I WANNA BE A SHOOT ‘EM UP

Alex McAllister, Eoghain McGrath, Luke O'Brien, Nathan Dunne

Title of Project	I Wanna Be A Shoot 'Em Up
Student Name & Number:	Alex McAllister - K00200042
Student Name & Number:	Eoghain McGrath - K00204708
Student Name & Number:	Luke O'Brien - P11011180
Student Name & Number:	Nathan Dunne - K00211819
Date	8th February 2018
Word Count	17005

Games Design and Development – BSc (Honours) in Computing (Level 8)

Limerick Institute of Technology (Tipperary)

Certificate of authorship

We hereby certify that this material, which we now submit for assessment on the programme of study leading to the award of [degree title] is entirely our own work, and has not been taken for the work of others save and to the extent that such work has been cited and acknowledged within the text.

Name of Candidate: Alex McAllister

Signature of Candidate: _____

Name of Candidate: Eoghain McGrath

Signature of Candidate: _____

Name of Candidate: Luke O'Brien

Signature of Candidate: _____

Name of Candidate: Nathan Dunne

Signature of Candidate: _____

Dated:

Table of contents

Acknowledgements

First and foremost, we'd like to acknowledge our project supervisors; Liam Noonan and Eugene Kenny. Liam has taught us a lot about game design in the past two years, as well as teaching us a large amount of what we know about C++ and SDL. Both our project supervisors were very supportive of our project ideas, as they knew and understood what we wanted to create from the beginning. They both showed a lot of interest in the AI aspects of our project, as well as the procedural generation. We feel had we had less understanding lecturers, our project may have been pushed in a different direction, to something we didn't want, and we wouldn't have been able to stick so closely to what we had envisioned.

A large section of our knowledge on this project was taken from the book, Beginning C++ through Game Programming by Micheal Dawson; It helped give us a comprehensive overview of the C++ language, as well as further cement in our minds the ideas we were learning in class. Without going through this book and learning the core principles of how the language works, we feel our ability to work on the game would have been severely inhibited.

The second book that helped us complete this project is SDL Game Development by Shaun Mitchell. This book helped immensely with understanding the SDL library, as well the fact that our games code base was from this book, in the form of Alien Attack. Without Alien Attack, and the SDL library functions, we would not have been able to make the we did in such a short period of time.

For inspiration on the game mechanics and feel, we took a lot from the game The Binding of Isaac by Edmund McMillen, which is also a top-down, rogue-like game. We were also inspired by a game called Nuclear Throne by Vlambeer, also a rogue-like, but a lot more bombastic. Both the games have helped us a lot with design choices and ideas, as they're games we consider to be epitomes of the genre.

Abstract

I Wanna Be A Shoot 'Em Up is a top-down, procedurally generated, dynamically adjusting, shooter game, with rogue-like elements. You take on the role of a nameless protagonist controlling a robot, as you attempt to purge worlds of grotesque monsters and return order to the multiverse.

The game is written in C++ using the code base for Alien Attack, and the SDL library for audio, input/output, and graphic rendering.

This report is going to examine in great detail how our third-year project was conceptualised, built, and finished. The report will cover our design choices, in terms of gameplay, as well as the design methodologies we used. It will cover the literature we consumed, and the research we did. We will talk about the system design and architecture of our project, and the testing that went into it. Finally, we'll take a critical look at how well our project did compared to our original goals.

1 Introduction

In the following documentation, we will be discussing every element of our project, from conceptualisation to completion. We'll be illustrating the project from paper to our upload. The documentation will include our original design ideas, and how we managed working as a team. Then we'll talk about the literature we reviewed, and the research that went into making our game. Then we'll take a look at the system design and architecture, how we implemented collision and dynamic difficulty, as well as how the object orientated aspects work together. From there, we will look at Testing and Implementation. Following on we will take an in-depth look at how we performed on our project in the Critical Analysis section. Finally, we will finish off this document with our personal reflections.

One thing that's very unique about this project for us as a team was the fact that we started off with the large code base of Alien Attack. This initially meant an awful lot of learning, as we figured out how the game worked, and how SDL and C++ melded together. This has long term benefits to our understanding of program modification and creation.

We are using C++ in college because it is the leading language in the Games Industry. It is very powerful and efficient which makes it perfect for video games which are usually intensive to run. C++ is also very flexible and can be used with many methodologies and programming patterns in mind.

SDL stands for Simple DirectMedia Layer. We use it to abstract a lot of the computer graphics side of things. It is also used for Input Handling. By using this in tandem with C++, we feel like we are using the right tools to make the game that we want to make.

2 Problem Definition and Background

The goal of this project was to create a shoot 'em up for Gamesfleadh, Irelands digital programming festival, and for our third-year FYP. While doing the project we were to cement our understanding of the C++ language, the SDL library, and how they work in conjunction with each other. We also were required to work well as part of a team, resolve any conflicts,

and schedule everyone's time effectively so that we could create what we had envisioned. To effectively do that, we made use of the agile methodology.

Project Scope

Our game at the start was very ambitious, as we wanted to justify having an extra team member compared to everyone else this year, who had three members. We also aimed high, because as a group we enjoy being challenged, thinking outside the box, and creating something polished and unique.

We feel our project scope was justified, as we've all worked well in the past to produce a game, and we know what we're capable of.

After discussing the scope of this project, we have decided that we will have 15 minutes of balanced gameplay, solid controls and to have effective dynamic difficulty implemented.

We decided that including multiPlayer or online play was unachievable within the scope of this project.

Project Objectives

For our project objectives, we wanted to create something that could be played in around ten minutes, and be very balanced, with polished controls, good procedural generation, dynamic difficulty being well implemented, and have a lot of inimitable gameplay that would be enjoyable and fresh with every playthrough.

We had many objectives at the start, these included:

- **Balanced Procedural generation:** We wanted there to be balanced procedural generation, that allows for unique and interesting playthroughs each time the Player plays.
- **Dynamic difficulty:** We wanted to add the feature of dynamic difficulty to the rogue-like genre, as the genre is rather difficult and inaccessible to the average gamer.

- **Multiple characters, bosses, items:** We intended to have a large variety of enemies, bosses and items so that the Player can spend a lot of time learning the way the game feels, as well as having unique rooms/levels all the time.
- **Statistics:** We wanted to have statistics for the Player to view, as well as modify in combination with the items they pick up, this again would lend to the game feeling fresh each time the Player plays. We also wanted to have a leader board that the Player could view and compare their times with other Players.
- **Permanent death:** We wanted a perma-death feature in our game, to give the Player that sense of dread and adrenaline that comes with only having one shot at beating the game before they must start all the way at the beginning again.
- **Good memory management:** We knew from looking at previous 3rd year projects, that memory management would be an issue, so from the get go, we wanted to have a heavy focus on the memory management. It was also very important for a rogue-like game as new levels with lots of different enemies and items are created, and hopefully deleted, all the time.
- **Polished core gameplay:** We wanted the game to feel very polished and semi-professional. We don't want to release a game that doesn't feel good to play, and as a result, one of core objectives was polish.
- **Clear and concise user interface:** We wanted to have a clear and concise user interface, one that was unobtrusive to the Player, and instead let them focus all their attention on the game.
- **Original or open source assets:** As a team we wanted to separate our game from the game Alien Attack as much as possible. In order to do that, we knew at the beginning that we'd attempt to make all our own assets, sound effects, and music.

Feature list:

For our project, we had several core features that we needed to implement for the game to be complete. We first broke these features down into a user story map. A user story map is a subsection of agile development and helps break down core features of your product in mannerisms that aid the user experience. See figure 1 below for a section of our user story map.

LEVELS	ENEMIES	AI	ITEMS	PLAYABLE CHARACTERS
Room map layout	Interaction Type	Pathfinding / Movement	Active	Gameplay Type
Tile generation	Appearance / Visual Clarity	Aiming	Passive	Difficulty
Doors	Design	Dynamic Difficulty Influence	Availability	Risk vs Reward
Theme	Bosses	NPC vs NPC*?	Gameplay Influence	
Room Design	*Attributes*	States		
Start room	Health	*Disposition*		
Boss room	Move Speed	Aggressive		
Shop room	Damage	Passive		
Secret room*	Fire rate			
	AI Patterns			

Figure 1: User Story Map

As you can see from figure 1, we've broken down the key features of our game into manageable sections that present what the Player will experience in the game. The names in yellow are the high-level features, then it's broken down into what there will be in that feature, and finally, in the box, we have sub categories that deal with the coding elements.

The features we had in our user story map included:

- Levels: This talked about how the levels would be presented, and what the Player would be able to interact with.
- Enemies: In enemies we talked about how they would interact with the Player, their design, and what statistics would be involved in their creation.
- Artificial Intelligence: In AI we talked about the different kinds of enemies, the states their behaviours would be in, as well as how we'd implement movement, and attacking.
- Items: In items we talked about the effects the items would have, if they were passive items that would make a permanent change to the Player, or if they were active items that the Player would be able to drop or pick up, and use.
- Playable Characters: For Player characters we discussed the type of gameplay they'd have, for example a character could be very fast but take a lot of damage or be slow and tanky. We also were concerned about how the Player would choose from these, and how the risk vs reward would be balanced.

- Collision: In collision we needed to account for projectiles, the Player collision vs the enemy collision, objects that you could interact with and destructible environments.
- Dynamic Difficulty: Here we talked about the rate at which the difficulty would change, how lenient the game could get, and how difficult. As well as how data would be stored and accessed.
- Controls: In controls we mention having both keyboard and game controller support, the movement, shooting and mechanic activation, such as how you would activate a active item.
- Interfaces: For interfaces we discussed the different menus we'd like to have, as well as settings that were modifiable, statistics that would be shown to the Player, a character select screen, and the user interface itself.
- Gamestates: For game states we talked about the various states we'd include, such as the menu, the pause state, the play state, room to room transition and loading.
- Inventory: For inventory we mention how they would be stored and displayed.
- Player Direction: In Player direction we discussed having a tutorial, having haptic feedback for the controller, and giving the Player feedback based on what they were doing.
- Art: In art we listed all the art sections that would need to be in our game, these included the background art, the character/characters art, enemy art, UI art, projectiles, various menu art and particle effects.
- Sound: In sound we listed music, sound effects and audio cues for Player feedback.
- Story: In story we discussed having text in-between levels, at the start of the game, and at the end to give a basic story to the Player.
- Database: For the database we listed in-game statistics, level generation, leader boards, and separating the data being fed into the database from the data already in the XML files.

We feel having a user story map at the beginning that was fleshed out to such a large degree allowed for us to implement exactly what we needed and focus on making the game we wanted.

Due to using the agile methodology, we needed to include a way to organize what work we would be doing and when we'd be doing it. To deal with this, we used a specialised subsect

of the agile method named Scrum; Where in each development cycle is called a sprint. We broke out sprints down into sections that spanned a week, as we had meetings each week with our supervisors, where we'd show off what we'd added. An example of one of our first sprint sheets can be seen below in figure 2.

Key	Number:	Week: 1 - October 25th Basic Functionality	Week: 2 - November 1st Level Focused	Week: 3 - November 8th Player vs Enemy Focus
Alex	1	~~**Refactor Alien Attack, possibly some Art:** Configure health display on screen.~~	**Reuse Code Base:** Create more rooms, hardcoded, possibly. Explore expanding screen size.	**Reuse Code Base / Enemies:** Basic single good enemy.
Nathan	2	~~**Refactor Alien Attack:** Scrolling to a static screen.~~	~~**Collision:** Non-damage wall collision.~~	**Reuse Code Base:** ~~Refactor Controller Library~~, Player Attributes.
Eoghain	3	~~**Refactor Alien Attack:** Basic shooting and movement mechanics.~~	~~**Gamestates:** Explore Gamestate transition between rooms / basic door transition.~~	**Items:** An extra item that the player can use.
Luke	4	**Art:** Simple background and concept art.	~~**Play a Roguelike game over the break.**~~	**Reuse Code Base /Audio:** Some basic sound effects.
All	5		**Database:** Begin development.	**Database**: Integrate into game.
	6			

Figure 2: Scrum sprints

This was our first attempt at using scrum, and it was very effective for focusing the team and getting our workload under control. Over the course of the project we would add to this sprint sheet, until it finally spanned the course of the year.

3 Literature Review and Research

Initial Research

At the end of 2nd year we had decided that we would group together for the 3rd year project. Over the summer we had multiple meetings and brainstorming sessions to try and fine tune our game idea and expectations for the project. We used Google Drive to collaborate on documents where we had our base plan for the game. This allowed us to easily tweak and change the documents as we learned more about the project and timeframe that we had.

From the start, we knew that we wanted to use C++ and SDL2 for the game because of this we were able to start researching the languages during the summer break. This gave us a head start and allowed us to become familiar with both before college started.

Books

After speaking to students in years above us, we were able to learn of the two books used in our upcoming Game Programming module. We decided that getting these books before we returned to college would be key to getting a head start on the project and improving our understanding of C++ and SDL2.

The first book we looked at was 'Beginning C++ through Game Programming (Micheal Dawson, 2010)'. As mentioned above, we chose this book because it was the basis of our upcoming Game Programming module. This book was a great help because it gave us a comprehensive overview of the C++ language. It showed us why Object Orientated Programming was such a good model for creating games. It helped us to gain an understanding of many programming concepts such as inheritance and memory management. The best thing about this book is that it gave simple examples of how all the programming concepts could be used in relation to game development.

The next book that we started to learn was 'SDL Game Development (Shaun Mitchell, 2013)'. This is the book that we used in class while covering SDL. It covered most if not all of the functionality that SDL can bring to a project and it was also what the framework that we were given in class was based on. By reading this book we were able to learn how to handle the creating and deletion of textures and sounds, how to handle inputs from the user and how to render our game to the screen.

When we were having trouble with a topic in C++, we used 'Programming principles and practices using C++ (Bjarne Stroustrup, 2008)'. As this book goes incredibly in-depth, we found it useful to look up what we were having trouble with in the index and read the relevant chapter or paragraph. This was extremely helpful for understanding complex concepts such as smart pointers or abstraction.

Early in the projected cycle we read 'Clean Code (Robert C. Martin, 2008)' this provided us with the means to select a coding and convention standard that we would all use throughout the project. This prevented a slowdown in development because when paired with the commit messages on Bitbucket, we could easily understand each other's code and we could often just look at function names to see what they did.

Online Resources

Over the course of the development cycle, we looked at many online resources and they were often more helpful than the books that we had read.

The main one that we used was the list of C++ Core Guidelines that is written and maintained by Bjarne Stroustrup and Herb Sutter. These guidelines are a list of do's and don'ts for modern C++ that aims to make code less error prone and more maintainable. While we didn't follow this ruleset religiously, we feel that it helped us to write very efficient, simple and tidy code.

Another great asset for us was the 'Beginning Game Programming v2.0' online tutorial series by Lazy Foo' Productions. This tutorial series covered a lot of relevant SDL topics. We used these extensively while learning SDL at the end of the summer break. The tutorials were very easy to follow, well explained and each task was broken down into small rememberable segments.

YouTube was also a great asset. Sometimes looking at text just wasn't enough and when that happened, we turned to YouTube to look for a video tutorial. We used popular tutorial channels such as the SDL tutorials created by 'thecplusplusguy' and 'Let's Make Games'. Both channels had simple and easy to understand videos and the hosts were well spoken and explained everything slowly and thoroughly in a manner suited to our current abilities.

Finally, we used 'StackOverflow.com'. Whenever we had to research how to implement something into our code, we often ended up at Stack Overflow. This forum has a very active userbase that is happy to try and help with any programming question you might have.

In Class

As we had read the books that were used to form the Game Programming module, the extra information that our lecture offered in his slides and the various diagrams really helped to tie everything together. The coursework and assignments really helped to enforce the importance of OO and the programming patterns that we used such as the game loop or the finite state machine.

Games

The final type of research that we did was playing popular top down rogue like games. This allowed us to explore the good and the bad from the genre. We played games such as 'The Binding of Isaac' and 'Enter the Gungeon'. As we played them we would take note of the various programming patterns that we thought they were using and we also looked at gameplay elements that we could implement into our own game and improve. While playing them we took note of things like the size of the objects, the difficulty of the game and even how long it took to move from one end of the screen to the other.

4 System Design and Configuration

Collison Detection

The original framework provided collision detection in two forms:

- In the form of Axis Aligned Bounding Boxes (AABB) with aid from SDL's built in Rectangle data type, *SDL_Rect*

- In the form of reducing the position of an object to that of the same structure as the placement of tiles, in terms of where that object was in a grid rows and columns rather than where the object was in terms of their x and y values coupled with their width and height.

For the AABB functionality, using the example of the Player colliding with an Enemy, the Player and a vector of type GameObject is passed into a function. This vector contains the instantiated objects of the Enemies on the current map including objects such as the Pickup and the Player. The Player's x position, y position, width and height of their model is assigned to the respective elements of an *SDL_Rect.*

For all objects that were a type of Enemy an *SDL_Rect* is made for them and their position and size is also assigned to that *SDL_Rect*.

The two rectangles are passed into the AABB function *RectRect(SDL_Rect* A, SDL_Rect* B).* This function returns a boolean value.

They are given a small buffer of 8 pixels and compared in the following order:

- If the bottom of A is less than the top of B
- If the top of A is more than the bottom of B
- If the right of A is less than the left of B
- If the left of A is more than the right of B

A return of *false* is given in each realised comparison case, otherwise there has been a collision and a return of *true* is given.

Further to this AABB comparison a check is made against the dead and dying state of the Enemy. If *RectRect(SDL_Rect* Player, SDL_Rect* Enemy)* returns true and the Enemy is not dead or dying the *collision()* function of the Player is called, in which the Player takes a point of damage, beginning their invincibility frame period.

At the end of each loop the SDL_Rect storing the Enemy is deleted. At the end of the loop entirely the SDL_Rect storing the Player is deleted.

For the functionality of reducing an object's position to that of a tile, using the example of a Player colliding with a tile, the Player and a vector of type TileLayer of collision layers is passed into a function. This vector contains within it the identification numbers for each tile in the 20 x 15 grid. With locations where there is no tile, the assigned ID is 0.

Temporary integer variables tileColumn, tileRow and tileid are made. The tile column is set equal to the Player's x position plus the Player's width divided by the size of the Tile. The same is done for the tile row but uses the Player's y position and the Player's height.

For example, if the Player's x position is 500 and their width is 32 the value 532 is divided by the tile size, 64, to give 8.3125, truncated to 8.

If the Player's y position is 400 and their height is 64 the value 464 is divided by the tile size, 64, to give 7.25, truncated to 7.

This tells us that the Player is currently within column 8 row 7 of the tile grid.

The tileID from that position in the grid is then extracted. If that tileID does not equal to 0, the tile space at that location is not empty and a collision has occurred.

The failings from this implementation is that it cannot check the collision properly due to the truncation and use of the general form of the Player, often giving incorrect results depending on what angle the Player came from.

These solutions could not be found adequate for all the collision considerations contained within what we wanted to implement in terms of game mechanics.

Specifically, we required collision detection and reactive functionality to allow objects to collide with terrain, on any one of the four edges, and still be able to move on a non-collision inducing axis while colliding on another axis.

In the resolution of this issue, we implemented two more forms of collision detection to the project:

- In the form of a minimal displacement function that could, with some failings, be able to return on what side of a tile a collision occurred.

- In the form of a velocity guided ahead of time positional check that could detect a collision on an any given axis and stop velocity on that axis.

The first solution was the attempt at using minimal displacement to correctly reverse the change in position by knowing at which edge of a tile a collision was going to occur.

The reasoning was that if we knew the Player was moving, for example, with positive x and we collided with the left-hand side of a tile, we could deduct the positive change so that the Player was placed back into the correct position.

We were naïve in the sense of that we weren't really stopping a collision from occurring, we were just "repairing" it in some mediocre way.

The solution, although lacklustre, worked for sliding along terrain for both axis but fell short when the Player could access, from both angles, any of the corners of a tile.

As the Player moved just a single pixel into any of the extreme corners of a tile the minimal displacement resolution would fail, giving a displacement of 1 on two different edges. As only the perimeter of the displacing object was considered the Player was then free to glide right through the tile.

We would still have more problems if we continued down the minimal displacement route, we had to be smarter about how we were going to resolve a collision. So, we looked to velocity.

As often, the simplest solution was the most useful one. With knowledge of how to reduce the position of an object to the positional structure of the tiles, we used the transformed position given an applied separate x and y velocity.

In addition to the previous implementation we changed the functionality to check every pixel of the Player, looping through their starting and ending x and y positions. I understand this to mean pixel perfect collision. (This meant a large performance impact on the run-time of this type of collision checking and had to be taken into consideration, discussed further in this section)

To do this we took the velocity based on Player input before it was applied to the position. We then set a temporary vector to the Player's current position. This temporary vector would then have the x velocity applied to it.

If this vector did not collide with a tile the x position of the Player would equal the x position of the temporary vector.

If a collision occurred the x velocity of the Player would be set to 0. The same was then repeated for y velocity, checking where the Player *would* be against where the tile was located and then allowing or disallowing the change in velocity.

This provided the Player with the ability to glide along walls, run into any corner without entering the space of the tiles and properly collide with a tile from any edge.

The minimal displacement functionality was not used in the submitted version of the project. As a future improvement it could be used, with some tweaks, to allow the decision of where to move a tile that was being "pushed" by the Player.

The AABB functionality was used in the project for checking if the Player walked into an enemy.

The basic form of the checking the position of an object against the position of the tiles is used to check the collision for all the bullets collision considerations against tiles, including the destructible terrain.

The advanced form of checking the position of an object against the position of the tiles with pixel perfect checking given an applied velocity to the position of that object is used for the Player and the Enemies colliding with the tiles.

We felt that we needed to not use more resources where it was not needed and decided the collision priority in this way.

As an unintended result, the way that collision is resolved in the advanced form of the tile positioning meant that the Uno enemies “bounced” off the wall in the opposite direction at an opposite angle.

This was the case because the Uno and the Assassin both invert their velocity on collision. The Uno would collide with the wall, stop their velocity on that axis from the tile position checking, invert both velocities and then go off on the unstopped, inverted velocity.

Dynamic Difficulty

As a core part of our project we wanted to implement difficulty that could, in some way, be adjusted based upon the Players actions or indeed in-actions during gameplay. This was to be done without the Player having to change a setting or select a difficulty at the beginning of the game. We wanted a seamless transition between the difficulty levels.

Such a system also affords the Player time and space to become comfortable with the games mechanics such as the movement and shooting or the Enemies and their patterns, before being challenged in a greater capacity later, but only when they are ready.

At the foundation of this dynamic difficulty implementation is an ever-changing difficulty index. This index is a point scale ranging from 1 to 12 where 1 is the easiest level of difficulty and 12 is the hardest level of difficulty. When anything attempts to influence it, it will stop either at the floor or the roof, as to not go above or below them during execution.

Factors that influence this index:

- Level completion times.
- Damage taken from enemies.

Specifically, if the Player completes a level in under 10 seconds their index increases by 2 points. If the Player completes a level between 10 and 25 second their index increases by 1 point. If the Player takes longer than 25 seconds to complete a level their index decreases by 2 points.

In terms of the damage taken, for each point of damage taken by the Player their index decreases by 1 point.

Factors that this index influences:

- The type of map that the Player will next attempt.
- The rate of reduction for the score.
- Enemy lethality.

Specifically, if the difficulty index is above 8 the next map that the Game will load for the Player will be a hard level, if the index is between 4 and 8 a medium level will be loaded and lastly if the index is below 4 an easy level will be loaded.

Figure 3: Heads Up Display

The difficulty index can be seen in the HUD at all times with the colour of the text correlating to the colour of the background in each map. This gives the Player the opportunity to relate the difficulty of each map the colours, as intended by design.

For of the rate of reduction concerning the score the algorithm is Score equals to Score minus 0.2 divided by the difficulty index divided by 2 with the division by 2 occurring before 0.2 is divided by it.

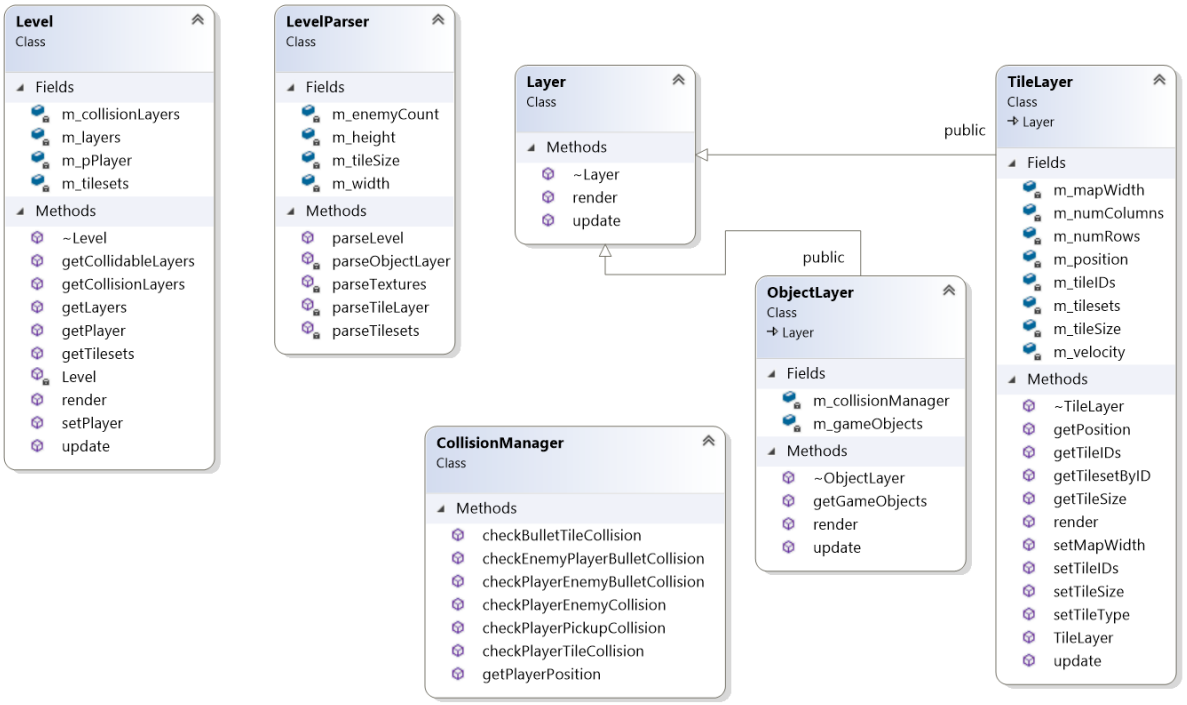

Figure 4: Level creation and order structure

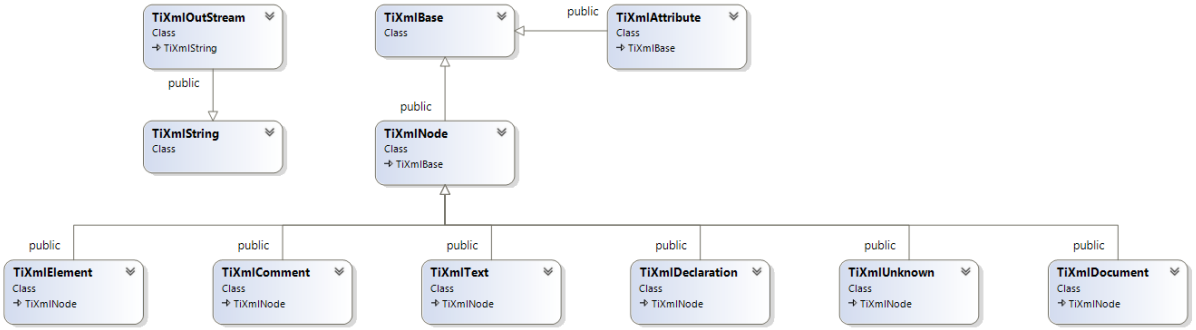

Figure 5: XML Parsing

This rate then provides the score with a reduction of 0.4 with an index of 1 and 0.033 recurring with an index of 12 every 16ms of time spent.

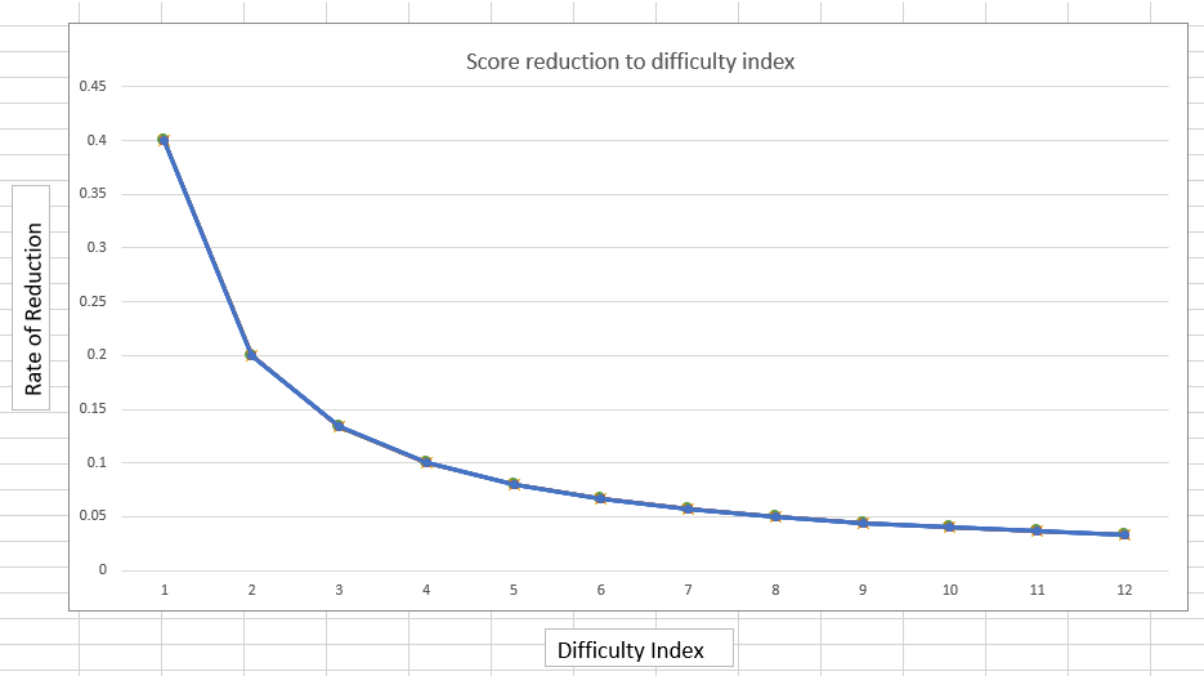

Figure 6: Score Reduction to Difficulty Index

Lastly, the lethality of the Enemies is increased or reduced based on the difficulty index.

For example, in the case of the Assassin enemy, they will fire 1-3 bullets each volley with the same decision tree of the easy, medium and hard maps. 1 bullet if index below 4, 2 bullets if between 4 and 8 and 3 bullets if above 8.

Alien Attack refactorization

The codebase that was given to us held within the completed game of Alien Attack. This codebase provided the backbone of our project.

The high-level components of the original codebase and assets included:

- XML Parser for reading level files.
- Object Factory for streamlined creation of GameObjects.
- Finite State Machine for better control of state.
- Extensive inheritance from one GameObject super class. (See Figure
- Texture Manager for rending the graphics.

- Inputhandler for processing user input from the keyboard.
- Sound Manager for Audio playback.
- Collision Manager with AABB and tile positioning based collision.
- Core Loop based execution.
- Single scrolling playable level with basic data driven entities.
- 32 x 32 tile size. 150 x 21 map size.
- Window at 640 x 480 Resolution.
- Single element UI.
- Basic Enemies.
- Linear shooting
- Single level exit.
- Basic art assets
- Basic sound assets.

We modified or added to this with:

- The control of Memory over subsequent levels.
- Attribute class to better store the GameObject variables.
- Dynamic difficulty control.
- RoomTransition state, Tutorial state, Controls state. Rewriting the flow of the states to suit our project.
- Re-use of inheritance to supply more types such as Pickup and our new Enemies.
- SDL_Text functionality to the TextureManager that allowed us to effectively print out text to the screen .
- A new Controller Library written 2 years after the release of the SDL Game Development book.
- Improved collision detection and response.
- Database connectivity and management.
- 51 static screen procedurally ordered playable levels.
- 64 x 64 tile sizes. 20 x 15 map size.
- Window resolution doubled.

- 5 Item pick ups.
- 4 Enemies with distinct personalities.
- Four directional shooting.
- Dynamic level exits.
- Custom made art assets.
- Custom and open source sound assets
- Scoring system.

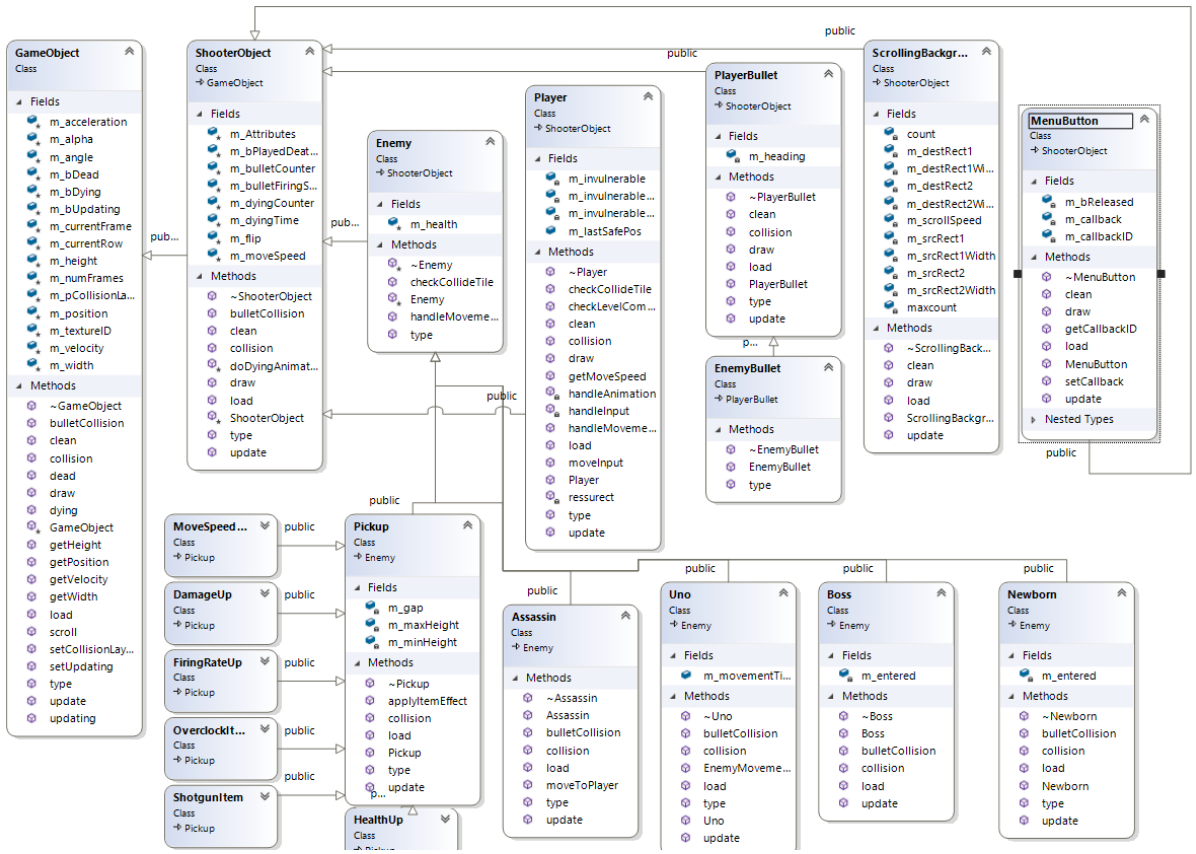

Figure 7: GameObject Inheritance

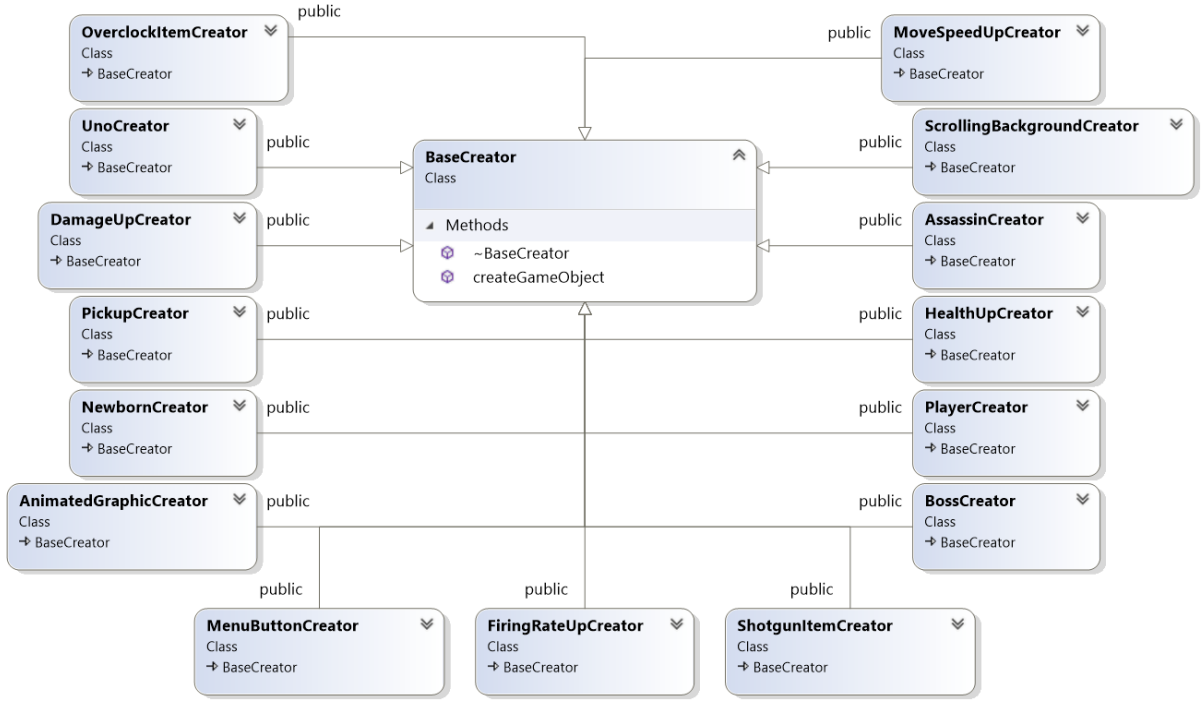

Figure 8: Object Factor for the GameObject

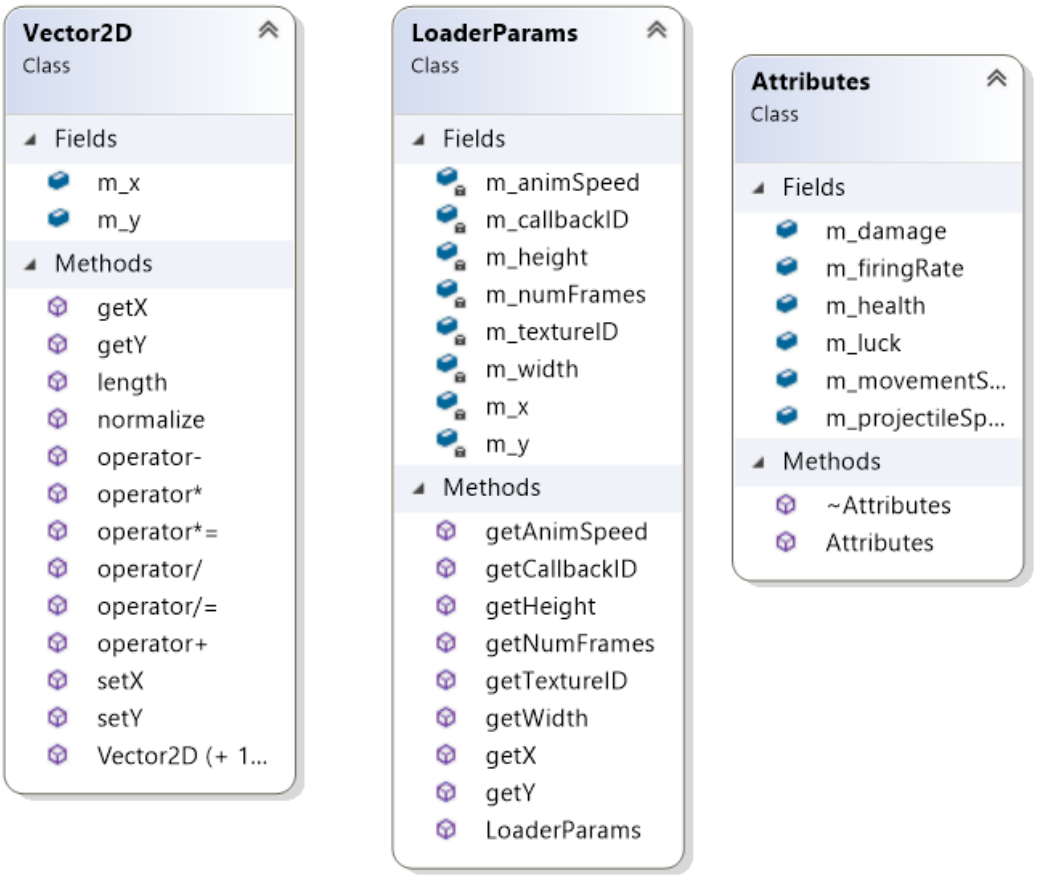

Figure 7: GameObject class abstracted member variables

Main loop execution

The entry point and beginning of execution for our project begins in main. From main the console is allocated and initialised to be used alongside the output window.

Main then calls to the Game to initialise by:

1. Initialising all SDL components.
2. Creating an output window.
3. Creating a renderer, loads the sound assets in.
4. Initialising the controller.
5. Registering the distinct GameObject types in the game.
6. Instantiating the GameStateMachine
7. Changing State to the MainMenu state.

Main then begins the Core Game Loop and:

1. Stores the time in milliseconds since the start of execution.
2. Processes Input from the user.
3. Call on the Game State Machine to run the update of the state on the top of the stack.
4. Calls on the Game State Machine to run the rendering process of the states on top of the stack.
5. Stores the time in milliseconds since step 1.

If the main execution cycle is finished and 16 milliseconds has not passed a call is made to SDL_Delay to delay for the remaining time.

Main continues to loop until *quit()* is called in Game.

The Finite State Machine

Use and understanding of the finite state machine programming pattern was a large part of our project. A finite state machine has a few simple rules that you must follow when creating one.

- You must have a set of states that the game can be in.
- The game can only be in one state at a time
- States must point to other states based on input of conditions being met.

In the framework that we were given there was a finite state machine implemented that were used for the various menu's and the play state. We use these states because the functionality that we need from each of them is vastly different from each other. We use a class called the StateParser to parse and make use of each state. Each state is responsible for creating and deleting any resources that it needs and thanks to this we were able to easily manage the memory usage to an extent.

We used states to show the user the controls and the tutorial screen. Because the play state can't point to itself, we needed to create a transition state so that when we progressed to the next level, we would have a buffer in-between the levels. The finite state machine has allowed us to easily define functionality that we only want in certain parts of the game.

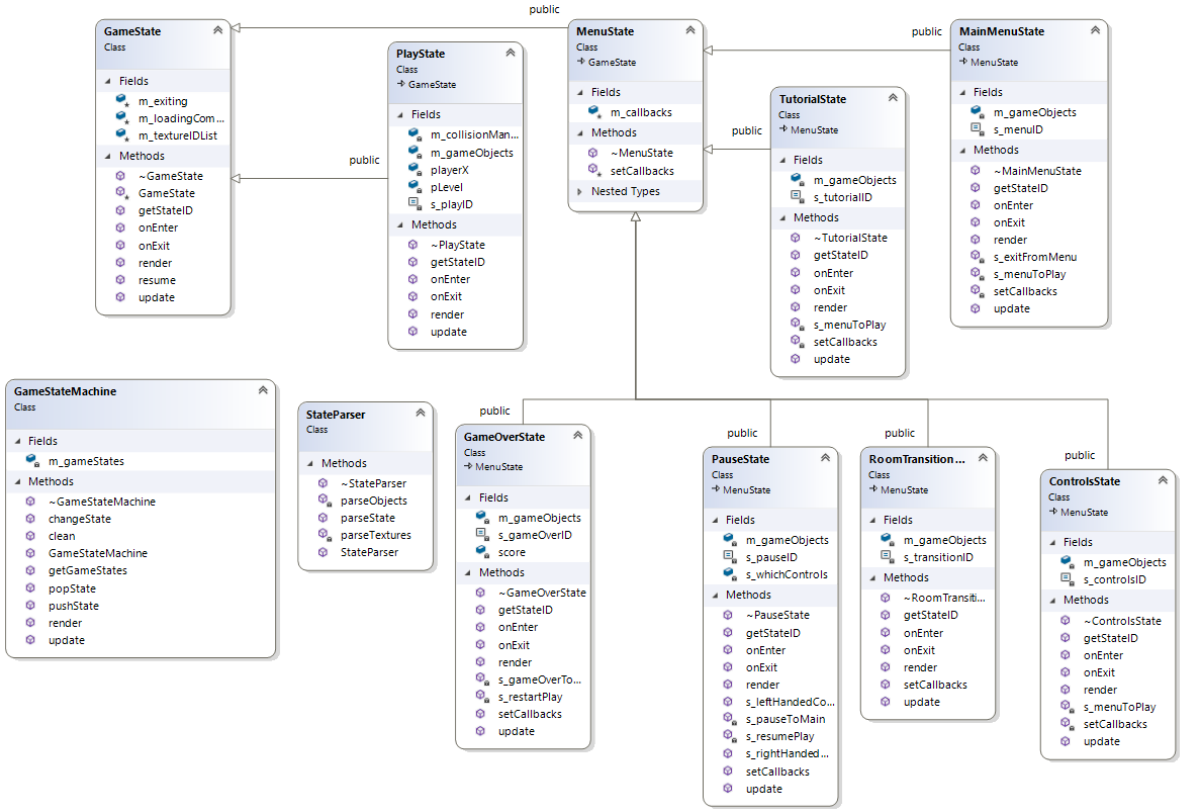

Figure 8: GameStateMachine and its possible states

The Input Handler

All of the user input, whether it was through keyboard, mouse or game controller was handled by the InputHandler class. This class is one of the singletons that are in the game as we only ever want one instance of it and it needs to be accessed from many different classes in out game. The Alien Attack framework that we were given came with this class although it was outdated and didn't provide much of the functionality that we needed.

We had to add an updated controller library that was made for SDL2. This updated library is much better than the old library that was originally in the code. Doing this required a complete rewrite of how the controller functions took in and returned values, this overhaul was done by Nathan. These changes allowed us to make simple calls to the input handler as we required them. These changes include every function that check to see if a button or key has been pressed and the function required to reset the input handler when we change states or restart the game. Nathan also added a function for haptic feedback which will make the controller vibrate when called.

Now that we had the ability to use a controller we had to map the controller buttons to actions in the game. Luke was given this task. As our Player character had four directional shooting, we decided that it would be best if the shooting was controlled with the face buttons (A, B, X, Y) of the controller. This was to avoid the user thinking that they could shoot in all directions by using the right joystick. This was something that we noticed while playing popular top down rouge likes in the market. They nearly always used the joystick for the shooting even though you could only shoot in four directions. From the start, having tight responsive controls was a goal of ours and we felt like that wasn't possible if we followed the trend of using the joystick for shooting, this is the reason we decided to use the face buttons.

Using a joystick for the movement is another story. We wanted to have fully circler movement as we felt that top down games that didn't limit your movement to the 4 cardinal directions were more fun and felt nicer to control.

We were able to achieve this by using percentages. The joystick direction and force are returned as a value between -32000 and 32000. To make sure that the movement was fully circler, we needed to get the percentage of the joystick value that was X and the percentage that was Y. To do this we made sure that both values were positive by multiplying negative values by -1. After this we added the X and Y value together and stored it as the full axis value. By dividing the X axis value by the full axis value, we were able to get the percentage of the full axis value that was X. We then did the same with the Y value. Once we had these two percentage values we could map them to the Players movement speed. Finally, we would return those values to serve as the Players X and Y velocity. This would allow us to make the Player move at any possible angle while maintaining the same speed. As an example, refer to the graphic below

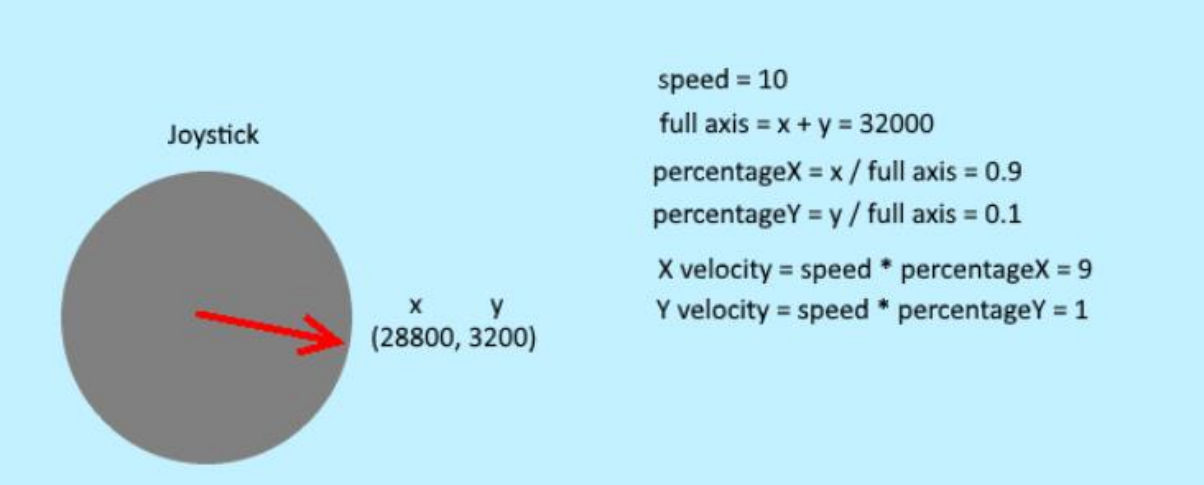

Figure 9: Joystick cirular movement example

Database

One of the main additions in terms of System Design was the introduction of a database. This database was a local instance of MySQL 32 bit connected over a TCP connection at 127.0.0.1:3306. The connection was facilitated by the 32-bit MySQL C++ connector 1.1.9 and boost libraries for additional support.

We wanted the database to be able to store the attributes for the Player and Non-playable characters as well as store the statistics that we could generate in game.

The initial script, read and writes were handled by the program while the creation of the schema was done through MySQL workbench.

The datatypes within the program include the driver, the connection, statement and a result set.

There was an attempt and extensive testing done to try to maintain a constant connection to the database while playing the game. Unfortunately, all attempts to maintain the connection led to large memory leaks and instability.

To this effect, functions were created to connect and disconnect from the database and were used for every read write.

The connection function passed the driver connection to the connection datatype, created a statement from that connection and executed the *USE* statement. The disconnection function

freed the memory stored for the result set, the statement, and the connection. The connection was then manually closed.

Only the driver remained throughout execution.

On one side we had no more memory leaks, however, the requirement to constantly connect and disconnect massively impacted the performance of the program. The threshold for updates was a cycle time of 16 milliseconds but every read or write to the database took 20-30 milliseconds. This meant we had to be very careful about where and when we made these calls.

We felt that it was better to carefully manage where this update delay was to remedy this issue than to allow any intended leaking of memory.

These failures meant that creating additional tables and storing statistics became a lot more work than we intended, thus the functionality was scrapped for the project submission.

We had high hopes for using the database as part of our project, but the myriad of technical difficulties made it frustrating to expand upon it.

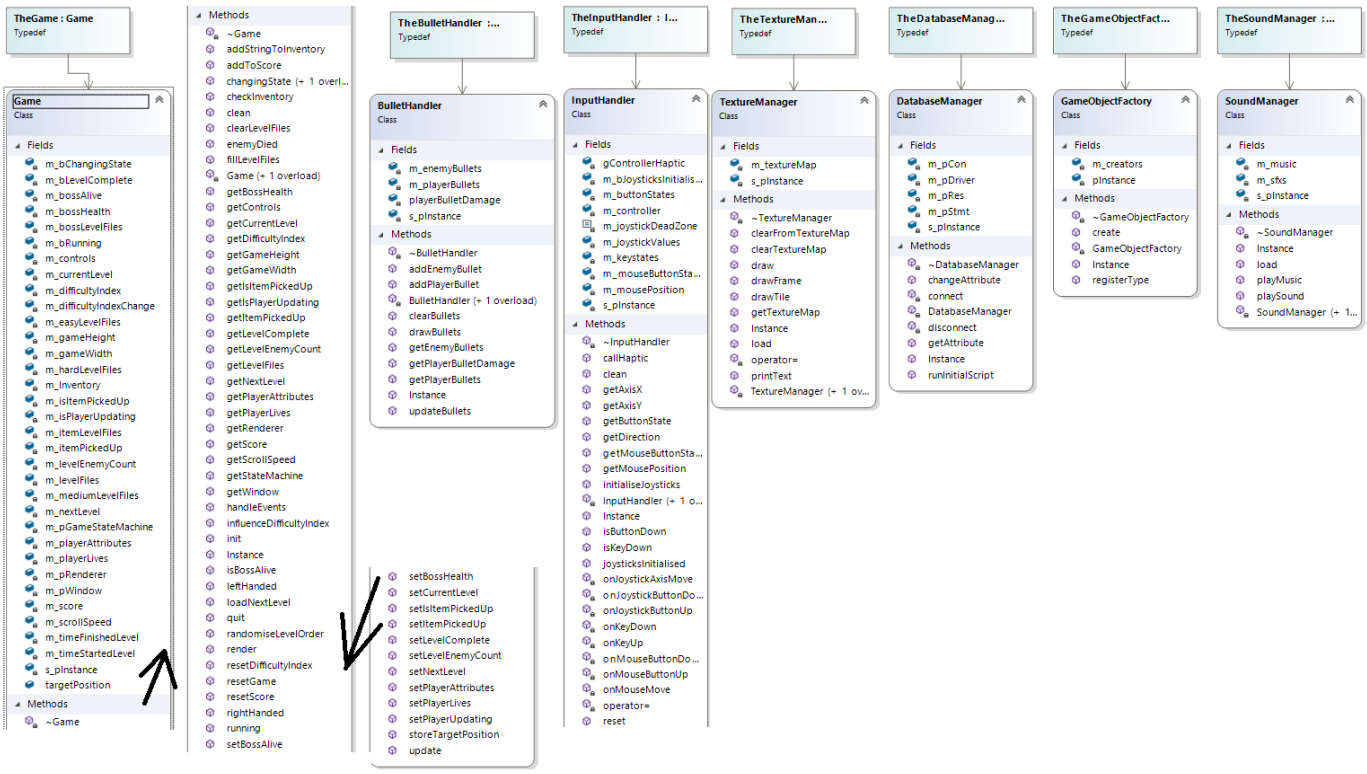

Figure 10: Singletons

Padraig's Question

During the Q&A session following our Presentation, Padraig asked us a question about how the update rate within the game could impact networked play. We felt like we didn't have a satisfactory answer, so we will include our thoughts about it here.

From the outset we had decided that our project would not implement or attempt to implement any network connectivity for online multiplayer or online play. This meant that we had given almost no consideration or thought to this area of video games.

From our own understanding of online game servers and how they serve the clients the server that the client is connected to will have its own update, or tick rate, usually given in hertz. An average online game might have a rate of 20 ticks per second. This would become intertwined within terms of how quickly the game could poll Player input and update for that Player.

The second consideration is the performance of the hardware that the client is using to play the game, generally bottlenecked by the GPU. Our game must run at 60 fps to be playable. If it was running extremely low then inputs from the Player would be missed and updates would be delayed.

This creates a dynamic between the server and client where if the server was experiencing extremely low tick rates the performance of the client hardware wouldn't matter and if the server was running smoothly the Player would only have missed inputs and delayed updates if their hardware performance was not up to scratch.

5 Testing & Implementation

Whenever a new feature was created for the game, we would conduct a series of tests in the current build of the game to confirm that the feature was correctly implemented. This helped a lot as it avoided creating future conflicts or dependencies. The most important adage for the team to follow was to always test the feature in the most up to date version of the game to avoid the scenario of having to scrap hours of work, so source control proved invaluable when testing and adding features.

When testing a feature, we would use a few essential tools and techniques to ensure the feature was being implemented correctly:

1: Performance Profiler

A debug tool that is built into Visual Studio 2017, this tool detects memory and CPU usage while the program is running. This was instrumental in detecting memory leaks, or features that were not working as intended. If a leak was spotted, the group would be alerted and the problem would be worked on until it was fixed, by one or more members of the group.

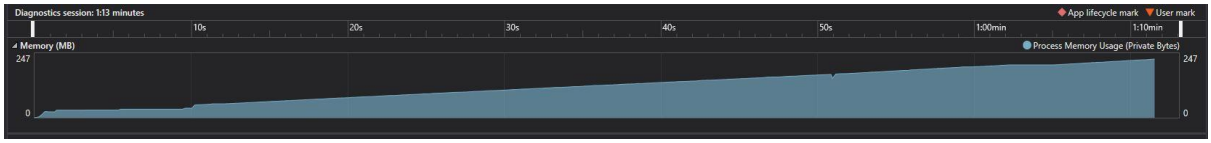

Figure 11: Detection of a leak

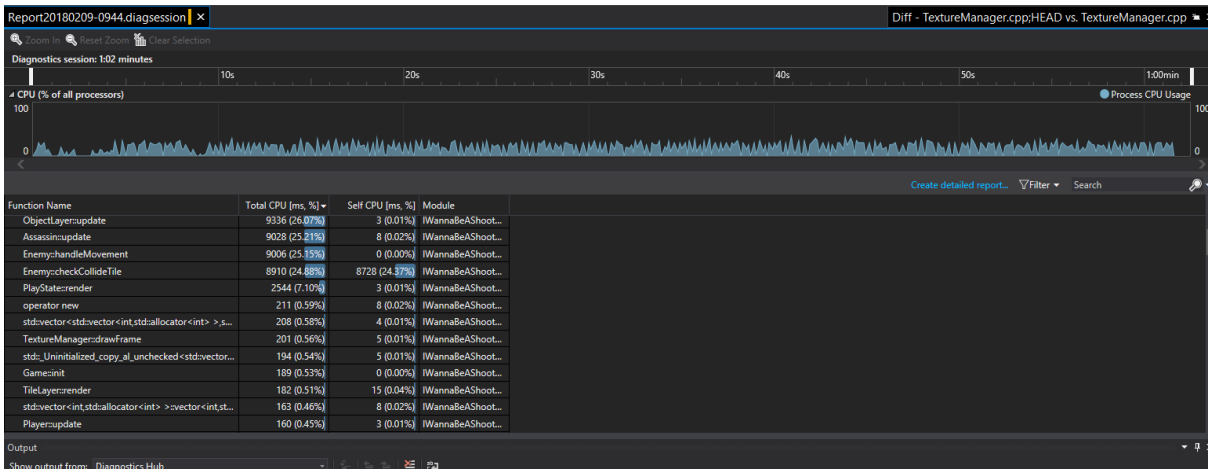

Figure 12: CPU execution time per function analysis

2: Debug Macros

A short macro was implemented in Game and some of the classes that would output a short line to the console informing the tester if the class was being created when the game was running. These were useful later in development when we had multiple types of enemies and items in the game to make sure they were being created, and if they weren't being displayed for whatever reason then they would be investigated. Another benefit of using these macros is that they could be turned off very easily.

```
#define GAME_DEBUG
#ifdef GAME_DEBUG
#define GAMEDEBUG_MSG(str) do { std::cout << str << std::endl; } while( false )
#else
#define GAMEDEBUG_MSG(str) do { } while ( false )
#endif
```

Figure 13: GameDebug Message Macro

```
GAMEDEBUG_MSG("Uno at: " << this << " spawned");
```

Figure 14: Macro being called with a class

3: Watched Variables

Watching variables while debugging the game was very important, as it gave a live update to changes that we were happening within the selected variables. This was useful when we would add a new item that changes the attributes of the Player, or for keeping track of the damage dealt to the Player and enemies.

4: Printouts

By having the game print a statement to the console at every update, we were able to test for a myriad of errors while still running the game and test for errors in real time.

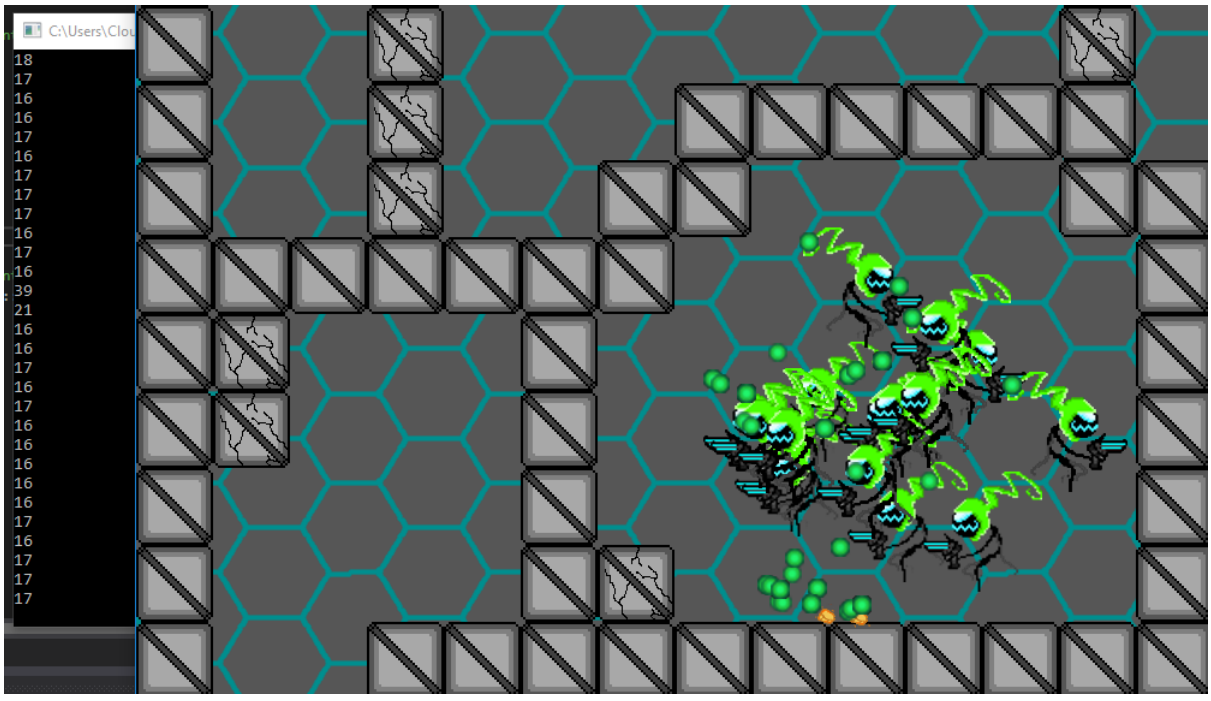

Figure 15: Testing update rate with 100 Assassins, above acceptable amount

5: Visual Test

Another, perhaps more basic form of debugging, was to run the game and spot errors that may appear, such as functions not working as intended or graphical errors, etc.

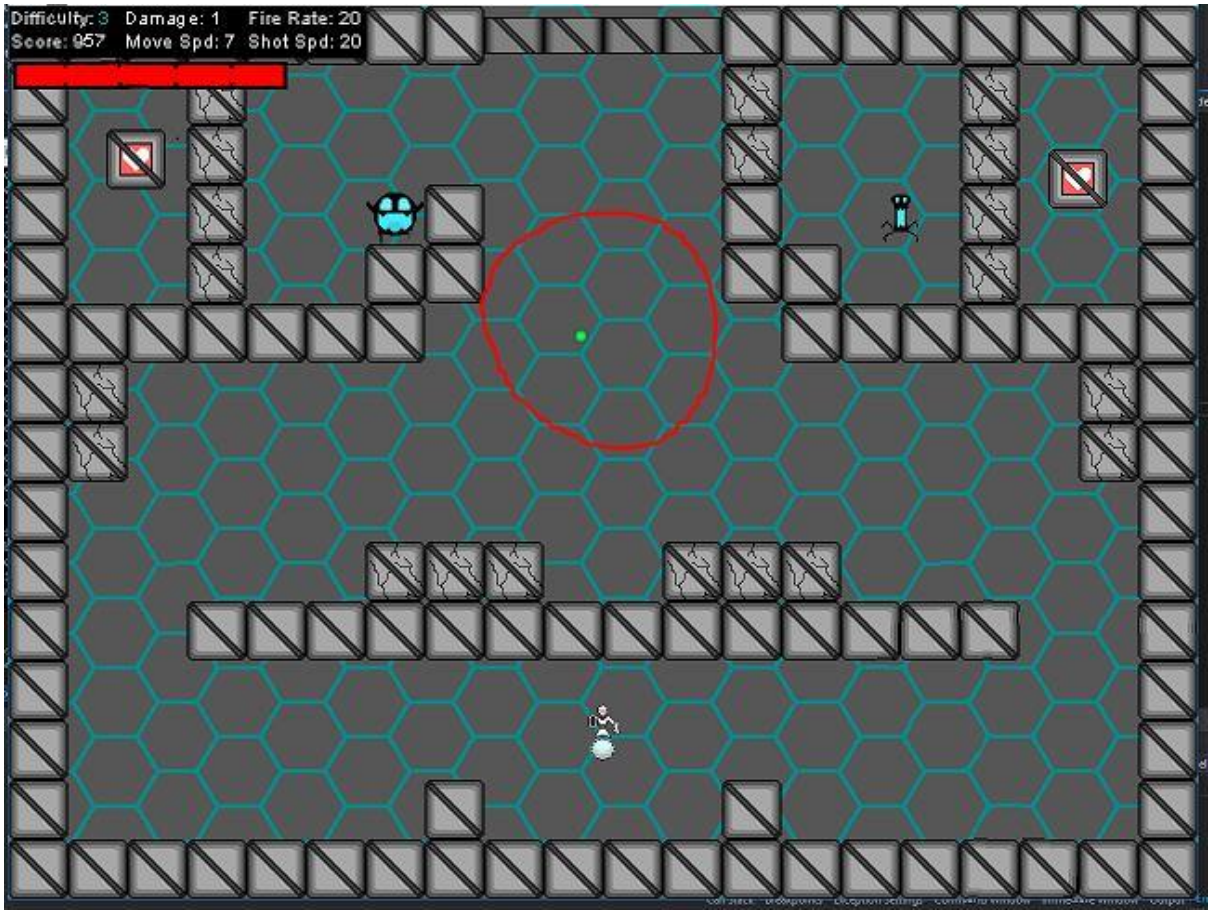

Figure 16: Enemy not displaying but acting as normal otherwise

Problem Features

There were some features that were either cut or were difficult to implement, and perhaps with more time we could have figured out how to fit these into the game:

Pickups:

Pickups were at first proving to be difficult to implement because of collision detection; outside of tiles, enemies and the Player collision did not exist. Because of this it was decided that collision should inherit from the Enemy class just to make our lives easier.

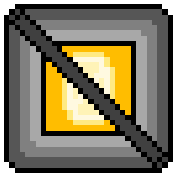

Figure 17: A pickup item

Animated Background:

Initially, we had the idea to create a sprite-sheet for the background and have it animated. However, when it came to implementing it we noticed that it caused huge amounts of strain on the CPU and memory after examining the profiler. While we tried to fix these issues, eventually we decided that it wasn't worth the stress and spike in memory, so we cut it from the game.

Unos and the Game Breaking bug:

There was, unfortunately, a very rare event that would cause the current map to become impossible to complete, keeping the Player trapped inside the map with the doors locked even after defeating all of the enemies in the room, which is the intended trigger for the doors to open. Initially we were unsure what the problem was, however after testing each enemy individually we deduced that the bug was caused by one of the enemy types, the, somewhat ironically bug-like in appearance, Unos. We noticed that occasionally these enemies would sometimes just vanish from the map, however the game did not count these disappearances as deaths and would not decrement the enemy counter needed to open the door. After trying to find the cause of the problem for many hours, we were running out of time for the project deliverable, so we invented a failsafe as a temporary workaround. We are still focusing our efforts on rooting out this bug and fixing it.

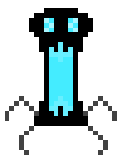

Figure 18: The Uno enemy

Future Developments & Cut Features

There were some features that were either cut or were difficult to implement, and perhaps with more time or better time management we could have added them before the due date. However, there are some features that we will add for Games Fleadh:

Leaderboard:

This was left to the wayside during development in favour of improving the core of the game. However, since our game already reads and writes to a database it would be a relatively simple task to create a new table, write to it and display high scores with a name next to it during the game over screen. This is certainly something that can be added.

Endless Mode:

This is another simple change of code- where after the boss is defeated the levels will all loop again, but the Player will keep the difficulty level along with all of the powerups they obtained, and all of the enemies will become more powerful. This could give the game more longevity and attaining a high score more competitive.

More Enemy Types:

We originally had more than six types of enemies planned out, but unfortunately we were running low on time and the enemies we already had added a good amount of depth to the game, so a lot of these ideas were scrapped entirely. We are currently throwing around ideas about how to add “Champion” enemies into the game, however.

Multiple Bosses:

We originally wanted to have at least three bosses in the game, and with more time we certainly could have, but there were more pressing matters after we implemented one boss, so the other boss ideas were left out of the delivered product.

Survival Mode:

This could be implemented as a side mode, where the Player plays the game normally but their maximum health is reduced to 1. However, since it would have required an overhaul of the menu and UI we decided against adding this feature.

6 Critical analysis

In terms of how many goals we achieved from the original list of features we wanted in our game, overall the project was a resounding success. We hit many of our target goals, and those that we missed were peripheral and generally not integral to the core experience of the game. What we created was a finished product, but one that has room for improvements.

By examining our formal proposal, we can see which features we kept, which ones were cut, and which ones were adapted to fit the finished product.

Balanced Procedural generation:

This is a feature which we adapted to fit our finished game, and while what is in the game wouldn't strictly be counted as "procedurally generated", we believe we made the correct call to craft the levels by hand and instead opt for procedurally ordered playable levels. This has multiple benefits; Firstly, it allowed us to make levels crafted for the Player, instead of relying on some kind of level generator, making those levels the best possible experience for the Player. Secondly, it saved us from making a level generator algorithm, which would have taken much longer and been much more difficult to integrate into the program. And lastly, having a set of hand crafted levels allowed us to order them into difficulty brackets with great ease, and relying on a level generator to balance the game could have resulted in a diminished experience for the Player. Overall, we think this was the correct option to keep our vision of playing pseudo-random ordered levels, while also creating the best possible user experience.

Dynamic difficulty:

Although perhaps not as advanced as we proposed, our game very much contains dynamic difficulty. We have two factors to determine the difficulty: the amount of damage the Player has taken, and the amount of time the Player takes to clear all of the enemies in a room. Instead of the enemies dropping items or health pickups when the Player is having difficulty, instead the Player is given an easier block of levels to play, and those levels have more health pickups within them. Contrariwise, if the Player is doing well they'll be given a more difficult block of levels with more enemies and a more difficult level layout. In addition, the Assassin enemies will fire more bullets depending on the difficulty level (1 in easy, 2 in medium and 3

in hard). Given the amount of time we had and our greater understanding of the language, we think that this is a more than fair compromise, if it can even be called that.

Multiple Characters, Bosses, Items:

We intended to have multiple Player characters, as well as multiple bosses in our finished game. While we have multiple items added, we were unfortunately only able to add a single playable character and a single boss before the deadline. However, this was more of a matter of weighing the benefits versus the effort needed. In order to add multiple Player characters we would have to overhaul the entire menu system that we had implemented, which would have taken up valuable time that could be used to improve the gameplay experience instead. As for the multiple bosses, we just merely ran out of time. However, we hope to add another two bosses by the time of the Games Fleadh.

Statistics:

While we do have a statistics system in place, it works somewhat differently to how we originally proposed. Since we don't have multiple Player characters currently in the game, we instead have a character with base stats that are improved as you collect pickups throughout the game. We also don't a system in place that shows how many enemies the Player has defeated, instead we have the Player attempting to get a high score by defeating enemies as quickly as possible, with each enemy rewarding the Player +100 points while the score constantly decreases. While this is a slight deviation from our proposal, we think this system adds a competitive edge to the game and emulates the systems of other games in the genre that have proven to work.

Permanent death:

This system was implemented fully into the game. When the Player is defeated, their score is displayed to the screen and if they want to continue playing they must start from the beginning, getting better at the game as they play. A good feature to add to this however is the ability to save your score upon death, which would give the Player a goal to surpass.

Good Memory Management:

We were extremely careful to manage our memory appropriately, to avoid memory leaks and to allocate it as wisely and conservatively as possible. We think that the way we have handled our destructors and solid structure means that we have exceptional memory management in place.

Polished Core Gameplay:

After testing this game as a team and individually for numerous hours, as well as performing demos for friends, peers and anyone we could get our hands on who was willing to try the game out, we think that our gameplay loop is of a high level of quality. Veterans of the genre, as well as new Players were all able to grasp the game and succeed in proceeding through the game. Those who were familiar with the genre were challenged by the harder difficulty levels and were aiming to get a high score, whereas new or inexperienced Players still had fun just playing the game on the easy or medium difficulty levels. Response to the game was universally positive, with people praising the fluid movement and shooting mechanics, as well as the predictable enemy behaviour between the three enemy types and how they interacted, as well as the level design and the look of the game. Overall, we consider this the goal in which we succeeded most.

Clear and concise user interface:

We crafted a UI that was unobtrusive and provided all of the essential information to the Player; such as the Player's stats, current/max health, score and level of difficulty. This was our goal from the start and we were overjoyed to implement it exactly how we wanted to from the beginning.

Clear attainable objectives:

The primary focus of the Player is obtaining the highest possible score and defeating the boss, and every action the Player takes informs this decision. The longer the Player takes to complete a level the lower their score will be, but if they become reckless and take damage too often they won't progress to the higher difficulty echelons where the score gained has a multiplier and ticks down at a slower rate. But the higher difficulty levels are, well, more

difficult, and the Player faces a higher risk of being defeated. All of these things inform the Player actions and are in service of them attaining the highest score possible.

Focused Knowledge:

This project has certainly taught us a lot about game development, as well as better ways to work as a team. From learning all about C++ in our daily classes, the literature we studied out of college hours, as well as utilising SDL and just making a game in general has taught us a lot about game development. In addition, we also set out to create a great video game, and we think that we absolutely achieved this goal.

7 References

Stackoverflow.com. (2018). Stack Overflow - Where Developers Learn, Share, & Build Careers. [online] Available at: https://stackoverflow.com/ [Accessed 9 Feb. 2018].

YouTube. (2018). SDL tutorial 0 - install SDL. [online] Available at: https://www.youtube.com/watch?v=Lb_Jy5HGMsk&list=PL949B30C9A609DEE8 [Accessed 9 Feb. 2018].

YouTube. (2018). Making A Game #1: Making The Game Loop : C++ And SDL2 Tutorial. [online] Available at: https://www.youtube.com/watch?v=44tO977slsU [Accessed 9 Feb. 2018].

Dawson, M. (2010). Beginning C++ through game programming.

Mitchell, S. (2013). SDL Game Development. Birmingham: Packt Publishing.

Stroustrup, B. (2008). Programming principles and practices using C++.

Martin, R. and Han, L. (2012). Clean code. Beijing: Publishing House of Electronics Industry.

7.1 Alex McAllister Reflection

Introduction

In this reflective report I will detail my active role in the team over the year, what I contributed overall, what I learned from the experience as well as any issues we faced together.

Working as part of a team, a cohesive whole that strives to create something together, that is where I believe I work best and is the type of environment I enjoy working in most. In addition to this, the team I was a part of are all great friends of mine, and we have proven our team dynamic before when, the year previously, we worked as a four-man team and claimed the Game Design award at Games Fleadh 2017. So, when we were allowed to work again as a four-man team despite it being unconventional, and when the theme of the game was announced to be “Shoot Em Up”, a broad category that we all enjoyed, it felt like nothing could stop us from making the very best game in the time we had. And although there may have been a couple hurdles here and there, not to spoil the conclusion of this personal reflection, but I am very proud with what we achieved as a team.

Research & Design

Upon finding out the theme of the game we immediately began brainstorming ideas, although by almost pure coincidence we all had a very similar vision of what we wanted to create; a specific genre of shoot em up, a top down tin stick shooter with rogue-like elements. This is a genre of game we all enjoy and creating one ourselves would be interesting to see if we could come even a little close to matching some of our favourite titles.

The biggest challenge at this stage however was justifying a four-man team. Three people is usually the maximum per team, and we had to prove that our concept was worthy of an additional team member. After brainstorming coming up with numerous concepts, Luke had the idea of adding some kind of dynamic difficulty, which is something that we hadn’t seen in titles of the genre before. After presenting a concept and getting the greenlight on the team size, we set to work researching the foundation of the game.

I proposed a story and aesthetic; However I got a little bit of pushback from some members of the team. I was to be the designated artist for the project, so I had to back up my ideas with designs. After showing the team some rough sketches of enemy and Player designs, as well as the colour palette and possible ideas for enemy AI, the dissenting voices were soon on board entirely.

The programming language of choice was already decided; we were going to be studying and learning about C++ in 3rd year regardless, so we all agreed that it was the wisest choice. We examined the literature suggested to us by our lecturer in game programming, Liam Noonan, and found some external materials as well. We researched the best way to implement dynamic difficulty into the game and decided that it would be best to use a SQL database to store the variables that would be changing with the difficulty and update the game from there.

I learned that having a solid plan and vision from the start is important. Even if certain areas need to be compromised or cut from the finished product, having a goal to strive towards and keep the team focused maintains unity in the team, and every step towards that goal is a boost of morale

With a plan of action in mind we went into the development phase.

Development

In the first couple weeks of game programming we were acclimating ourselves with programming in C++, so we were creating a base foundation for us to work with. It wasn't really until we started learning about SDL and were given the code for "Alien Attack". After this point we created the basics of the game very quickly- taking the code and wrangling it, moulding it into what we wanted. The code was daunting, but once we started isolating parts of it and altering it we began to have a greater understanding of how these puzzle pieces fit together. Everyone was working hard to get a basic skeleton of a game before the Christmas holidays. We regularly communicated what we were working on with one another, which was helped by the fact that my three teammates all lived together and we all communicated regularly over social media on top of that.

Unfortunately, it was during this phase that I began to let my team down. After completing my initial sprints, real life events began to seep into my work and greatly affected my schedule. I was moving house during this time, and because of the stress and time investment caused by this I was lagging behind on not only my schoolwork but also my project sprints. My team members were either forced to work without the features I was meant to add or cover for me and do my sprint for me, and this wasn't fair on them at all.

Over the Christmas break, barely any work was done on the game. Perhaps my lack of contribution up to this point was causing burnout among the team, or our heavy workload was creating a backlog of assignments that needed seeing to; I couldn't say. I was still behind on my assignments, and I could barely find any time to work on the project myself.

One important thing that I learned in this phase was this; don't move house during a semester of college when a project is due. But on a more serious note I learned that it's important to stick to a schedule once the plan is made. Because of the lack of content created during this period we missed out an opportunity to add a lot of features that could have made the game better.

During this phase I was approached by the project supervisors who informed me that my team was unhappy with my contributions to the project thus far, which I completely agreed with as I knew that I was letting my team down. In one of the meetings during that time our supervisors told us that "we reached for the moon and barely made it to orbit, but at least we tried". To me this was a much needed wake up call. I knew that we could knuckle down and turn the project around.

And it was a weekend in January were we began to do just that.

The Turnabout

We agreed that we would meet up during the weekend of the international Games Jam and lock ourselves in college, and code for the entire duration of the jam. We coded almost non-stop, in total I think we each barely had six hours of sleep for the whole three days. Since we had a solid foundation to work with, we spent this period just creating content for the game;

improving all aspects of the game from collision, Player movement, enemy behaviour, creating maps, items, and new enemy types.

I was mainly tasked with creating items and integrating them into the game, as well as creating maps for the game and a new enemy type. I managed to create a huge amount of maps in a very short amount of time, editing the map files manually in Notepad++ and creating the map layouts in Tiled. I managed to create multiple items as well and integrate them successfully into the game. I began work on a new enemy type too, however I was unable to integrate it into the game during this time. I also created new sprites and edited existing assets in the game, removing old placeholders and adding in the finished assets.

I completed more work during this time than I have done over the course of the project up until that point. My teammates expressed their appreciation for my contributions and welcomed me back without reservation.

Once the Games Jam was over, our game has improved dramatically. It went from a husk of a game, with no end goal and very little content, to an almost completed product in the space of three days. And this rapid increase in content creation would not cease, as from this point on we spent nearly every day we weren't working on other assignments working on the project; polishing and adding features all the time.

I learned that even when things go wrong and morale is low, all it takes it determination, sleepless nights and take-away to turn it around. And I think that this plan worked so well because of our drive to create the best game we could, and how we work as a team. If I knew that all we needed to do was to lock ourselves in a room together with some snacks and code for three days straight I would have suggested it a long time ago in the development stage.

Final Push

Since production was going well and we had implemented nearly all of the features we set out to have in the game, we decided that it was probably wise to focus on testing, balancing and polishing the game for the remainder of the push. I took it upon myself to do bug testing, as well as fixing up sprites for the Player and enemies in the game.

In this final stretch to reach the deadline we generally didn't encounter any hiccups. There were a couple issues with certain features, however the majority were quickly ironed out. The sizes of some of the enemies and the Player needed to be changed as well. In addition, we decided on a new algorithm to determine what levels the Player will encounter while playing the game, which, in order to work correctly, required double the amount of levels we currently had, and to have all of the levels we currently had to be organised into level of difficulty. I created many more levels, with contributions from everyone on the team as well, and rebalanced the levels and organised them by difficulty. Once these levels were added to the game and the new algorithm was added, the game finally had an end state. By all technical standards, the game was finally complete, but far from finished.

In terms of difficulty during this phase, a terrible bug was encountered. We noticed that occasionally levels would not complete when they should, and narrowed it doen to one of the enemy types, the Unos, disappearing from the map while not decrementing the counter needed to open the gate to progress. Despite hours spent on trying to solve this bug, we never found out what the cause of this was. We created a quick fix to this, by pressing a certain key we were able to manually set the counter to zero and open the gate, but this is something that we must fix before the Games Fleadh.

Overall, I learned that polishing a product can be just as difficult as creating it. And that even if something does go wrong at the final hurdle there's always a way around it.

Team Performance

I couldn't have asked for a better team. We work very well together, and we all play to our strengths and cover each other's weaknesses. We're perfectly content to work long into the night together and when issues do arise, we are direct and transparent with one another. In fact, the only fault in our team cohesion during the whole project was me and my performance at the start.

While I might have initially been as dedicated to the project as everyone else on the team, life unfortunately can take a turn at any point. I was unable to correctly handle the stress of being behind on school work as well as moving house, and recent health issues meant that my drive

to contribute to the project was next to none. But I can't thank my team enough for being patient with me and for covering for me, as well as my lecturers for informing me about my team's dissatisfaction with my performance. After the games jam weekend almost all of the tension was dissolved, and we carried on as if nothing happened.

Our team was solid, and even when there was a weakness or some tension between team members, we quickly came to a swift and fair resolution.

Conclusion

I learned a lot, about coding, work ethic, teamwork, time management, and dealing with real world events in relation to a project. I couldn't be more thankful for Luke, Nathan and Eoghain for being my teammates, as well as our project supervisors Liam and Eugene for providing help, support and direction.

7.2 Eoghain McGrath Reflection

Introduction

This is a personal reflection of the work I did on the team, and what I contributed. I'll begin by talking about the research I did before the project began, how I felt we worked as a team and how good I was as a team member. Then I will discuss the programming I did on the project, and the data driven aspects I changed. I'll talk briefly about the art I created, the sound effects I added to the game and finally I'll talk about the trailer.

Research

I could say I've been doing research on this project for years, as everything I've contributed has been a culmination of my life experiences, but that would be silly; My actual research for this project was started at the end of last year when we decided as a group that we'd work as a full man team together.

We needed to make come up with a concept for our lecturers that would blow them out of the water, so I began playing a lot of rogue-like games, analysing how they worked, what made them balanced, and what made them so addictive and fun for so many people; For me it boiled down to the randomness of the game, and the items in particular, having different builds for your character each time you played, and being able to pick and choose items that better suited your playstyle. I also found for me that the immense learning curve of the game helped add a lot to the enjoyment, being able to learn new things and be challenged.

Another form of research I did for the project was watching an immense amount of "game feel" lectures, videos and game design blogs. I watched a large number on rogue-likes themselves. All of this I feel has helped me culminate a lot of knowledge on the subject of game design, which I believe I transferred into our game.

I also began learning C++ during the summer, in advance to get a head start on the project. This definitely helped massively with my ability to work on the project.

Teamwork

Overall, I was very happy with how we worked as a team. We've all worked together in the past to make a game, so we were very aware of each other's strengths and weaknesses and were able to allocate work productively as a result.

Personally, I struggled with the workload at the start of the year, so I didn't put in as much work as some of the others, but once I got stuck into it, I learnt how the codebase worked, and I was able to do enough work that I'm happy with my contribution. If I were to do the project over, I would have attempted to work on the project in full a lot sooner, as it wasn't as difficult as I originally thought.

Programming

The first thing I did on the project in terms of coding was looking at the Player class and refactoring the shooting that was there to accommodate four directional shooting. This was very easy as I was simply adding more directions for the Player to shoot, and only had to change the angles and where the bullet would spawn from the sprite.

The next thing I worked on was changing how the Player is damaged, this involved changing the Player sprite when they were hit, adding a new sound effect for when they are hit, and making some data driven changes in the XML files of the maps.

Following on from that I added some functions to the collision manager, enemies, and Player. These were what I had hoped to use to for determining the Players position for the enemies to move towards them. The functions I added were later scrapped as I found a simpler and more effective way to do it.

Next, I added control modifications for keyboard, so you could switch between left handed and right-handed controls. This involved adding buttons to the pause state of the game that the Player could click on and they would flip the "WASD" and the arrow keys. To get this working I added two functions to the pause state, one for the left handed and one for the right handed. These functions would call on the game class instance to get two functions; The functions in question would simply set a Boolean value to true or false. In the Player class then, I called a game function to check what the controls were, and if they were true/false, it

would change the Players controls respectively. I went through a few iterations of this, but I'm happy with it currently from a functional point of view, although the way I did the Boolean checking in Player, and adding a second set of controls, was bad programming practice.

The next coding change I made on the project was fixing a bug that I had noticed. Our game was missing continuous firing. I did some bug testing for this, using break points and print lines, however the fix was a lot more simple than I expected, the bullet counter being set to zero was in the wrong place, and so the Player could not fire continuously as the check against whether they can fire or not is comparing bullet counter to the firing rate, not resetting it to zero after every shot meant a comparison could not be made.

After that I worked adding some of Alex's new animations to the project and getting the animation to run, as well as doing some game feel modifications with the Players statistics, changing the fire rate and so on until it felt more smooth and responsive.

Next, I began working on implementing a boss into the game. To do this I first removed the boss enemy from all the levels, and then changed all the levels to account for less enemies. Next, I changed the enemy class, taking some of the code from our glider glass, and putting it in enemy so that all the enemy types could use the function, thus making our enemies more generic. After that I added the boss being able to follow the Player, to do this I used the target position in game, which was the Players position being passed into game with an instance call in the Players update. Once I had the Player position, I refactored a section of code in the SDL book which made a sprite follow the mouse position. Instead of following the mouse position the boss would instead follow the Players position. The function works by setting the bosses velocity to be the Players position minus the boss's position, divided by 100. This created a boss who would zoom towards the Player very quickly. Luke later on made a change to an enemy he had created, so it would regulate the speed, and I added that into the boss. Finally, I changed the bosses firing, adding multiple bullets, and I added in my map for the boss into the game.

After that I was met with another bug to do with the Player not being able to hold down a button to fire. This took me a very long time to figure out, and took a lot of bug testing, but long story short, when the Players fire rate reached zero, it was no longer comparable to the bullet counter, as bullet counter is always greater than zero. So, I simply added a check for when the Players fire rate reached four, it would set the fire rate to four.

Next, I fixed another bug this time to do with the Player being able to go half way inside the boss, this took awhile to figure out, but the fix I came up with was changing the collision buffer, setting the value to eight meant you could no longer go inside the enemy. This took some tweaking, as it also affected everything else in the game.

After that I added a shotgun item class. This would be a different bullet pattern for the Player to use and different fire rate. I used Lukes item classes as a base to allow for the Player to pick up the item and have it disappear. I then added two functions to game, one to add a string to the Players inventory, and another to check if a certain string is in the inventory. Then I added when the Player picks up the item, an instance of game is called, and shotgun is added to the vector of strings/the inventory. Then in Player, it checks if shotgun is in the inventory, and if it is, the Players spray pattern is changed, and their fire rate is lowered considerably.

Sound Effects & Art

Alex was the main artist on the team, and I'm very happy with what he produced. It works very well in the game, and due to the large majority of it all being done by him, it's very cohesive to the eye. I added a few bullet sprites, all of the explosion sprites, and the images for my controller flip in the pause state.

For the sound effects I used a free tool called SFXR, which was created by DrPetter for use in games jams and the like. It allows for very quick creation of sound effects for small games, where time is of the essence. It works by allowing you to generate a random sound from a set of options, such as a coin sfx or a jump sfx. Then you're able to modify a tonne of different values for the sound effect until you're happy with it.

Trailer

To create the trailer, I used a free trial of Adobe Premiere Pro CC. It was my first time using the software, but it was very enjoyable to use compared to the software I used last year, and similarly to SFXR, allowed for the rapid creation of the trailer.

Conclusion

In conclusion I've really enjoyed working on this project, it's been a lot of fun, and I've learnt more about games than I had in the two years previous combined. I learnt an awful lot about good programming practices, as well as just some niche functionalities of C++. I also learnt a lot about time management, working well in a team and producing something from scratch. Producing something from scratch was very eye opening for me because writing in a high-level language allows for so much freedom, and by doing the project I've really learnt to appreciate that. In previous years I thought using engines such as Unity would be the best way for a small team to produce something, but the pros of creating something from scratch greatly outweigh the cons.

7.3 Luke O Brien Reflection

Introduction

In this report I will reflect on my contribution to the team project. I will talk about the work I completed and the difficulties I encountered. I will also cover the group's dynamics and what I learned from undertaking such a big project as part of a team.

I feel that I have learned a lot from this project I got much better at coding and problem solving but not limited to that I think that I can now work well as part of a team whereas before this project I hated the thought of team-based assignments.

Research

I started my research for this project at the start of the summer break. I did this by starting to play top down rouge like and analysing them. I would often play one with a text file open on my second monitor and as I played I would take notes on various things that I thought the game did well or thinks I didn't enjoy. After I had this list I would read over it and see how I could try and implement and improve the features that I liked and what I would change about the features that I didn't like. This help a lot when we were trying to fine tune our vision for the game that we wanted to make.

About halfway through the summer break I started learning C++. I chose to read the book that the upcoming Games Programming module was going to use. This helped a lot as by the time college rolled around, I was able to put my efforts into learning SDL which meant that we as a team could start working on the project sooner.

Planning

We were using Google Drive to coordinate our planning of the project. We had many different documents that had things like concept ideas, sprints and implementation ideas. We often met up at our house for a couple of hours and these various documents were the result. We would sit around a computer and brainstorm or discuss ideas. In my opinion this was a testament as to how well we could work together as a team and I feel like we all had equal input to the planning of the project.

Programming and Features I worked on

As Nathan had tidied the framework that we were given in class, I was able to start implementing features from the get go.

The first feature that I implemented was the health bar. It was important to me that the health bar was scalable and work no matter how much health the Player had. To do this, I created three sprites. One for the start of the health bar, one for a middle segment and one for the end. I then used a for loop that would start the health bar and depending on how much health the Player had it would add middle segments to the bar before putting the end of the health bar on it. So, for instance, if the Player had 5 health then the health bar would have 3 middle segments inside the start and end pieces of the health bar.

In the framework that we were given, the transitions between levels was very poorly coded. It had to go back to the main menu before it could go to the next level. To remedy this, I created a RoomTransitionState class by utilizing the finite game state pattern that was present in the framework. The job of this state was to act as a buffer between levels so that the previous level could be cleared from memory and the next level could be loaded.

After this I added the functionality for items in the game. I had an item that changed how the Player could shoot and one that increased the Players movement speed. Even though I spent about a fortnight working on this sprint, this functionality went unused in the end game because in the timeframe we had we couldn't figure out how to make the items collidable so that the Player could pick them up. This was partially down to the poor use of inheritance found in the framework that we were given and due to my programming ability not being up to snuff. This was later remedied by making the items inherit from the enemy class. While my code wasn't used in the main game, I was able to repurpose it for debug purposes and activate items with a key press.

After we had a couple of levels, we needed a way to make it so that the Player could only exit a level after they had defeated all the enemies on it. I tried many different ways to try and do this but, in the end, I settled for adding a value to the levels XML file called enemyCount. I then changed the LevelParser class that was used to read the levels from the XML files so that it would extract the value from the XML and store in in a variable inside the Game singleton. Then whenever an enemy was destroyed, the enemyCount would be decremented. Once the

enemyCount was equal to zero, the doors would open, and the Player would be able to leave the level. The check for if the Player was about to leave was based on whether all the enemies were killed and if the Players sprite was touching the border around the game window. This was later change to a tile that the Player stands on to leave.

To test this change, I created 4 or 5 simple levels using the Tiled software that allows me to visualize the components in the XML files. The transition between the levels worked but if you ever opened the levels in Tiled, it would delete the enemyCount value that I had in the XML sheet and it would have to be added again. I never found a solution to this problem, so we eventually revamped how the transition worked.

My next task was to make the bullets collide and get destroyed when they hit walls. I tried to implement this in a couple of ways, but I was conscious of how much of an effect on the framerate that calculating collision on all the bullets would have. Luckily, Nathan had already implemented a rather efficient algorithm for checking Player/tile collision. So, I was able to take his implementation and rewrite it so that it would work with the bullets. What the algorithm does is calculate where the bullet will be after this velocity is added to its current position and if the bullet was going to be in a tile that was occupied by a wall then it would be destroyed. As the Player bullets and the enemy bullets are different objects in the framework. I had to also implement this function for the enemy bullets. This didn't put much of a strain on the tick rate of the game and won't induce a noticeable delay until the number of bullets on the screen were nearing 100.

The day after I had committed the bullet collision changes, we found a bug where if a bullet was to go through the open doors and pass of the screen, the game would crash because of an out of range error. This happened because although the bullets get destroyed just before the go off the screen. My new collision changes meant that for a very small window the game was trying to do a calculation to see if the bullets would collide with something just offscreen and this was causing the crash. How we fixed this was by destroying the bullets a couple of pixels earlier than they were originally being destroyed.

Later, in the development phase I was able to implement an item pickup class that was collidable. This class inherited from the enemy class and was to serve as a parent class that other we could inherit from to quickly make different items. Inside the class was a virtual

function called applyItemEffect(). This was all that needed to be changed for the different items.

To accommodate the new items, I created an item room. This room would only contain the item. Because the item room was a type of level, we were able to easily slot it in the level vector at a random interval. With the framework that we were using, we couldn't figure out a way to have random elements in the XML levels, so we needed to create an item room for each new item that we made.

I created the sprites for the door and the destructible tiles. I made these in paint.net by altering the tile sprite that Alex had made. I also used Alex's Player spritesheet and added the code necessary for the game to use the right portion of the spritesheet depending on which way the Player was moving.

I wrote the code for the spider enemy(Uno). I discussed with Alex and we decided that we wanted the enemy to move quickly and erratically in short bursts along a random vector. Every time it stops it shoots a bullet towards the Player. To achieve this, I used a counter that was incremented in every update cycle. Every time that the counter got to 100, the enemy would start moving until the counter was equal to 125. At this point it would stop moving and the counter would get reset to 0. Once the counter hit 50 the enemy would shoot at the Player using the code that was implemented for the Assassin enemy. There is a bug with this enemy that we were unable to solve in time. The enemy sometimes will just disappear. After debugging Nathan found that the enemy wasn't getting destroyed or leaving the level, so we are honestly just at a loss.

I created two more states for the finite state machine. These states display the controls and the tutorial to the Player after they start the game.

I changed all of the menu states by centering and changing all of the buttons. I also created titles and images for the menu states.

I added a printout for the score in the game over state by using SDL_text.

I created another 4 levels after we had the groundwork done just to add more content to the game before the due date. I also add lots of comments to the code on the final day before the upload.

The sprint that I am most proud of was my controller functionality. After weeks of research and not finding an answer that worked for our game, I came up with an algorithm that would allow for fully circler movement when using the joysticks with the controller. How it works is:

The joystick direction and force are decided by X and Y values between -32000 and 32000. How I went about getting circler movement was by first making sure both X and Y values were positive. I did this by using if statements to multiply any negative values by -1 and I would then reset them before I returned the Players velocity. I would then add the X and Y controller values together to get the full axis value.

Now that I have the full axis, I can divide the X axis value by the full axis to get the percentage of the full axis that is X. I do this for the Y value as well.

I can then take these percentages and map them to the Players movement speed. This will give me X and Y velocities that will add up to the Players movement speed, so regardless of what direction the Player moves in, he should always move at the same speed and at any possible angle.

Conclusion

Over the course of this year I have learned a lot. I have become much better at working in a team. The planning involved with such a large project will be valuable experience moving forward. I have learned how effective the Agile software development methodology is for creating games. I was incredibly useful to be able to break the project down into weekly deliverables that we could hand up. Using Agile gave us lots of flexibility during the development cycle and we were able to shuffle our sprints around if a member of the team started to fall behind.

I feel that Alex, Eoghain, Nathan and myself work well as a team. We didn't have any major conflicts and when we did disagree, we were able to easily resolve the issue and compromise. I am glad that we had Liam and Eugene as our project supervisors they were able to offer support and direction to the project on a weekly basis. Overall this project has been a great learning experience.

7.4 Nathan Dunne Reflection

The year in brief.

Previous to the beginning of the year I began studying C++ and SDL intensively in preparation for the Game Programming Module and our Project. I felt that I had a great deal of the fundamentals of programming nailed down in 1st and 2nd year so making the leap to C++ wasn't as difficult as it would have been otherwise. I'm very happy that I spent the time doing this as it came to benefit me greatly during my time working on the Project.

With regards to the year itself, everything was changed in terms of the structure and timing of deliverable modules as they were compacted together and then spread out over a much larger timeline than before. I felt like this impacted me in a way where even though I had the preparations in mind and I knew what was coming the conditioning from previous years timelines meant that a sense of false security was created.

During the Christmas break from previous years all College work, for me, grinded to a halt for a few weeks. That was the natural order of my work ethic. Over the Christmas break during this year all College work, once again, grinded to a halt for a few weeks. This continuation of inactivity began snowballing as time went on and on, causing productivity to become inert completely.

Yes, this reads more like an excuse than anything else. In my mind that's just how it felt.

Codebase.

I was excited and enjoyed working with the codebase that was given to us as a foundation. It was a breath of fresh air to be able to work with something concrete for a large project rather than our own work from scratch. The base game of Alien Attack was okay to work with but needed significant improvements to allow multiple levels and this was a core goal for us, duly achieved.

I appreciated the exposure to diverse types of systems, programming patterns and paradigms. It was also fun to learn about differing coding styles from the conventions that were in place.

The learning curve of taking in every aspect of the system and figuring out how it all worked was huge in comparison to anything we have tackled before. Learning how to take a complex problem like the project, break it up into smaller parts, solve those parts and put it all back together again felt great throughout. I feel I've learned so much with regards to the art of taking someone else's code, refactoring it and expanding upon it to make so much more of the scope of its capabilities.

The contributions that I made.

My contributions to this project include:

- Initial setup and configuration.
- Refactoring the game from scrolling to a static screen.
- Creation, installation and configuration of the database.
- Collision improvements.
- Dynamic Difficulty and level order structure.
- Refactoring of the Controller Library.
- Doubling the resolution of the game window.
- Refactoring the Tile Layer to 64x64 from 32x32.
- Extensive Memory Management and debugging.
- GameObject Attributes implementation.
- Implementation of SDL_Text.
- Game quantifiable measure, the scoring system.
- Level design for the Easy levels.
- Functionality for generic level exit placement.
- The heads-up display.
- Modular bullet damage.
- General system architecture guidance.
- Fair contributions to all submitted documentation, presentations, proposals etc.
- Destructible terrain

The most enjoyable work that I contributed was the dynamic difficulty. I had a great deal of responsibility in terms of game balance and I had the opportunity to put myself in the shoes of Players with a wide variety of experience playing video games.

The most challenging work was the collision, researching and trying out so many different implementations was fun but often the issues that arose were hard to pin down and allocate to parts of the functionality. I am proud to say that in the end it was handled quite well within the game.

The most frustrating work was the Database implementation and all the technical difficulties that came with it. You would think there would be extensive guides to integrating MySQL into C++ but the only real help I found was the official MySQL Connector/C++ Developer Guide on the MySQL website. I tried other methods, but they all fell short very soon. These difficulties included the afterthought and regret of a massive dependency being introduced to the project. “You can play our game sure, but you need to install a local instance of MySQL and create a new schema first” feels like it’s just a massive burden to the Game as a whole.

In the end to only have a single table, it all felt like more of a large individual learning exercise than any real contribution to the project.

The Team.

Like all teams, we had our fair share of difficulties. I compounded these difficulties in a large way by own near-sightedness in wanting to improve upon something or have it to an ambitious standard that was done by others and just going at it without consulting them. Essentially, I stepped on toes

Another less defined difficulty for us was the fact that 3 of the 4 team members live together and do not often leave the house to sleep and live elsewhere. This creates an environment where it can be tough to find time to wind down and chill out after an argument with another team member or just to get a break from engaging with the team altogether for a bit.

Regardless, I feel we have succeeded greatly during this project as a team. We have all learned so much about Game Design, programming, general computing and if not anything else, a great deal about ourselves.

I'd like to thank the team for their dedication to this project, and including in part, but definitely not limited to:

Eoghain:

For managing a large extracurricular project, without funding from the student's union as was before. If there was "a place the game was made" I felt it was Games Jam. The Games Jam needed to happen so Eoghain took it on and nailed it. It essentially forced us to work straight for 48 hours and it made the world of difference to the success of the project being there in that environment.

Alex:

For bouncing back into action when his chips were down. This is a 4-person team and has been for a long time. We needed everybody to accomplish our goal with this project. I was worried that Alex would fall off the face of the Earth and miss out on the experience of creating the game as a team once more. Happy relief.

Luke:

For his endless supply of humour, positivity and patience.