009)

2.2 EXPERIMENTAL GAMEPLAY PROJECT

The initial of this project is based on a project done in 2005 by a team of 4 grad students from Carnegie Mellon University. The project titled "Experimental Game play Project" began as a project pitched at the Entertainment Technology Centre at Carnegie Mellon University. The rules for production of the games were simple.

- Each game needs to be made in less than 7 days
- Each game needs to be made individually
- Each game needs to have a theme (Blomquis et al. 2009)

This project was a instant hit and received a lot of media attention and was publicised a lot as is evident from the fact that BusinessWeek published a article on this project(**Jana.R., 2005)** and gamesutra had a article on

it as well (**Gabler**.K *et.al* 2005) with G4TV making a interview on this project with the project members.

The following extract is taken from the blog of one of the student involved in the original project .

> *"As the project progressed, we were amazed and thrilled with the onslaught of web traffic, with the attention from gaming magazines, and with industry professionals and academics all asking the same questions, "How are you making these games so quickly?" and "How can we do it too?" Though we successfully met our goal of making over 50 games, we realized that this project had become much less about the games, and much more about the crazy development process"*
>
> **(Jana.R., 2005)**

*Fig1 : Some Examples from the games done in the EGP (***Shodhan.S [no date])**

As we can see the projects were more of prototypes intended to check the worthiness of the idea rather than adding very high graphics and high utility features to it . In other words the project was intended to prototype the design of the idea rather than to create a game.

Chapter 3

Technical Background:

3.1 OpenGL

OpenGl is a cross platform API used for writing applications that produce 2D and 3D computer graphics applications .The basic aim of Open gl is to pipeline is to accept primitives and convert them into pixel information that can be displayed onto the screen. OpenGl is a low -level procedural API requiring the programmer to dictate the exact steps that will render the scene in a particular way. For this it uses a graphics pipeline called OpenGL state machine which has been described below by means of the diagram

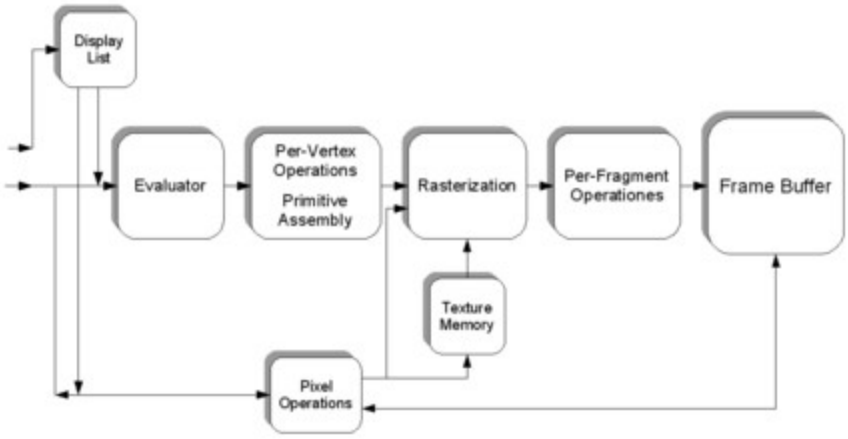

Fig 2: Opengl State machine (Wikipedia.,[no date])

For example to draw a triangle on the screen we need to follow the following steps .

- STEP 1: Clear the colour using glClear(GL_COLOR_BUFFER_BIT)
- STEP 2: Set up projection matrix using glMatrixMode
- STEP 3: Load the identity matrix using glLoadIdentity
- STEP 4: Translate to where the image needs to be rendered
- STEP 5 : Call glBegin to state the start of drawing routine
- STEP 6: Call the three vertices in order pushing them onto the stack
- STEP 7: Exit from the drawing routine using glEnd

A complete documentation of the working of Opengl is available from various sources a few of which has been mentioned in the references.

3.2 Simple Direct Media Layer

Simple Direct Media Layer (SDL) is a cross platform multimedia library providing access to the keyboard, mouse, joystick and other 3D hardware via OpenGl and a 2D video frame buffer (SDL, 2008). This works with C++ and supports cross platform applications.

SDL works on a principle of sub-systems that need to initialized and operated upon individually. The subsystems include the Video, Audio , the time Joystick and the CD-Rom subsystems. The initial three subsystems have been widely used through the course of this project.

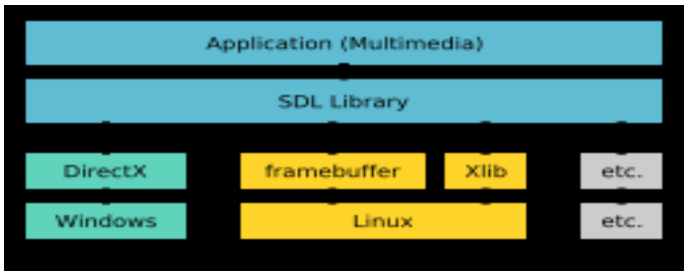

Fig:3 Abstraction layers of SDL(wikipedia 2008)

3.3 QT

QT is a cross platform application that is used to generate the UI frame work. Applications written in QT can be ported across various plat forms easily. QT works with a intuitive C++ library and is Opengl compatible when a QT-opengl library is included in it. QT creator is the UI interface design tool for QT providing a visual UI management interface. The newest version of QT provides features for C++ code editing debugging and code management as well.

QT works on the principle of signals and slots. All the UI widgets in qt are linked to private slots in the program. So when the UI- widget gets the proper signal it sends a call to the proper slot and invokes it . This is used a lot in the application development process.

A brief algorithmic example to illustrate the principle of signals and slot is given below.

- Define the slot in the header file as a private slot of the class
- Link the signal and the slot in the cpp file. This linking can also be done visually using the QT Creator tool. In code the link is made with the likes of the following definition:
- *connect(ui->Elen,SIGNAL(editingFinished( )),this, SLOT(recalculateBounds()));*
- This command links the ui element called Elen to the slot recalculateBounds when the signal editing Finished is achieved. All the ui elements are derived from the Qwidget class and have a

few common functionality . But most of the widgets are also extended to add additional functionality and signals.

- Then the slots are defined in the program which does what it needs to do.

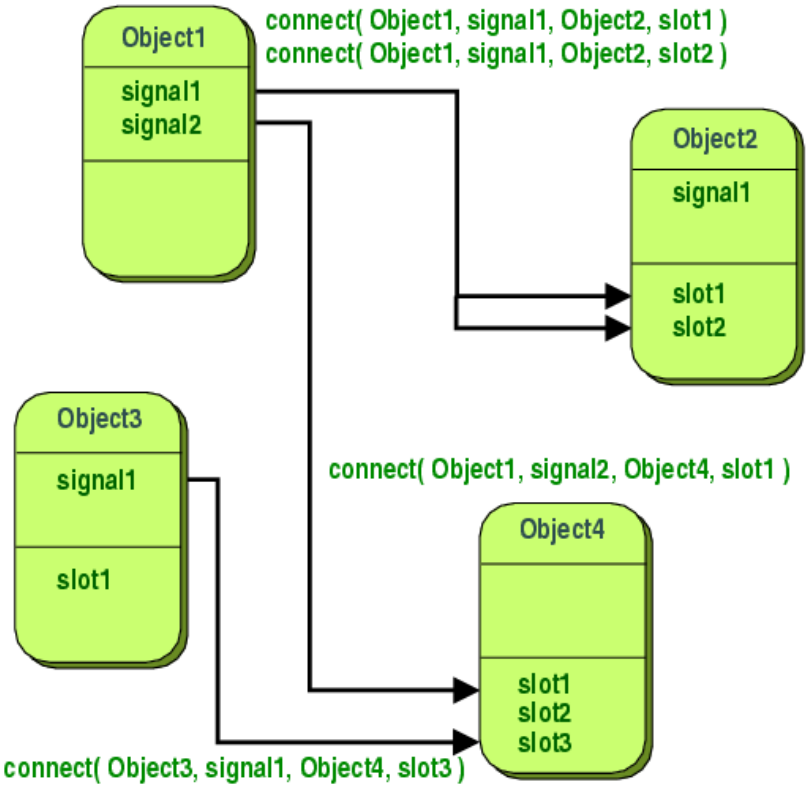

Fig 4: Signals and slots in QT(Troll Tech, 2008)

Chapter 4

AIM AND PROGRESS

4.1 Aim

The aim of the project was to concentrate on prototyping mini games and mini game applications in the shortest possible time over the period of 10 weeks. Concentration and focus of the project was towards writing nice well documented and understandable code rather than the visual and graphical appeal of the games. Also since prototypes of the games were aimed concentration was aimed at reproducing as many core features of the final product as possible. The aim of the project also was made to write clean , correct and extensible code with properly documented features .

4.2 Progress

The time-line of the project has been briefly described . The table below gives the approximate number of days that was taken to prototype the individual designs.

Prototype	**Days Taken**
Cutey Cute	5 days +2 days porting to QT
Fire	6 days + 1 day graphics
L-Sys Viz	7 days
Fuzzy Blocks	5 days
Pick-a-Path	9 days
Pac –Man Demo	4 days
Golli	4 days
Chicken crossing	4 days
Initial planning	7 days
Play Testing /Debugging	8 days
Unreleased Prototypes	6 days
Report and documentation	7 days

Chapter 5

Prototype 1: Cutey Cute

5.1 Introduction

Cutey Cute is a 2d Particle system visualizing tool that has been written to manipulate the particle system data to create interesting particle effects and to render out the particle as particle sprites which can be imported into the games as sprites. Cutey Cute provides features for manipulating the way the particles are emitted and with a completely customised function can also control the way the particle behaves . The explanation of the particle system and the way it works is explained in the forth coming sub sections.

5.2 Previous Work

Particle systems is a way of modelling fuzzy objects such as fire smoke and clouds . A particle system is based on three fundamental principles of a object being represented by primitives, the particles are dynamic and its shape and form are indeterminate. . William T Reeves (1983) in his paper on " A Technique for modelling a class of fuzzy objects " talks about the list of procedures that are necessary for developing a good particle system which are.

- Creation of new particles
- Assigning a set of attributes to the set of particles
- Deleting all the particles that exceeded its life time
- Translating and transforming the existing particles
- Rendering out the particles.

5.3 Particles

The particles are the basics of the particle system it defines how and where the individual particles are rendered. Every particle has its own list of property. The properties include its position, velocity, size, colour, life, its maximum age and the direction the particle will move. It also has methods for updating the particles position and painting the particles on the screen. This forms the basic data holder for the particles.

5.4 Global Particle data

Every particle in the simulation is initialized with its own set of values when it is created. In order to do this a global singleton class is created and referenced through out the course of the simulation. By

creating a global singleton class we provide access to the same data instead of referencing it through out the course of the simulation and accessing data of the references. This way we have the ability to access the same data through out the simulation.

5.5 QT – Open GL interaction

Qt has a inherent support for rendering out data using Opengl . For this purpose the QT Opengl header needs to be added for the opengl support. QT doesn't provide SDL support and all the user interactions need to be done through the QT mouse events and the QT key handling events. The linkage between QT and Opengl is done by means of signals and slot. A wide array of QT -UI elements has been used in Cutey cute. And the QT elements are linked with the proper slots in the main window class. A few of the QT elements have been explained. The Qt Opengl works by means of instancing a pointer to the Opengl application class and by using the slots to call the functions dereference from the pointer.

5.5.1 Qtimer

Qtimer calls a time out signal at the end of every timer event The Qtimer is started at the press of the simulate button and stopped when simulate off is pressed. The Qtimer helps control the time updates of the simulation. Any number of Qtimers can be initialized in the QT application.

5.5.2 QlineEdit

This is one of the most frequently used QT Ui elements. It basically helps the user to feed input to the program. The text in the line can be retrieved by means of the text () function which returns a Qstring of the text in the line. This QString is converted into a std::string or a integer or a float using the proper to functions. Similarly when the text in the line is changed the editingFinished() signal is called to link to the slot .

5.5.3 QComboBox

QcomboBox were used for drop down lists. This uses the signal currentIndexChanged() to pass the integer of the index of the current value in the Qcombobox which can be used for redoing to simulation.

5.5.4 QcheckBox

QcheckBox use the signal of toggled(bool) to pass a boolean to the slot to see if the check box has been toggled. The Boolean is passed to the window class to do the necessary changes to the simulation.

5.5.5 Redoing values in the interface.

Qt also provides functions for resetting the values in the interface by a feed back look. For example the value of some field can be changed during the course of the simulation using the proper functions, This allows us to keep a visual tab on what the current settings in the simulation are .For example ***ui->load_2->setEnabled(false);*** sets the ui element called load to be disabled.

5.6 The Emitter

The emitter class is the prime controller of the particle system . Emitter class works by a method of instancing pointers to the application every-time a particle is created in the emitter. For this we use a stl vector to hold the pointers to the particles and we add to the vector of pointers every time a new particle is created. Also when ever a particle is deleted from the system it is de registered from the particle array .

5.6.1 The Particle List

As mentioned earlier the particle list is STL vector of pointers to the particle class function. This in code is defined by the following

std::vector <Particles> ParticlesE;*

5.6.2 Adding to the particle list

Adding to the particle list is done every time the timer runs out and calls the add function. This retrieves the data from the data file and sets call to the register entity instance upon checking up exceeding the maximum count of the particles. In this function we check for the entry being registered earlier and if not then it adds to the new particle. When a new particle is created, the data which has been loaded from the QT is updated in the creation routine . Due to this we get particles created with the updated parameters sent from Qt. This makes the simulator much more dynamic to work with and more visually reachable and appeasing.

5.6.3 De registering the Particles

Emitter age update is done in the system by the tick event of the timer This then passes on to see if any particle has outlived their time .and if found to have outlived their time then it de-registers the particle . De registering from the array is a two step process because a vector or pointers is being dealt with here . For this we get the index of the vector to be deleted and then delete the i^{th} element before erasing it from the vector. So in code this de registering would look like this .

This process needs to be followed when the final clear up function is called as well where we loop through all the particles in the array and delete the pointer before erasing it from the array.

```
//first delete i th element from the vector
delete ParticlesE[i];
//then delete it from the vector
ParticlesE.erase(ParticlesE.begin() + i);
```

5.6.4 Rendering out Images

Rendering out images of the Opengl screen as a tiff image is done by the frame writer class which was taken and modified form Jon Macey's Graphics Library. For this a combination of Opengl and ImageMagick is used. ImageMagick is a software suite for creating editing and composing images. For rendering the images we use the gl ReadPixels to read the data to a array(Image Magick, 2009) . In code this is given by

glReadPixels(Xpos, Ypos, Width, Height, GL_RGB, GL_UNSIGNED_BYTE, image);

5.6.5 Textrues in QT Opengl

For rendering the textures we need to load in the images as a Qimage . For doing this we load the texture as a buffer and then use the convertToGLFormat function to convert a normal image into a QImage. Then the following process is followed to load a GLTexute for drawing

- Generate the texture using glGenTextures
- Bind the texture
- Texture Parameter .
- Map texture to a 2d tex image
- Then map the texture to the proper co-ordinates using glTexCoord2i
- Delete the texture

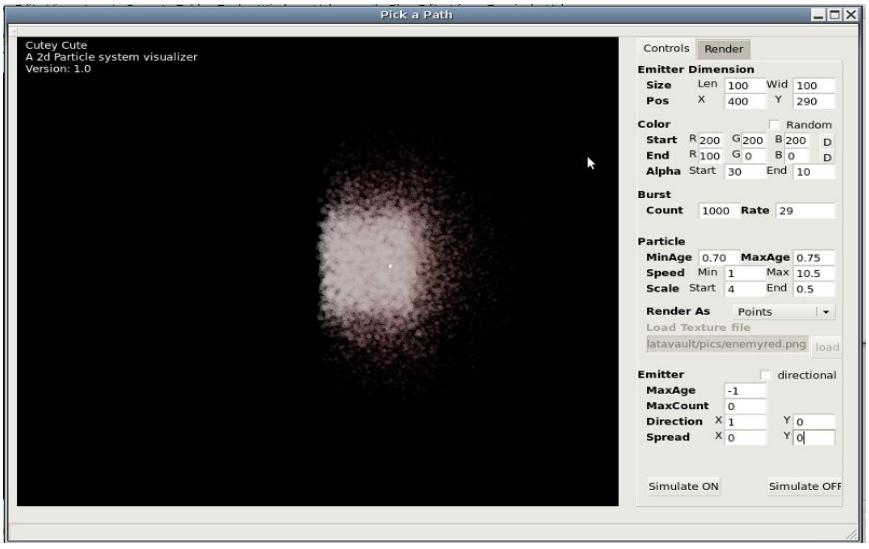

Fig 5 : The particle system editor in action

Chapter 6

Prototype 2: Pick-a-Path

6.1 Introduction

Pick-a-Path is a little AI tool which is used to make enemies move along paths in a game . It was done using Opengl for rendering and Qt for the interface. Pick-a-Path works on a concept of nodes and edges and the character always seeks the next node which is connected to the current node . A little Pac-Man game was codes using this tool to showcase how the tool worked.

6.2 Interface

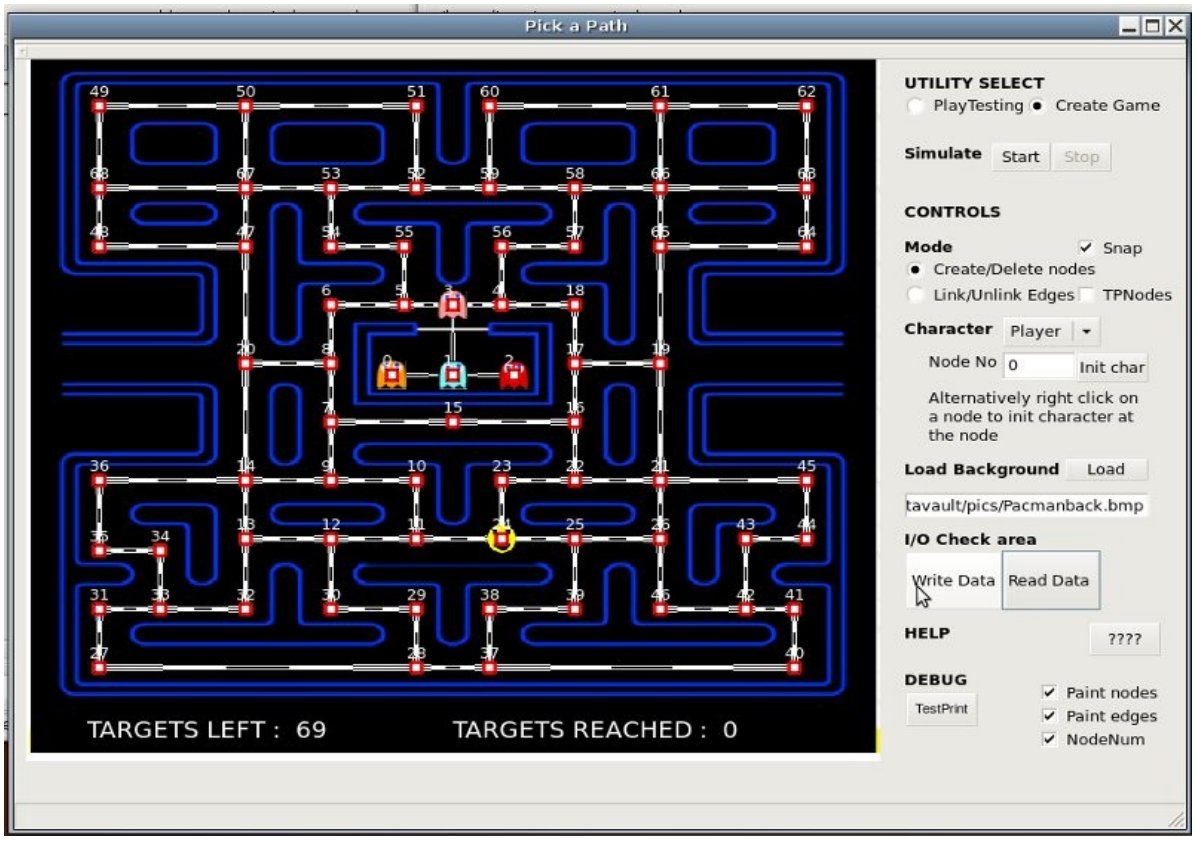

Fig 6: Demonstration of Pick-A-Path

Pick-a-Paths interface was done in Qt and was made to prototype as many features as possible within the time frame. Attempt was made to make it as user-friendly as possible. The features of the UI have been explained below.

6.3 Basic Definitions:

NODE:

The point where the character will make new decisions on which path to seek from that current position.

EDGE:

The path along which the character will travel from one node to another .

CHARACTER:

User controlled or AI controlled, it is an entity that moves between the nodes along the edges.

6.4 Features and Algorithms

6.4.1 Play Testing / Create Game

This is done using a Qradio button which passes the current state. Qt automatically groups the radio button in code if it is in grouped under the Ui editor. This sets or resets a boolean which is used to check whether the features of creating the game can be allowed or not.

6.5.2 Simulate

The simulation is run by means of a Qtimer which updates the screen every time the time-out event is encountered. The Simulate button controls the start and stop of the Qtimer on the simulation.

6.5.3 Creating/ Deleting Node

The creation of node uses the inherited mouse press event to gather the position of the mouse click on the screen. A pointer to the mouse event is passed to the mouse press function every time a click is made. The pos() function of the event is used to retrieve the current mouse click position. This position is checked for consistency in the screen .

Two modes exist. The first is the creation mode and the second is the deleting mode. Every time a valid click is made the program checks to see if there is a node existing at that point. If a node exists then the program deletes it . If there is no node existing within the range then it checks for the snapping algorithm to / whether it needs to snap it into position. Then a new node is created and is updated into the vector list.

So the algorithm can be briefly explained as follows:

1. Get current mouse position and check whether it is in screen
2. Get whether a node exists at the range of the mouse click
3. If a node exists then delete it

4. If not then check whether snapping needs to be done
5. Create node at the new point.
6. Add it to the node list

6.5.4 Linking/Unlinking Edges

The creating and deleting of edges follows a similar algorithm to the nodes . But the edges are a similar to edges in the graph data structure. Edge is vector of vector of node numbers denoted in code by

std::vector<std::vector<int> > Edgelist

Edges hold the index numbers of the nodes that are linked. The size of the edge list is the number of nodes that are available in the system. Each node is assigned a list of edges that are linked with it . The node numbers of the linked node are stored in the edge list and when the proper arrays are accessed the node number is returned. This node number is passed to the other functions for making decision. The process of linking and de-linking nodes can be explained by the following algorithm.

- Get the current mouse click position
- Check to see if the mouse click is made in a valid position
- If a node is selected check to see if the node is either the first node selected or the second is. If the first node is selected then start the linking.
- If the second node is selected then push back indexes of the second node into the position edge list of the first node.

6.5.5 Initializing the character

The character is initialized by right clicking the node.. Which character to be initialized can be chosen from the drop down menu available in the Ui. This works by identifying the nodes where the character will be initialized checking edge lists to be sure there are edges connected to the node and then initializing the character to the node.

6.5.6 Loading and reloading data from a file

Every time the data is stored on the file it writes out the data into 3 files. One is of type ndat which holds the node data in the proper format . The next is the .edat files which hold the edge data. And the last is the cdat holding the character data. O the input. Ostream and Istream are used to write out the data to the files and load the data back into the files.

6.6 Pac Man Demo App

A little pacman demo application was written using the pick – a – path tool to demonstrate the working of the tool . A little explaination of the pac man demo has been given here . The path that needs to be followed by the enemy and the characters are plotted in the pick-a-path tool. And the character and the enemy are initialized and data is stored.
The archives folder contains a test map created for the pac man demo.

6.6.1 Character

The character used in the pac-man application has inherent methods for updating bounds, loading textures and moving at the current velocity. The AI algorithm which was used for making the character move along the path is explained below.

6.6.2 Collision Detection

Collision detection algorithm works on the basis of returning from the loop in the shortest possible time . All collisions are made between bounding rectangles. The collision detection between rectangle 1 and rectangle 2 can be explained as follows.

- Check the X axis position of rectangle 1 with the X bounds position o f rectangle 2 . If the minimum x-bounds of rectangle 1 is greater than the maximum x bounds of rectangle 2 or if the maximum x bounds of rectangle 1 is less than the minimum x bounds of rect 2 .then there can be no collision.
- Do this test for the Y axis position as well
- If the function has'nt returned no collision then return collision has been found.

Fig 7: Criterion for detecting no collisions

The first step in the character and enemy ai algorithm is the collision between the enemy and the character. If a collision is detected then the character and the enemy are reinitialized to their original position a check on the lives left is made and the proper response followed. If this is not encountered then the program checks for a node collision. If a node collision is detected then it increments the visited count and checks to see end of game. Also upon reaching every node the character loops through the list of edges that are linked for the node and chooses one of the link and sets the end node and starts to move along the edge. This will make the character move along the edge till it encounters a node with no edges upon which it will stop and the character will not move. The character moving along the edge is done by directional vectors that connect the start and the final position of the node and velocity can be obtained from the directional vectors by multiplying the directional vectors by their speed. The character follows the same ai movements as the enemy but upon reaching every node it checks for user input to determine in which direction the character will move from that position.

Chapter 7

Prototype 3: L-Sys Viz

7.1 Introduction

L-Systems is a method of generation of patterns using a formal grammar and recursively substituting the variables by other variables and constant. Discovered my Lindermeyar , L Systems is one of the major tools in the graphics industry. In games L_Systems is highly used in the production of road networks , sprites and rendering o trees plants . L-SysViz is a 2d L-system rule creation and visualization software that can aid in the creation of L-System rules and visualizing the output of the rules. This stored information can be used to remodel L-Systems in games

7.2 L-System Basics

L-Systems work on the basics of a formal grammar and rules. The L-System grammar is usually a tuple of four components.

- Variables are the symbols which can be replaced during the recursive replacement process of generating a end rule.
- Constant are the symbols which are substituted as is during the replacement process.
- Axiom is the string of symbols that is the starting rule
- Production rule defines which variable will be substituted by what during the course of the replacement.
- Generations defines the number of iterations that the loop will run to substitute the variables using the production rule. All the variables have a predefined meaning .which will be translated to give the proper output in the render.

7.3 L-Sys Viz

L-Sys Viz works in three distinct layers. They are the File parser, the expansion on rules and the application of proper grammar to the rules . Any changes to the Qt UI system are redone in the L-System text data file that holds the information. Every time the Re-Bake method is called it sets call to the parser method which reloads the data from the file with the newly updated data.

7.4 L-Sys Viz User interface

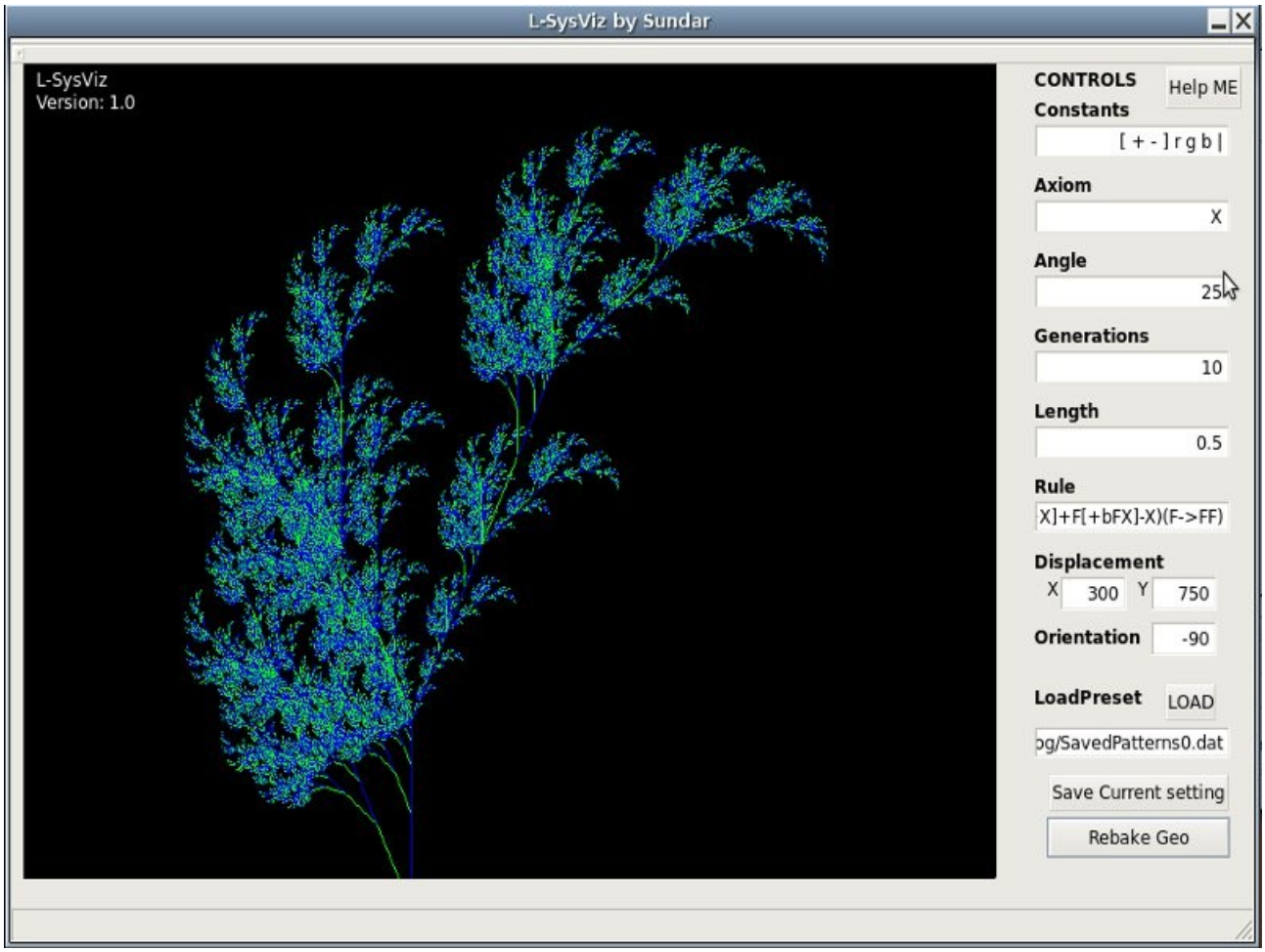

Fig 8: L-Systems User interface

7.5 L-Sys Viz Algorithm explained.

The L-Sys Viz has three fundamental algorithms which are explained here

Parser:

L-Sys Viz fundamentally works on the basis of querying key words using the parser. The parser is sent a request to retrieve the value associated with the key word. The parser follows the following algorithm to retrieve the key values from the data file .

- The key whose key value needs to be retrieved is passed into the parsers Query key function.
- We loop from the start to end of the file until we find the key value. If we do not find the key value then we return a false.
- We check for the comments in the file and if comments are found then the execution is not continued.

- Here we use the find_first_not _of function of the string class to find the index of the first character without the delimiters then .We then take the difference in the get the sub string to retrieve a array of string.
- These arrays of strings are processed to get values as integers or floats.

Expand Axiom

Expanding the axiom works on the principle of recursively transferring the data from one vector to another and while transferring, reading the production rule to expand the axiom into the rightful size. This process is repeated until the number of generation of the L-System. To do this a character array is created and the existing axiom is read and transferred into the character array. Every time a character is read it is translated to find out whether it is a variable or a constant. If it is a constant is substituted as such. Or else it is compared with the production rule to check what the expanded rule is and pushes it back.

Applying Grammar

In this step we read each character from the rule set and translate the meaning of each character into point position. Each variable in the L-System has a meaning associated with it . For example F draws a line forward, '+' rotates the L-System by the given degrees in the clock wise direction. , - rotates it in the anticlockwise direction and so on. The current position of the branch is stored and depending on the definitions of rules new points are calculated.

In case of branching we create a new array to store the position of branches and keep adding to the array as we encounter more branch starts in the rule list. As a branch end is encountered a pop out is called to retrieve the last loaded position and this is initialized to the current positions and the calculations are done using this position as a base.

Chapter 8

Prototype 4:Fuzzy Blocks

8.1 Game Play

Fuzzy blocks are a recreation of the Mind Jolt game "Tower Blocks" where the aim is to build a tower as high as possible. When a mouse click is made the block falls down from the crane. A life is lost when a block misses the last block on pile. When three lives are lost then the game gets over. Incremental bonus is added every time a block is placed exactly on top of each other. As more blocks are placed on top of each other the tower becomes less stable and begins to shake. The higher the tower gets built the more widely it starts to shake and placing of blocks on top of each other becomes progressively more difficult.

8.2 Progress of the project

The aim of the project was to implement as many features of the game design as possible in the prototype in the given period of time.

SDL_Timer

SDL_Timer is added to the simulation as and when required. SDL times work on a principle of call-back functions. Call back functions are functions which are called every time a event has occurred. So every time the specified time has run out SDL_Timer sets call to call the call back function.

```
SDL_Timer* Displaytimer;
Displaytimer=SDL_AddTimer(40,GameLoopTimer,this);
```

In the above example the Displaytimer is a SDL_Timer which is added to the simulation. The GameLoop is a call-back function that is called every 40 milli seconds. But the call back functions are static member functions which execute a loop and returns the interval which is the time elapsed between the last tick and this tick. If the time elapsed is greater or lesser a new timer is initialized.

Static member functions had one problem in executing the loop. Unless an instance of the variable has been already declared it cannot be used in the member function. This means no call to the member functions or variables of the class to which the static member function belongs to can be made from within the function.

Box Movement Algorithm

All the boxes have two distinct types of motion. One is when the box is on the crane when it follows rotational movement. The other is when the mouse click is made and the object experiences free fall motion. Every time a collision is detected between the freely falling box and the box pile then the freely falling box is reinitialized to it original release position and a new box is created and appended to the box pile at the position of collision. The box pile has two types of motions that it follows. One is the horizontal sway and the other is the vertical scroll. The horizontal sway determines how much the box will sway at any point of time from its mean position due to the increase in the tower height. Scroll is done when a new box is added to the screen and the bottom box needs to be removed from the system. The steps in the algorithm for the complete box movement are described.

- Check to see if the box pile needs to scroll. If it needs to scroll then scroll down and delete the box that has scrolled under the screen
- Check to see if the box needs to sway. If it needs to sway then add displacement in the X direction and let the box pile sway.
- Check to see if the box is either in the suspended move mode or the free move mode. If former then we don't detect collisions
- If in the free fall mode then update the box position.
- Check for collisions with the top box of the box pile . If collision is found then increase score and check for scroll. Add to the existing box pile.
- If the box has reached the bottom then delete the box .

Fig 9: Snap Shot of the Fuzzy blocks game

Chapter 9

Prototype 5: Golli

9.1 Game Play

Golli is a board game which was scripted using C++ / Opengl and the controls were made using SDL. Golli's board is made up of 36 holes. Initially 35 of the 36 holes are filled with balls. The centre hole of the board is unfilled. The game rule is that every move you need to move one ball to a hole, 2 holes from its present hole and follow these two conditions:

- It needs to hop into a empty hole
- It needs to hop over a ball into the empty hole]
- When one ball hops over other the the centre ball gets deleted. The aim of the game is to delete all the balls until one last ball remains.

9.2 Prototyping the game

The actual game board contains a list of 9x9 holes and we set the ones not needed in the game to be not holes These are not rendered or used in the game.

Golli also uses the SDL_Timer to set up the game timing. The main game loop begins and runs during the mouse click events. The algorithm tat was used has been briefly explained below.

- Get the mouse position as X Y co-ordinates .For this we use the SDL_GetMouseState function to retrieve the XY co – ordinates.
- We then check up these XY co ordinates with the Game boundaries to make sure they don't overlap or extend.
- Every hole is denoted in boundaries by a maximum and minimum vector holding the x y co-ordinate of the positions.
- When a click is made the column in which the mouse click is made is found out by checking in which bounds the y position of the mouse click lies.
- Once this is found the x position of the click is calculated to find the index of the box where the click is made.
- Each time a ball is selected we check to find out whether it is the first ball or the second. And based on it the static indexes are set. Then these indexes are compared to find out whether they are 2 indices apart
- We go on to check if the centre hole has a ball in it and the end hole is empty then swap the balls in the hole with the final hole

Chapter 10

Prototype 6: Fire

10.1 Game Play

Fire is a game made out of the particle systems. The focus of the game is the burn down all the objects on screen before the timer runs out. The player controls a burning flame which is a particle system emitter emitting out particles continuously. We need to fill the yellow objects with particles. We call these objects buckets. Every bucket has a capacity of the number of particles it can hold at any point of time before which it gets destroyed. Every level demands the player to destroy as, many buckets to end the game. Enemies move around in the scene and constantly seek the player. The number of enemies is dependent on the level in which the player currently is in . The higher the level the more the number of enemies present in the system. If a enemy manages to reach you get frozen and lose time after which the system is reset and you get to go again. The prototype consists of 10 levels with increasing difficulty.

9.2 Sounds in SDL

Sounds were tried as a part of this prototype. Sounds in SL work need SDL mixer to work . Audio in SDL needs a few basic parameters like frequency (usually set to 22050 for games) , format , and channels (2 for stereo and 1 for mono) . The audio subsystem needs to be initialized for playing music. LoadMus() method is called to load the music as a file in a recognizable format and the Mix_Playmusic needs to be called every time the music needs to play. We can set the number of times the music loops while defining the mix_Playmusic.. Mix_HaltMusic() is called to stop playing the music and the audio sub system needs to be freed and closed everytime. By using a combination of all this we get music working in the system .

9.3 AI

The enemy AI in the system is basic in nature. A seeking algorithm makes sure the enemy constantly seeks the character position and redirects its velocity to that position there by constantly moving towards the character. When the enemies are initialized each enemy is assigned varying speeds all less than that of the character to make the game interesting.

9.4 Collision detection

Every particle that is alive in the system is tested for being within the bucket which is alive . If the particle found to be inside the bucket then the fill percentage of the bucket is increased and then checked to

see if the bucket is alive. If the bucket has reached its maximum fill the bucket is killed.

9.5 Game Algorithm

- Update the emitter age using the current time
- If the emitter is not deactivate then add particles , clear up the dead particles and update the state of the particles.
- Calculate enemy movement towards the character
- Check for collision detection with the enemy and if detected reduce the life of the character and checks for the game getting over .Also reset the position of the character and the enemies.
- Else update the characters position and check to see if it is in bounds.

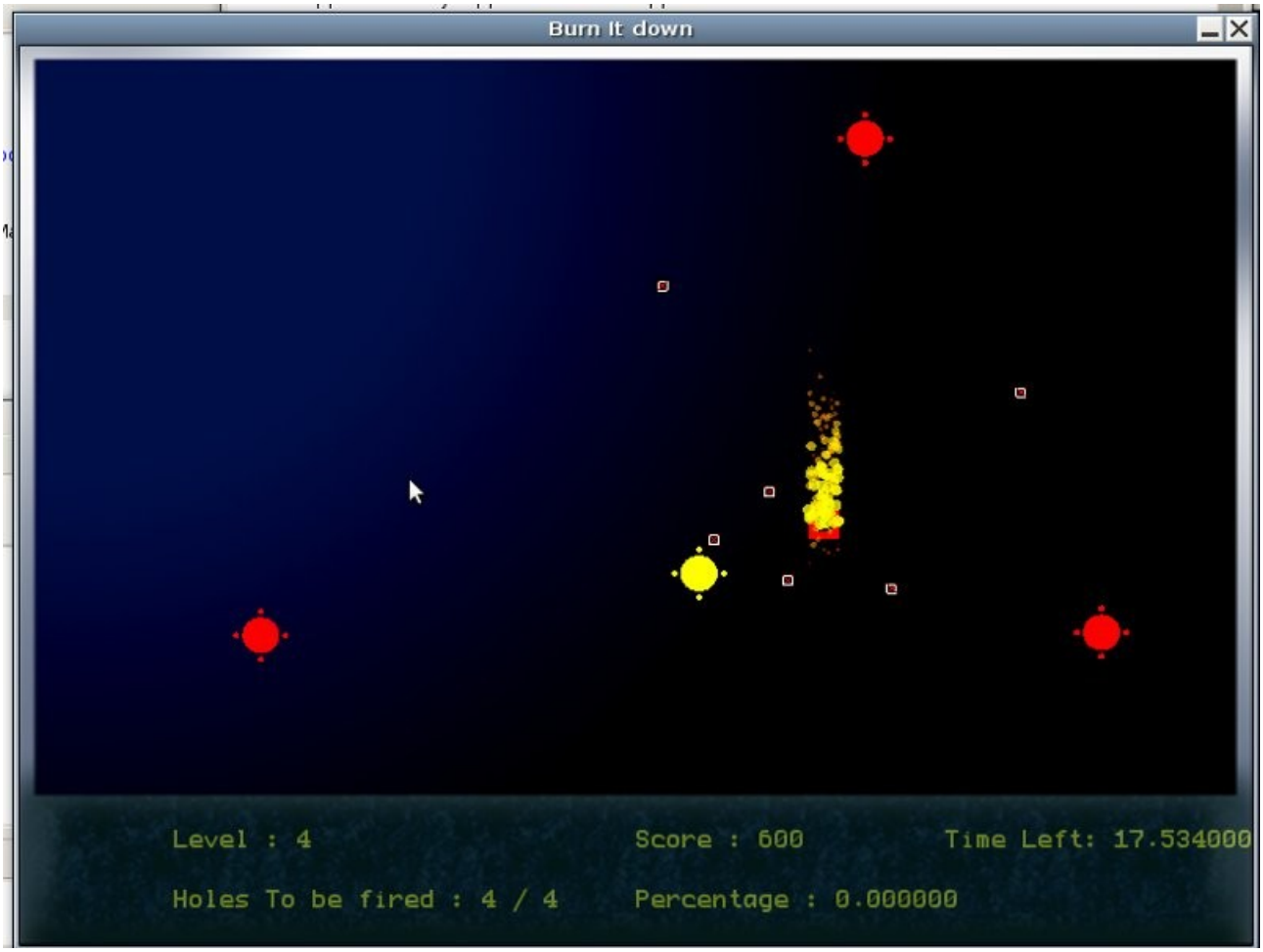

Fig 10 : Snapshot of Fire

Chapter 10

Prototype 7: Chicken Crossing

7.1 Game Play

Chicken crossing is a little fun game which is basically a Opengl version of a flash game . The player is in control of a chicken and the aim of the game is to get it safely to the other end of the road without being over run by the vehicles. The prototype has 6 tracks each having a

7.2 Timer

This game uses SDL_Timer and call back loops explained above. Three timers are used in the game. One for displaying the screens, one for the vehicle updates, the next is for displaying the UI and the graphical elements the last is a auxiliary timer used as and when required.

7.3 The Game Algorithm

- The algorithm works on the basis of filtering out the vehicles and testing collisions with the nearest vehicles To do this every time collision detection is made the fist step in the collision detection would be to test the character to find which track it is in.
- By doing this we find out which track the character is in. A provision has been provided to check for collisions in two tracks as well.
- Then the algorithm loops through the vehicles in the track and checks for collisions between the character and the vehicles. If collision is found the character is reset back to the start position and the life is reduced
- The creation of the vehicles and placement are procedurally controlled . Every time a vehicle goes out of scope the vehicle is deleted and a new vehicle created
- Finally the player position is updated.

7.4 Vehicle creation and movement

During initializing the game we retrieve the data from the global file and check to see how many vehicles can be created on a single track . Also check to see which direction the vehicle moves . For the proper setting of the vehicle we create two vehicles more than the number that can be accommodated in the track. 2 is not arbitrary in nature. Every time the number of vehicles is returned we get a integer number . This integer number is the ceiled value of the float obtained by dividing the length of the track by the sum of

the length of the vehicles in the track and the distance between the vehicles in the track . Often there are cases where this value doesn't exactly fit into the track . So in order to satisfy these dependencies we add one more vehicle to the track list. But due to the way the creation algorithm works a vehicle is created everytime a vehicle is destroyed. And a vehicle is destroyed every time it goes out of bounds. If count + 1 number of vehicles are alone present the algorithm threw up errors because overlapping of vehicles occurred and also vehicles were created very sooner than they normally should because the vehicles get destroyed. So another extra vehicle was created to satisfy this dependency. So initially the creation position of the entities is set two blocks behind the screen so that smooth translation between the creation and deletion will occur.

Every time the update function is called the vehicles position is updated using the velocity and the bounds are recalculated. This then undergoes a check to find whether the vehicle has outlived its bounds . And if it has outlived it then this vehicle is removed from the system , Also a new vehicle is added to the system at the init position which has been calculated during initializing the game.

Collision detection between the character and the vehicles is made so that unnecessary tests are avoided. Every frame the current track of the character is found by checking the characters position against the track bounds . Since a attempt was made to keep the game as procedural as possible the character is tested for each bounds individually and the collision test returns where each of the characters bounds are . If the character is found to be in two different tracks at the same time then collision tests are made on both tracks. If both the bounds of the character are in the same track then that track is considered for collision. Only vehicles in that particular track are taken for collision detection testing and this reduces the number of collision detection tests by up to $1/6^{th}$ in the game .

Chapter 11

Bugs and Fixes

This chapter explains few of the common and interesting bugs that were faced during the course of the project.

Bug 1 Occurrence of white dots in images in Cutey Cute
Status Pending

As the simulation progresses there is occurrences of white dots occuring as a part of the simulation which gets magnified leading to blank frames in the simulation .

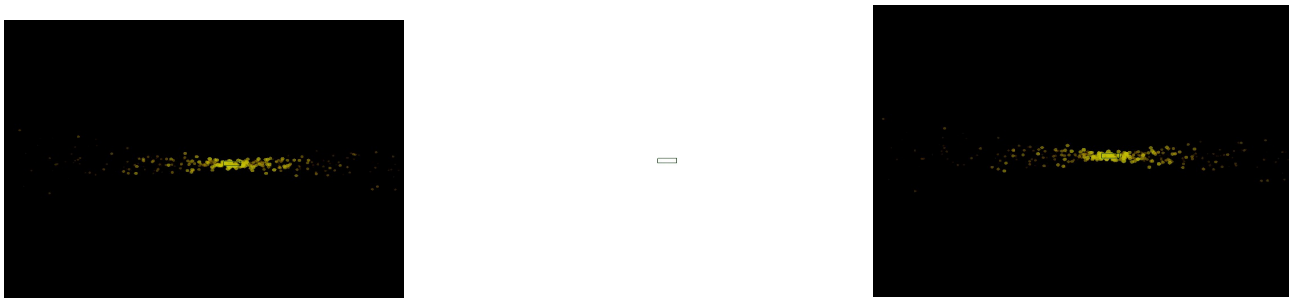

Fig 11 : Blank frame rendering

Bug 2 Memory Leaks in Cutey Cute
Status : Cleared

Since the pointers to the particles were used and the Particle array was basically a array of pointers to the particle improper clearing of the pointers. Since we were using a vector of pointers the steps for proper cleaning of the vector was to delete the pointer first then erase the vector which avoided segmentation faults.

Bug 3: Pushing in values into a 2d array in L-SysViz
Status : Cleared

L-Sys Viz required values being pushed in for every node as soon as it it created . Let us say we have have a 2D array of type int a.

vector<vector<int > > a;

We have n nodes and we want data of n node in this array .Then a[2].push_back() would throw up errors because initially the array doesn't exist. So every time a node is created a empty array was pushed in and the empty array stocked. On a better note a vector was defined locally and filled in with integers and the contents were pushed into the 2d array.

Bug 4: Size mismatch between Qt Window and Opengl window
Status : Cleared

The Qt Mouse click retrieves the position of the mouse in the Open Gl window . But it takes into account the size of the OpengGL window speified in code and not the GLFrame of the Qt application. Due to size limitation the size of the GLFrame in the UI window was reduced but the proper update wasn't made in code , due to which many discrepancies arose in the displaying and selection process.

Bug 5 : Deletion of Nodes in Pick-a-Path
Status : Pending

During the deletion of nodes there is a error in the way multiple linked nodes are deleted. When a singly linked node is deleted all links are detached and the nodes are detached but when multiple nodes get linked the path gets skewed up.

Bug 6 : Negative values assigned to unsigned long int.
Status : Cleared

In Fire the elapsed time is calculated from the difference of the current sim time and the time emitter start time. But due to the emitter being reset at the start of the simulation there were negative values being fed to the variable. But since it cannot accept negative values the value was reset to 4,294,967,295 which is the highest value of the unsigned long int. This caused the game to behave with unexpected results

Bug 7: Loading Opengl Textures

Status:Cleared

During the creation of textures the textures were initially made as BMP files which were saved at 24 bits per pixel. (i.e) has the R G B components and problem with this was no alpha could be rendered out using these images. So the images needed to be rendered out as png images with inbuilt alpha in it to work properly.

Also in code there were two major changes that were required to fix this change from a 24 bpp image to a 32 bpp image. Initially the SDL needs to be set up so as to accept a 32 bit image . For doing this we use the SDL_GL_SetAttribute call to allocate bits for individual colour

```
SDL_GL_SetAttribute (SDL_GL_RED_SIZE,8)
SDL_GL_SetAttribute (SDL_GL_GREEN_SIZE,8)
SDL_GL_SetAttribute (SDL_GL_BLUE_SIZE,8)
SDL_GL_SetAttribute (SDL_GL_ALPHA_SIZE,8)
```

While loading the textures proper changes needs to be done to glTexImage2D. In the parameters passed to this function, we need to set the internal format to be a 4 color one. And also the type to receive RGBA mode failing which it might lead to problems as shown in the image below.

Fig 12 Background Image error due to the mismatch of BPP

CHAPTER 12

CONCLUSION

12.1 FUTURE IMPROVEMENTS

Rapidly prototyping games are an interesting way of learning to do small things very quickly thereby aiding in the process of producing longer versions. A number of improvements has been suggested to the prototypes that has been created a few of which has been mentioned below.

The particle system editor can be improved to create 3d particle systems that can be rendered out as rib files. These rib files can be rendered out as primitives to create more dynamic particle systems. The feature for customization of code could be built into a code editor that can be compiled and in run time giving more flexibility to the particle system editor. Factors such as wind and other factors which can affect the particles can be added. Also collision detections can be implemented.

The pick-a-path is a basic demo version of the tool. Future developments in the tool could include a splines joining between the nodes rather than straight lines, options to orient the character along the line of movement , export it to a 2d map and try to actually implement the tool in a 3d game and also provide provisions for a larger map.

Suggestion of improving the L-Systems to 3d were also provided . Audio addition to the games and improving the graphical look of the game were the two main additions which were recommended for the games

12.2 EVALUVATION

The project started off with the aim of prototyping small versions of the game and applications in approximately 1 week's time. What was aimed at is 5-10 games and applications through the course of the project. The project was successful considering the minimum target was reached in the time frame.

Though the project did'nt aim at high quality graphics, there were glitches in acquiring good quality audio and assets for the project. A few new ideas have been generated as a part of the project and the future development section deals with a few interesting additions to the prototypes to make it full fledged applications.

A few of the algorithms work really very well like the character movement in the Pick-a-path tool though there is a great scope for improvement in the near future to build on the project and prototypes to make it more usable.

References:

1)Terdiman ,D. , 2009, Video game industry posts big February gains, Available from:http://news.cnet.com/8301-10797_3-10200720-235.html [Accessed 19 August 2009]

2)Anon, 2009,Software prototyping ,Available from : http://en.wikipedia.org/wiki/Software_prototyping [Accessed 19 August 2009]

3) Blomquis.A., Gabler K., Gray K.,Purho .P., Shodhan .S .,[no date] About .Available from http://experimentalgameplay.com/blog/about/ [Accessed 19 August 2009]

4) Gabler.K., Gray ., K ., Kucic.M., Shodhan .S., 26 October 2005. How to Prototype a Game in under 7 days : Tips and Tricks from 4 Grad Students who made over 50 Games in 1 semester Available from http://www.gamasutra.com/features/20051026/gabler_01.shtml [Accessed 19 August 2009]

5)Jana.R., 8 November 2005., The Many Ways to Make a Game. Available from http://www.businessweek.com/innovate/content/nov2005/id20051107_181961.htm [Accessed 8 November 2005]

6)Petri., 18.August .2009., Articles About Rapid Game Prototyping. Available from http://www.kloonigames.com/blog/general/articles-about-rapid-game-prototyping [Accessed 19 August 2009]

7) Shodhan.S.,[no date] Portfolio of Programming Projects . Available from http://www.shalinshodhan.com/projects.htm [Accessed 19 August 2009]

8)Owen.S., 8 . February 2000., Particle Systems . Available from http://www.siggraph.org/education/materials/HyperGraph/animation/particle.htm [Accessed 19 August 2009]

9)Morefield.R and Malloy.B.,[no date] 3D Game Development Tutorials., Available from http://www.cs.clemson.edu/~malloy/courses/3dgames-2007/tutor/index.html [Accessed 19 August 2009]

10)Wikipedia., [no date] OpenGl., Available from www.en.wikipedia.org/wiki/Opengl [Accessed 19 August 2009]

11)OpenGl., OpenGL API Documentation Overview ., Available from http://www.opengl.org/documentation/ [Accessed 19 August 2009]

12) SDL., 17-August-2008., The Simple Directmedia Layer Documentation., Available from http://www.libsdl.org/cgi/docwiki.cgi/ [Accessed 19 August 2009]

13) TrollTech., c.a 2008., Online Reference Documentation., Available from http://doc.trolltech.com/ [Accessed 19 August 2009]

14)ImageMagick., 2009., Introduction to Image magick., Available from http://www.imagemagick.org/script/index.php [Accessed 19 August 2009]

14) Macey.J ., 21- April-2009 ., Graphics Library [Computer graphics Library] Bourenmouth University

15) Games in a Flash., c.a 2009., Games : Why did the Chicken Cross the Road [Computer flash game] Eyeland Studio

16) AArondbaron .,12-April-2005, Sample::messed.mp3[Sound Sample] Available from http://www.freesound.org/samplesViewSingle.php?id=575 [Accessed 19 August 2009]

17)Watt.A., Policarpo.F ., 2003,3D Games Animation and Advanced Real time rendering., Essex :Pearson

18)Woo.M., Neider. J.,Davis .T and Shreiner. D., 1999.,Opengl Programming guide., 3rd ed .,Massachusetts : Addison Wesley