Usage of Random Number Generators and Artificial Intelligence to improve replayability of a Roguelike Game

Author: Oscar Witney

Date: 21st March 2019

1. 0 Table of Contents

2.0 Abstract

Replayability of games is always a concern with developers as it affects the lifespan of a game. Roguelikes are a genre of game in which that is even more true than normal. How developers in the roguelike genre increase replayability comes down to the usage of Artificial Intelligence as well as Procedural Generation with Random Number Algorithms. This paper goes over the premise of how such components can be implemented into a game environment. Addressing what components can be influenced and adjusted by AI and RNG throughout the development of a non-commercial game for the intent of research. Covered is what elements of the game include AI and/or RNG as well as how it has been implemented. For this paper in particular, the area of procedural generation RNG as well as the implementation of a Director AI are the key areas investigated.

3.0 Introduction:

3.1 Report Question:

What ways can Random Number Generation In combination with Artificial Intelligence be applied to game elements to allow an improvement in a Roguelike games replayability and to what effect can it do so?

3.2 Project Motivation:

The concern when it comes to games is the lifespan of a game, or how long players will continue to play said game. Ideal developers should strive for long lifespans of a game. Online or multiplayer games tend to not suffer as much from this concern. However, single player games are often limited based on the replay value. Games that make the user come back and play them over and over again are the goal. The roguelike genre is a genre that takes this to heart in what is intended. By providing a different playthrough of a game each time while still maintaining a base rule set or gameplay elements. The purpose of this report is to look more into how games can be designed with Random Numbers and Artificial Intelligence so as to provide a more replayable experience of the game, such as in roguelike games. While also designing the game engine from the ground up to serve as an example of potential implementation. As such, the project can be useful to those intended to develop some form of game or more specifically those intending to look into the usage of AI and RNG with a games content.

3.3 Objectives:

- Develop a Game that falls into the Roguelike Game Genre.
- Determine what Gameplay elements can be combined with either Artificial Intelligence of Random Number Generators to improve replayability.
- Record to what degree the implementation of AI or RNG had on the gameplay elements.

4.0 Literature Review:

4.1 Roguelike Genre:

The roguelike genre is a area of interest for the project as the form of game being developed for the project will fall under the roguelike genre. The reason the genre was picked is that is has qualities which allow for the greatest implementation of AI and Random Numbers into the actual gameplay elements. Other genres of games do facilitate the inclusion of AI and Random Numbers but primarily in different ways that are not as relevant, if at all, towards the goal of the project.

4.1.1 Roguelike Game Genre:

The roguelike game genre is a genre of gaming that came into light near 1993 (Solovay, Andrew, 1993) and was first used to collectively categorize a selection of games near the time as they did not fall under other genres as the time. Roguelike eventually became a well established and accepted genre with aspects stemming from the original collection of games that caused its creation.

For a game to be considered a roguelike game, it ideally follows the ‘Berlin Interpretation’ which was created at the International Roguelike Development Conference 2008 held in Berlin, Germany (Lait, Jeff, 2008). The Berlin Interpretation is a list of qualities to be expected from a roguelike game but is not required. A roguelike game would still commonly incorporate a large number of the qualities proposed but not including some does not discount the game from the genre. Some of the qualities proposed include the following.

- *Randomly Generated dungeons/environments/worlds/etc to provide replayability of the game.*
- *Permadeath Gameplay, meaning that the player is given only 1 life and should they die, they must try again with a newly generated dungeon/world/etc.*
- *Turn-Based as to allow players time to think, strategize, plan, before committing to a move.*
- *Non-Modal, meaning that every action a player can take, should be available regardless of location.*
- *There is a level of complexity contained within the game due to various systems in place. This could include allowing multiple methods to achieve a single goal.*
- *Resource Management is the key to survival. Players should be aware of their limited supplies and resources and act accordingly.*
- *Hack-n-Slash. The game is focused on the killing of large quantities of enemies. Usually, no other forms of progressing are given.*
- *Exploration. The player is given a space to explore and doing so can provide new information, items, etc, to aid their further playtime.*

  (Lait, Jeff, 2008)

Those listed are the main qualities proposed by the Berlin Interpretation, there are a number of minor qualities but are either more commonplace in today's games or outdated due to advancements of games.

All these qualities listed are expected when it comes to roguelikes and that still holds true today. The key qualities that are common through every roguelike, old and modern, is the existence of a randomly generated world, permadeath gameplay, resource management, and hack-n-slash. Examples of roguelikes include games like Dead Cells (Motion Twin 2018) and Rogue Legacy (Cellar Door Games 2013). Both games where the intent is to explore and slash your way through various dungeons to reach the endpoint, picking up new items and weapons along the way to improve yourself. These 2 games perhaps represent roguelikes at their core. Of course, roguelikes can deviate from this and include other qualities. Slay the Spire (Mega Crit Games 2019) is an example which takes advantage of turn-based aspects. It is a card-based game where you work your way through various instances of combat, special events, shops, etc, to reach the end. Along the way, finding new cards to add to your deck to provide you more power, or even remove bad cards from your deck. There are many levels of complexity even within Slay the Spire which provides immense replayability through the existence of strategy, ways to build your deck, and the various items given to you that provide other benefits.

4.2 Random Number Generator:

For random number generators, they serve as the backbone for the project. The reason a random number generator is to be included into the project is that it will be the base input for the algorithms that will generate and allocate the gameplay elements accordingly. By having a base input, the game will be able to produce the same game with the same input. The existence of the random number generator will allow the system to obtain multiple iterations of numbers from the initial input which can all be used when it comes to the AI and Algorithms behind the creation of gameplay elements.

4.2.1 Ankur, Divyanjali, and Pareek Algorithm (2014)

The first algorithm observed for the project is the algorithm proposed by Ankur, Divyanjali, and Pareek (2014) depicted works on the summation of numbers

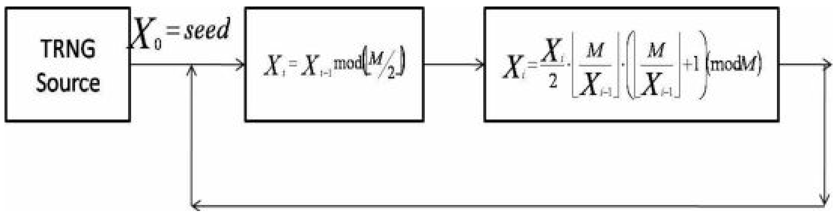

TRNG Source
$X_0 = seed$
$X_i = X_{i-1} \bmod (M/2)$
$X_i = \frac{X_i}{2} \cdot \left\lfloor \frac{M}{X_{i-1}} \right\rfloor \cdot \left(\left\lfloor \frac{M}{X_{i-1}} \right\rfloor + 1 \right) (\bmod M)$

Figure 1: *Algorithm Visual Representation (Ankur, Divyanjali, Pareek, 2014)*

from Z_M, chosen as multiples of a number from the previous iteration and mapped again to Z_M (Ankur, Divyanjali, Pareek, 2014). The proposed algorithm consists of two inputs, one ideally being from a TRNG source to allow a true-random sequence to a degree. The seed or X_0 ideally comes from a TRNG source while the *M* used within the calculations is not explicitly stated to come from anywhere in particular by

$$M = 10007, X_0 = 121,$$
$$X_1 \equiv X_0 \cdot \sum_{i=1}^{\lfloor M/X_0 \rfloor} i(mod M) \equiv 1476$$
$$X_2 \equiv X_1 \cdot \sum_{i=1}^{\lfloor M/X_1 \rfloor} i(mod\, M) \equiv 975$$
$$X_3 \equiv X_2 \cdot \sum_{i=1}^{\lfloor M/X_2 \rfloor} i(mod\, M) \equiv 3590$$
$$X_4 \equiv X_3 \cdot \sum_{i=1}^{\lfloor M/X_3 \rfloor} i(mod\, M) \equiv 736$$
$$X_5 \equiv X_4 \cdot \sum_{i=1}^{\lfloor M/X_4 \rfloor} i(mod\, M) \equiv 6394$$
$$X_6 \equiv X_5 \cdot \sum_{i=1}^{\lfloor M/X_5 \rfloor} i(mod\, M) \equiv 8927$$
$$X_7 \equiv X_6 \cdot \sum_{i=1}^{\lfloor M/X_6 \rfloor} i(mod\, M) \equiv 1765$$
$$X_8 \equiv X_7 \cdot \sum_{i=1}^{\lfloor M/X_7 \rfloor} i(mod\, M) \equiv 6461$$

Figure 2: *Algorithm Example Run (Ankur, Divyanjali, Pareek, 2014)*

the authors. Though through observation and simple run-throughs of the algorithm, the *M* was determined to be the sequence ceiling. Through testing done by the authors as featured in the paper, they have determined the quality of the algorithm presented based on the randomness of the output. A variety of tests have been used within the paper including The Runs Test, Tests for the Longest-Run-of-Ones in a Block, The Approximate Entropy Test, as well as many more in order to provide validity behind the algorithms quality. An example of the sequence is even provided to show the algorithm functioning. The great deal of tests done does show the algorithms value as well as its degree of randomness.

4.2.2 Lehmer/Park Miller Algorithm (1988)

Continuing on from the first paper and onto an algorithm discussed in the paper by Stephen K. Park; Keith W. Miller (1988) and featured in the paper by J. Lan,

$$X_{k+1} = g \cdot X_k \mod n$$

Figure 3: *Lehmer Algorithm Representation (J. Lan, W. L. Goh, Z. H. Kong, and K. S. Yeo, 2010)*

W. L. Goh, Z. H. Kong, and K. S. Yeo, (2010). The Lehmer or Park-Miller algorithm is a well established and widely used RNG. By looking at a proper RNG, it will help provide a benchmark to compare algorithms similar to the first algorithm reviewed, under the assumption they work. The Lehmer algorithm was chosen as the benchmark due to its simplicity, usability, and reputation. Displayed is the algorithm as represented by the authors in the paper as well as the algorithm in its general formula. Just like the first algorithm, the Lehmer algorithm is PRNG rather than a TRNG, but can still obtain the seed from a true source if desired. Also similar to the first algorithm is the existence of two inputs. The variables in question are the *seed*,

```
const
  a = 16807;
  m = 2147483647;
  q = 127773;  (* m div a *)
  r = 2836;    (* m mod a *)
var
  lo, hi, test : integer;
begin
  hi := seed div q;
  lo := seed mod q;
  test := a * lo − r * hi;
  if test > 0 then
    seed := test
  else
    seed := test + m;
  Random := seed / m
end;
```

Figure 4: *Lehmer Algorithm PseudoCode (Stephen K. Park; Keith W. Miller, 1988)*

and *A*, from the displayed pseudocode. Working along the same lines as the Ankur, Divyanjali, and Pareek (2014) algorithm, the seed is the more important of the two variables. But unlike the first algorithm, the *A* variable does not impose any innate limitations on values. It does not serve as a ceiling or a floor on values and does not hinder or limit the inputs of the algorithm. There are in fact no innate limitations to what the inputs have to be for the Lehmer algorithm. This alone would likely place it above the earlier algorithm in terms of usefulness, assuming the earlier algorithm functioned properly. A basic rundown of the algorithm is as follows…

1. *Initialize the input seed and parameters,*
2. *Compute the value of hi = (seed div q) and lo = (seed mod q).*
3. *Then compute the corresponding test value.*
4. *Save the new seed value. If test > 0, save test as new seed, otherwise save test + m.*
5. *Output the new seed.*
6. *Iterate, and let the output seed be the new input seed.*

(J. Lan, W. L. Goh, Z. H. Kong, K. S. Yeo 2010)

4.2.3 L. Milinković, M. Antić, and Z. Čiča (2011):

With the Lehmer algorithm providing a solid baseline of what is to be expected from other algorithms due to its influence in this field, the next algorithm as proposed

$$x_i = \left(\frac{(x_{i-1} + m) \cdot K_1 \cdot (M - i)}{K_2} \right) \bmod m,$$

Figure 5: *Algorithm Mathematical Representation (L. Milinković, M. Antić, and Z. Čiča, 2011)*

within the paper by L. Milinković, M. Antić, and Z. Čiča (2011) will have to compare against the Lehmer algorithm. The algorithm in question shown here is based on the idea of irrational numbers and as a result, no two values produced in the sequence would be equivalent, The entire proof behind this theory is written out and explained within the paper. Due to the length of the proof and complexity, it will not be brought up directly. Regardless, this algorithm is yet another PRNG, as all the algorithms investigated will be. Like the previous algorithms, this one takes in a number of inputs which can be obtained from a true source. However, completely unlike the previous algorithms observed, there are a total of five input variables towards this algorithm.

The values m, M, K_1, and K_2 are the parameters of the generator. The parameter m is the positive integer, m $\in$ N\{0} (where N is the set of natural numbers). The parameters K_1, K_2 $\in$ Q\Z, where Z is the set of integers, and parameter M is defined as M = N + p, where p $\in$ N\{0}. (L. Milinković, M. Antić, and Z. Čiča, 2011)

The fifth unmentioned variable is X_o or the seed value. Immediately observed from this alone are the limitations of input values but these qualities are to be expected as the algorithm relies on the usage of irrational numbers. K_1 and K_2 are rational numbers, *m* is Natural number, and M is as specified at the end of the statement. One quality not directly mentioned is the effect of *m*. *m* acts as both the ceiling and the floor of the output sequence, the ceiling being +*m* and the floor being -*m*. This quality of the sequence floor being negative is something not present in either of the earlier algorithms. Comparing to the first algorithm, this would essentially double the potential values should the same ceiling be used in each algorithm. The Lehmer algorithm does include a ceiling but it is often times classified as 2^{32}-1 which is the maximum value of an int variable in most languages. Though should 2^{32}-1 be used as the value for *m* in the algorithm by L. Milinković, M. Antić, and Z. Čiča (2011), it would indeed effectively double the values possible in the output sequence. While this is potentially a good quality for the algorithm to have, this is only true should there be a desire or use for negative values in the output sequence. If this is not the case and you instead take the absolute of each value to only obtain positive results, you no longer effectively double the potential results and instead keep the same potential results as another algorithm with the same ceiling. There is also a potential side effect of doing so depending on when you grab the absolute. You either cause no notable issues or you potential create a duplicate of every value should enough iterations be done. The point of this algorithm is the existence of no repeating values. However, for every positive value, a negative exists and should you absolute each value, you turn the negative into positive and have a double value, therefore not matching the desired effect of the algorithm. Of course, this can be avoided by doing the math correctly, but it is still an effect to be aware of.

4.2.4 Comparison and Analysis:

WIth the premise of the algorithms covered, the next appropriate step is to compare and analyze to determine what algorithm is most useful for the project. How such a task was done involved taking the information provided by the papers about the algorithm combined with testing done in a test harness to obtain information about the algorithm. The specific information being compared here is the average runtime of the algorithm over X runs. The specifics of this is the average runtime of one hundred runs of the algorithm being run for fifty iterations using a single set of inputs. This process was repeated using a range of inputs to obtain numerous runtimes as well as a range of different outputs to compare.

For the initial stage, shown is the number of inputs, runtime, as well as the output range of the algorithms observed. Specifics on the outputs covered later.

Algorithm	**Inputs**	**Avg. Runtime**	**Output Range (X)**
2.1.1	2	0.0019s	`X > 0 && X < K` (K = input Variable)
2.1.2	2	0.0324s	`X > 0 && X < ` 2^{32}`-1`
2.1.3	5	0.0023s	`X > -m && X < m` (m = input variable)

Table 1: *Algorithm Comparisons (Oscar W.)*

The initial response to the data shown is that algorithm 2.1.2, the Lehmer algorithm, is considerably slower than the other algorithms observed. Comparing algorithms 2.1.1 and 2.1.2 assuming both have the same output range (K = 2^{32}-1) expresses this concern of speed. Both algorithms require the same amount of inputs as well as produce the same output range (in this scenario) but 2.1.2 takes upwards of seventeen times more time to run through the fifty runs within the test harness. If the third algorithm is then taken into consideration, 2.1.3, it boasts similar runtime to 2.1.1 as well as double the potential output range (assuming K = m) with the downside of requiring up to five potential inputs. Assuming no requirements or limitations as pertaining to the software using the algorithm, 2.1.3 seems to be the most suitable with its fast runtime as well as large output range. This would indeed be the case. However, within the scope of this project, there are aspects of algorithm 2.1.3 which are not suited for what is desired. Most notably is the existence of five inputs. The premise of the project is to make use of an RNG algorithm in combination with AI as a Designer to procedurally generate content of a game. five Inputs are far more than is required as this provides a far greater range of potential sequences than is needed. A Similar concern arises from the output of algorithm 2.1.3. While It is indeed true that the algorithm provides double the potential outputs of the other algorithms with its potential for negative values, the existence of negatives is not necessarily good quality. Depending on the implementation of the AI within the project, negative numbers may or may not be suited for it without adding in functionality merely to convert the negative to positive. While considering the implications of the high inputs and potential negative outputs of algorithm 2.1.3, it can be deemed that this algorithm, while highly effective at its job, is not suited for the scope of the project. Essentially being overqualified for what needs to be done. Should the scope of the project have involved cryptography or similar fields of information, then algorithm 2.1.3 would have been highly suited for the task and most likely used.

Leaving algorithms 2.1.1 and 2.1.2 left as which is more suited in regards to the project. If it is assumed that K = 2^{32}-1 for the sake of keeping the ranges the same, Algorithm 2.1.1 only has one input while Algorithm 2.1.2 requires two. There is also the large difference in runtime also to consider with the algorithms. One last piece of data that will allow a comparison between the algorithms is the quality of the output. Using some example inputs, the output sequence can be observed and

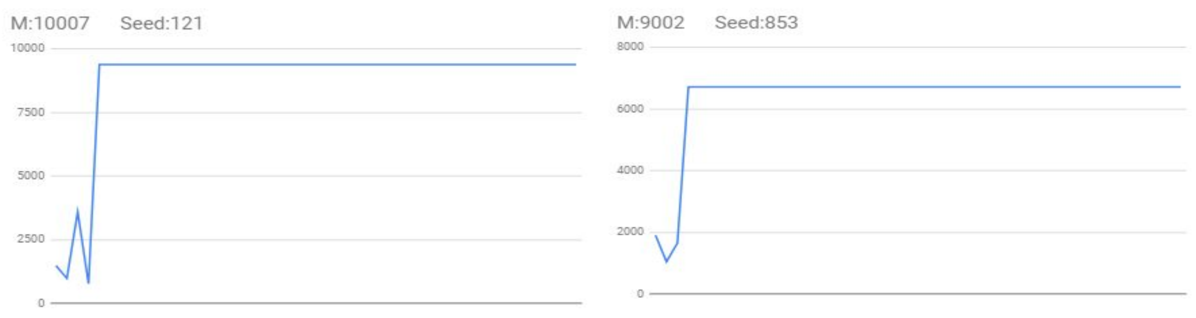


Figure 6: *Ankur, Divyanjali, and Pareek Algorithm Outputs (Oscar W.)*

further comments can be made. Starting with algorithm 2.1.1, two of the output sequences with the given inputs are shown below. Immediately there is cause for concern with the sequences. There are a total of fifty values in each of the sequences and it is observed that a large percentage of said values are the same. Considering the example provided by the authors is tested and shown here, there is cause for concern in regards to the validity of the algorithm proposed. Though this is not the place to be discussing the validity of an author's paper. For the sake of comparison, it will be limited to assuming the algorithm does not function as intended and as such, is not suited for the project. With this, algorithm 2.1.1 has been deemed invalid due to not functioning correctly, leaving only Algorithm 2.1.2 left. Example output sequences have been provided below to observe whether or not it functions as intended.

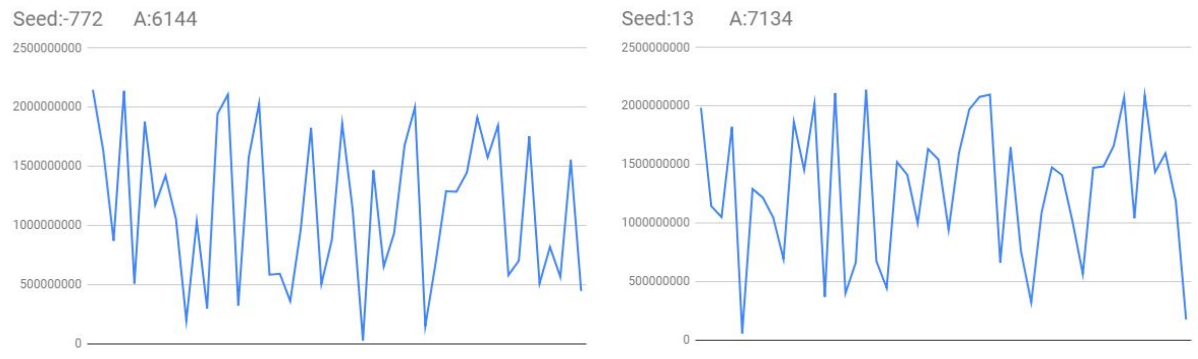


Figure 7: *Lehmer/Park Miller Algorithm Output (Oscar W.)*

As observed, the algorithm does indeed function as described and it can be determined that algorithm 2.1.2 is the most suited for the scope of the project compared to the other algorithms observed. The most glaring concern with this algorithm was its runtime relative to the other algorithms but it should be noted that it was the runtime of fifty consecutive iterations of the algorithm. And even then, the algorithm still runs at a fraction of a second. If the scope of the project required the fastest possible runtime available, then further research would be done to determine the most suited algorithm. However, with the scope of the project how it is, such level of quality control is not immediately required. The results being that algorithm 2.1.2 is deemed the most suited algorithm researched.

4.3 Artificial Intelligence:

For the Artificial intelligence and its importance to the project, the inclusion of AI is required for two components of the project. The first being the AI for intractable entities within the game so they appear as alive, even if not lifelike. The other is AI surrounding the creation of the game world and its gameplay elements. By using AI in said areas, improvements to the games generation and playability will be added.

4.3.1 Artificial Intelligence as a Designer:

Moving onto a different area to consider in regards to the project, Artificial Intelligence, more specifically, AI as it pertains to games. For the scope of the project being developed, the AI will primarily consist of one primary form of AI with a minor focus on a secondary form. The first of the AI's brought up refers to AI connected to Procedural Generation of content or AI as a Designer (M. O. Riedl and A. Zook, 2013).

The second role of Game AI is to mediate between the human designer (or developer) and the human-computer system comprised of game and player. In our metaphor, game designers are responsible for building and defining a game, analyzing how players interact with the game, and iteratively refining a game to achieve a design vision. This paradigm for artificial intelligence is often referred to as Procedural Content Generation(PCG)—algorithms and representations for generating any and all components of games. (M. O. Riedl and A. Zook, 2013)

Game adaptation combines content generation and player modeling to enable AI designers to tailor games to individual players, (M. O. Riedl and A. Zook, 2013)

As described by the paper, this form of AI works with the algorithms behind the creation and generation of world elements and combines them with statistical player feedback in order to adapt the game to the player's skills. Essentially allowing the creation of a more personalized game/playthrough. The common reasons as to why one would implement this form of ai would be automated adjustment of game content based on design goals for player behavior, player skill acquisition, or maximizing player enjoyment. (M. O. Riedl and A. Zook, 2013). While the authors fail to delve into further detail regarding this topic, this is understandable as it was not the major focus of their paper. However, what was covered provided excellent reasoning and understanding of this form of AI as to why it would be implemented and as well as what the AI should strive to achieve. It would have aided the paper further if an example or two of said AI being used in practice to provide a reference to a working example. Though AI as a Designer is constrained around the algorithms of procedural generation it is tied to, making it harder to have one AI adapted for multiple games unless the games implement similar or the same procedural generation.

Considering the paper as a whole, the authors included a solid range of information regarding AI and its potential applications in regards to games. Together, the paper provides a good basis for the forms of AI to be considered and potential developed within the scope of the project but lacks on some of the additional information regarding Ai in this field. A second paper researched and reviewed here covers some of the aspects the authors of the first paper do not cover and as such, the second paper to be discussed will help fill in the gaps.

Levels have been generated for various game genres such as dungeon-crawlers , horror games , space-shooters , first-person shooters , and platformers. (M. O. Riedl and A. Zook, 2013)

The second paper talks about AI as a Designer though it is not explicitly referred to as such. One of the primary aspects covered by the second paper is the notion that “general purpose search and learning algorithms by far outperform more domain-specific solutions and “clever hacks” (J. Togelius and G. N. Yannakakis, 2016). The authors took the results and information from a number of AI-related competitions such as the General Game Playing Competition (GGP) and The General Video Game AI Competition (GVGAI). As per what the authors researched and concluded, the importance when it comes to this form of AI is the ability to capture as much information about the game world as possible in a format that allows proper usage. The authors also mentioned “that variations of Monte Carlo Tree Search perform best on GVGAI and GGP, and for The Arcade Learning Environment (where no forward model is available so learning a policy for each game is necessary) reinforcement learning with deep networks performs best” (J.

Togelius and G. N. Yannakakis, 2016). The authors of the second paper provide a good backing regarding how to develop AI for this field and provided an example form based on the results of competitions.

4.3.2 Analysis of Papers

As mentioned when discussing the contents of the two papers about AI, the first paper by M. O. Riedl and A. Zook (2013) contained a good basis regarding the topic of AI as a Designer. The paper featured a good selection of the ideas behind this form of AI but fell short when it came to more specific aspects of the AI. At this stage, the second paper by J. Togelius and G. N. Yannakakis, (2016) mentioned fills in the gaps in information not contained within the first paper. In contrast to the first paper, the second paper features less about the concepts of the AI and more on the AI itself and aspects to consider in its development. Primarily taking the idea of General Purpose AI in the field of a Designer and discussing how it is in a large range of game genres, the superior choice over AI specific to the game.

Taking both papers into account, they contribute a solid background of information regarding the AI in both what kinds of objectives it should/could achieve as well as some principles behind how to develop the AI. Each paper lacks one side of information the other has. The first paper provides the objectives and such of AI in this department while the second paper contributes the principles behind how to implement it and other designs to consider. The two papers contribute equal levels of information and produce a solid understanding of the AI to be developed.

5.0 Design:

The contents of the follow section will initially consist of some planning around components to be implemented in their most base forms such as AI. User stories will also be written and addressed to determine tasks and features to be implemented. Continuing on to break down into functional requirements and other similar components required before implementation is to be done.

5.1 Project Components:

First discussing some key components that will be designed and implemented within the project. Primarily discussing their purpose and why they are to be implemented with further detail into how in the lower sections. There are two components specifically to be brought up before going into further detail about them and their designs.

5.1.1 Procedural Generation:

Procedural generation will form the main body of the world generation components. The idea of procedural generation is the usage of algorithms to determine where and how game elements are assigned, created, and allocated. For the specific case of the project, it will be used primarily when generating the world in which the player will play in rather than individual game elements. The reason procedural generation is mentioned here is that it will be a major component for how random number generators are used as the random number generators will provide the values to be used when determining where and how elements in the world are allocated. Of course this can be applied to other game elements outside of just the world and should resources facilitate it, they will be. However, procedural generation outside the game world is a second priority to everything else.

5.1.2 Director AI:

The Director AI as it is referred to is a form of AI that controls the pace of the game as the player plays it. The name comes from the game Left 4 Dead (Valve Corporation, 2008) in which the Director was the name of the form of ai doing this exact job. What the AI did was monitor various game elements while players played the game such as players health, their supplies, how many enemies they have killed in a period of time. Using the information and data the AI observed, it determined how to handle future events in the game, such as spawning additional enemies, changing spawn locations, and other similar elements. The result was an ever changing pace and difficulty of the game according to how the players proceeded and played. This also allowed for replay value as the Director AI would make different decisions dependent on that playthrough specifically. Because of the qualities described, a form of this AI is intended to be implemented into the project, of course due to the size of the project as well as limitation of game elements, the complexity of the AI will be nowhere as how it was in Left 4 Dead. Regardless, a basic form will be implemented and will be referred to as the Director or Director AI in future sections.

5.2 Requirements:

5.2.1 User Stories:

The first step to establish requirements is to write up a range of user stories to encompass a majority if not all the major functions of the software and what is desired. From the stories written, further requirements can be determined as well as priorities regarding tasks.

5.2.1.1 Game Engine:

GEUS.1: As a Developer, entities within the game must belong to different types according to how they interact with other entities.

GEUS.2: As a Developer, the game engine must be able to calculate locations and velocities of entities within the game.

GEUS.3: As a Developer, the game engine must be able to adjust locations of entities based on their velocities.

GEUS.4: As a Developer, the game engine must be able to detect and solve collision between entities within the game.

GEUS.5: As a Developer, the game engine must be able to detect and solve combat between entities within the world.

GEUS.6: As a Developer, the game engine must be able spawn, detect, and solve items and interactions with items between entities within the world.

GEUS.7: As a Player, I want to be able to control my character within the game.

GEUS.8: As a Player, I want to be able to attack enemies within the game.

5.2.1.2 Graphics:

GUS.1: As a Developer, the game elements should be represented in a form easily translatable to visuals

GUS.2: As a Developer, the game should turn game elements into graphical representation using the SFML library

GUS.3: As a Player, I want to be able to tell the difference between me, the player, enemies, and terrain.

GUS.4: As a Player, I want to be able to have a small map to show the entire level

GUS.5: As a Player, I want to see the health of myself and enemies within the game.

5.2.1.3 World Generation:

WGUS.1: As a Developer, there must be a suitable representation of the world and its attributes.

WGUS.2: As a Developer, the world gen algorithms must be able to interact with and manipulate the representation of the world

WGUS.3: As a Developer, the world gen algorithms must be based around a Random Number Generator.

WGUS.4: As a Developer, there must be a text representation of the game world.

WGUS.5: As a Player, I want to be able to play the same game world if I use the same seed more than once.

WGUS.6: As a Player, I want to the game world to be different each time I play.

5.2.1.4 Entity AI:

EAUS.1: As a Developer, the AI should change how it acts based on parameters.

EAUS.2: As a Developer, the AI should interact with the player.

EAUS.3: As a Player, the AI should not be too aggressive and attack non-stop

5.2.1.5 Director AI:

DAUS.1: As a Developer, the AI should look at the game state and observe game values

DAUS.2: As a Developer, the AI should be able to manipulate and change game values as and when it sees fit

DAUS.3: As a Developer, the AI should adjust game values based on its observations of the other game values.
DAUS.4: As a Developer, the AI should be able to manipulate game values permanently and temporarily depending on the situation.
DAUS.5: As a Developer, the AI should change its state randomly.

5.2.2 Functional Requirements

The follow section will address the user stories defined above and pick out functional requirements from them and allocate a priority from 1-10, where 1 is the lowest possible priority and can potentially be ignored, while 10 refers to requirements that have to be implemented no questions asked.

5.2.2.1 Game Engine:

User Story	Functional Requirements	Priority (1-10)
GEUS.1	- Entities within the game must belong to defined under different types using inheritance to distinguish entity interactions - Entities must have a parent class containing a vast majority of attributes including location, velocity, etc. - Children classes should only include additional functionality according to how the entity needs to interact	**10** **10** **10**
GEUS.2 **GEUS.3**	- Velocity and Location of entities contained within the game must be available for reading and writing as well as mathematical processing. - Velocity and Location must be represented using the same data type to facilitate ease of processing amongst the two. - System should provide calculations to update	**10** **10** **10**

	location and velocity of entities based on circumstances and on a regular basis (every frame/tick)	
GEUS.4 **GEUS.5**	- Using location and velocity, collision of entities must be calculated and addressed so as to not allow overlapping of entities unless desired. - System should adjust positions of entities after detecting collision to simulate the object hitting the object and stopping. - Collisions should be differentiated based on the entity types involved within the collision so the appropriate response can be achieved.	**10** **9** **8**
GEUS.6	- The system should support a subclass of entity to act as a special interactable. - Collision involving this type of entity should result in the appropriate effects occurring - Generation of this entity should not directly be tied to world generation and be adjusted based on the Director	**3** **3** **2**
GEUS.7 **GEUS.8**	- Input must be received from the user and translated into motion of the player or other game elements. - Direction of movement must correspond to directional input form player (WASD or Arrow Keys) - Mouse location and direction from player must be tracked to be able to determine attack location when mouse clicked.	**10** **10** **9**

Table 2: *Game Engine Functional Requirements (Oscar W.)*

5.2.2.2 Graphics:

User Story	Functional Requirements	Priority (1-10)
GUS.1 **GUS.2**	- Elements of the game must be represented in numerical formats or using specific data types as per graphical libraries functions. - Elements of the game must facilitate interaction as well as modification of their representation in real time.	**9** **9**
GUS.3	- Entities within the game must have different visual representation to allow ease of differentiation of entities during play (i.e color)	**8**
GUS.4	- World representation must be done in a format that allows for duplication and modification without impacting how the world works. - Entities must be able to be visually modified independent of other attributes of the entity	**8** **9**
GUS.5	- Health of entities must be represented in a format that allows for visual representation (i.e integers) - Suitable processes must facilitate translation of integers and entity position into a visible health bar to provide visual feedback to the player.	**7** **5**

Table 3: *Graphics Functional Requirements (Oscar W.)*

5.2.2.3 World Generation:

User Story	Functional Requirements	Priority (1-10)
WGUS.1 **WGUS.2**	- The elements that represent the world must be represented with data types that allow for ease of	**9**

	access and manipulation. - World representation must be fully accessible to the algorithms involved in world generation. - Algorithms must be able to modify the world representation as and when it needs without restriction.	**9** **9**
WGUS.3	- A Random number Algorithm that has been decided beforehand must be implemented correctly. - The current iteration of the algorithm must constantly be stored while the initial input must be saved elsewhere for reference. - Each time the current iteration of the algorithm is used in a calculation of sorts, the output must be iterated equal to the number of times it was used. - The current iteration must be used through the generation algorithms at any and all points where some value must be assigned pseudo-randomly.	**10** **10** **9** **10**
WGUS.4	- Elements of the world should be represented in a format that can, through either some processing or as is, translated into text output into a terminal or text file. - Text representation of the world should contain all appropriate attributes of the world.	**7** **6**
WGUS.5 **WGUS.6**	- Generation of world must be dictated by the initial seed only and no other factors. - Any other factors that should influence the world should only occur after world generation has completed and on elements not directly tied to the	**9** **8**

	world.	

Table 4: *World Generation Functional Requirements (Oscar W.)*

5.2.2.4 Entity AI:

User Story	Functional Requirements	Priority (1-10)
EAUS.1	- The AI for entities should consist of a state machine based on its current health. - The AI should behave differently at different percentages of health such as Attacking, Following, Fleeing.	**10** **9**
EAUS.2 **EAUS.3**	- The AI's prime goal should be to reach the player in order to damage them through contact. - The AI should only attempt to interact with the player if the player is a certain distance away from the entity so as to allow the player breathing room. - Upon interaction with the Player, the AI should not be able to act for a short period of time to allow the player to react and adjust instead of the AI constantly attempting to interact with the player.	**9** **8** **6**

Table 5: *Entity AI Functional Requirements (Oscar W.)*

5.2.2.5 Director AI:

User Story	Functional Requirements	Priority (1-10)
DAUS.1 **DAUS.2** **DAUS.3**	- The director should be able to see and use various elemental values within the game. - The director should be able to modify game values it has observed according to what it has observed.	**10** **10**

	- The director should adjust the values of modifiers within the game at any point based on its decisions	**8**
DAUS.4	- Changes to values within the game must be differentiated between permanent and temporary. - The director should have methods that altar values permanently or temporarily and is able to do so independent of one another.	**9** **8**
DAUS.5	- An internal modifier on the director should exist which will modify all its further actions based on it. - The state or internal modifier of the Director will change based on a combination of a random chance and distributed probability. - The chance of the Director switching to a different state is dependant on the observable values in the game, influencing the percentage chance of each state when rolling the dice to switch.	**8** **6** **6**

Table 6: *Director AI Functional Requirements (Oscar W.)*

5.3 Risk Management

This will cover the potential issues and events that could occur which would cause a problem to the project.

#	Risk	Desc.	Probability	Severity	Actions Taken
1	Loss of Project Artefact	Misplacing or deletion of the project artefact due to some actions that occurred (corruption, deletion, etc)	Low	Major	Regular backups of the artefact will be saved to a number of different locations and formats so that a version is always available.

2	Changes to external libraries being used	Updates to external libraries the system depends on that may interfere with functionality	Low	Moderate	In the event libraries are updated, only the version suited for the system will be used assuming the updated library interferes, otherwise the updated library will be used.
3	Content within the game infringes on copyright.	Some content used within the game regardless of form is under copyright protection and not available for public use.	Moderate	Major	For the purpose of the project, any and all outside resources used will be checked and confirmed to be available for public use. Also an attempt to lower the amount of external resources used.
4	Resources of the project exceed and budget set out.	Any resources that require a monetary or other transaction in order for usage must not exceed any limits set during the project.	Low	Minor	No resources that require monetary transactions are intended and in the event resources require purchase, free alternatives will be found and used instead.

***Table 7:** Risk Management (Oscar W.)*

6.0 Implementation:

With the implementation of the components laid out, each section will be covered with a provided model representing the component with their respective methods, classes etc. Further details on components will be provided in their respective sections as well to address why and how aspects were done.

6.1 The SFML Library:

Before going into detail regarding implementation of the components, some initial information regarding the graphics library used for the project is required. Simple and Fast Multimedia Library or SFML was chosen for the project due to its simplicity while still providing all the functionality required of the project at the current scope. Alongside providing graphical components, it provided some class and variable types which are used actively throughout the code. Within the code, all such items are preceded with `sf::` which indicates it comes from the SFML namespace. One specific item to bring up is `sf::Vector2f` which is a type provided by SFML. as its name implies, it acts as a vector and contains two values, an X and a Y. Since this type allowed for storage of X and Y values in one space, it has been widely used for a number of components such as motion, location, sizes, instead of storing information for these attributes as two seperate variables, one for X and one for Y. Vector2f is used throughout the code which is why a basic understanding of what it is has been provided. Other types include `sf::Color` as well as `sf::Shape`, both of which are used primarily when it comes to the translation into graphics.

6.2 Entity Objects:

The first component address is the class hierarchy representing game objects. Due to the fact that a portion of the game engine was previously developed, the class model being implemented now is based on the previous version with additional components taken into account. As for why said components are being implemented, it will allow more accurate representation of the game world and its various elements by defining a base and then differentiating the children for different purposes.

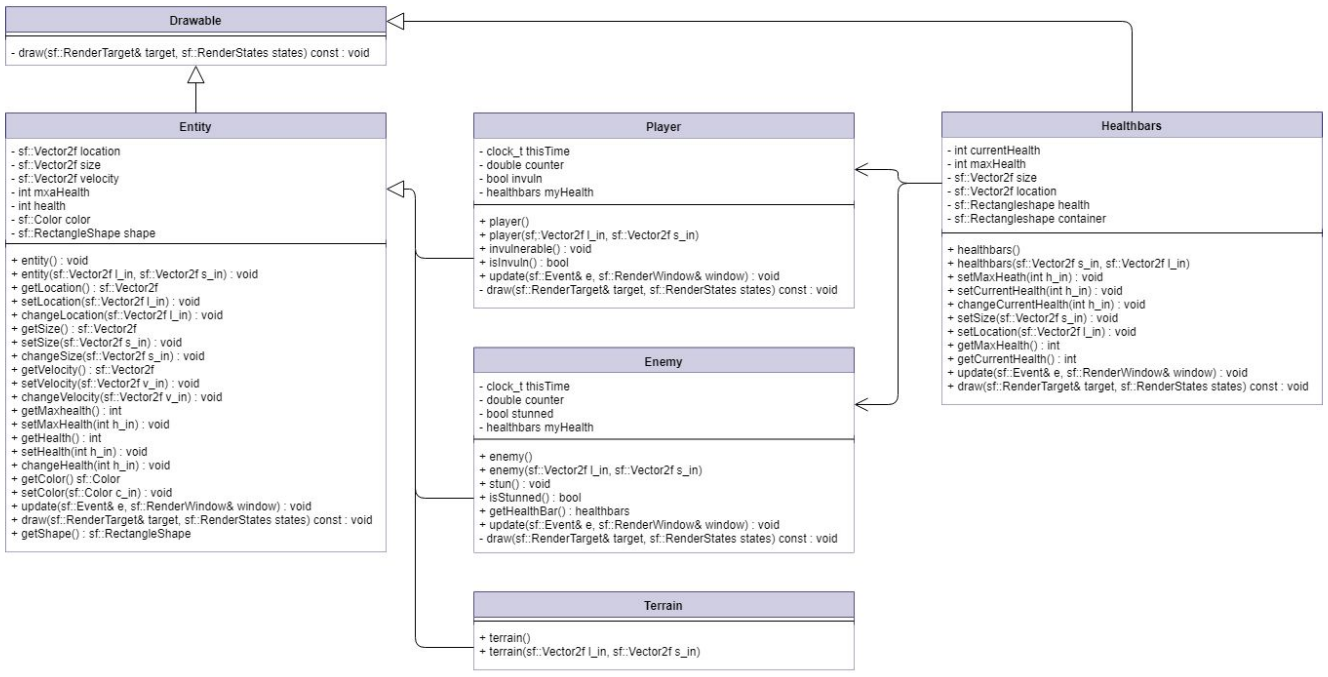

Drawable
- draw(sf::RenderTarget& target, sf::RenderStates states) const : void
Entity
- sf::Vector2f location
- sf::Vector2f size
- sf::Vector2f velocity
- int mxaHealth
- int health
- sf::Color color
- sf::RectangleShape shape
+ entity() : void
+ entity(sf::Vector2f l_in, sf::Vector2f s_in) : void
+ getLocation() : sf::Vector2f
+ setLocation(sf::Vector2f l_in) : void
+ changeLocation(sf::Vector2f l_in) : void
+ getSize() : sf::Vector2f
+ setSize(sf::Vector2f s_in) : void
+ changeSize(sf::Vector2f s_in) : void
+ getVelocity() : sf::Vector2f
+ setVelocity(sf::Vector2f v_in) : void
+ changeVelocity(sf::Vector2f v_in) : void
+ getMaxhealth() : int
+ setMaxHealth(int h_in) : void
+ getHealth() : int
+ setHealth(int h_in) : void
+ changeHealth(int h_in) : void
+ getColor() sf::Color
+ setColor(sf::Color c_in) : void
+ update(sf::Event& e, sf::RenderWindow& window) : void
+ draw(sf::RenderTarget& target, sf::RenderStates states) const : void
+ getShape() : sf::RectangleShape
Player
- clock_t thisTime
- double counter
- bool invuln
- healthbars myHealth
+ player()
+ player(sf;:Vector2f l_in, sf::Vector2f s_in)
+ invulnerable() : void
+ isInvuln() : bool
+ update(sf::Event& e, sf::RenderWindow& window) : void
- draw(sf::RenderTarget& target, sf::RenderStates states) const : void
Enemy
- clock_t thisTime
- double counter
- bool stunned
- healthbars myHealth
+ enemy()
+ enemy(sf::Vector2f l_in, sf::Vector2f s_in)
+ stun() : void
+ isStunned() : bool
+ getHealthBar() : healthbars
+ update(sf::Event& e, sf::RenderWindow& window) : void
- draw(sf::RenderTarget& target, sf::RenderStates states) const : void
Terrain
+ terrain()
+ terrain(sf::Vector2f l_in, sf::Vector2f s_in)
Healthbars
- int currentHealth
- int maxHealth
- sf::Vector2f size
- sf::Vector2f location
- sf::Rectangleshape health
- sf::Rectangleshape container
+ healthbars()
+ healthbars(sf::Vector2f s_in, sf::Vector2f l_in)
+ setMaxHeath(int h_in) : void
+ setCurrentHealth(int h_in) : void
+ changeCurrentHealth(int h_in) : void
+ setSize(sf::Vector2f s_in) : void
+ setLocation(sf::Vector2f l_in) : void
+ getMaxHealth() : int
+ getCurrentHealth() : int
+ update(sf::Event& e, sf::RenderWindow& window) : void
+ draw(sf::RenderTarget& target, sf::RenderStates states) const : void

Figure 8: *Entity Hierarchy (Oscar W.)*

As shown, the hierarchy is made up of five classes, Drawable, Entity, Player, Enemy, and Terrain. The Drawable class is a class implemented in the SFML Graphics Library being used for the project and as such, all its functionality has been implemented by others. The remaining four classes were written for the project though. Entity is the main Parent of the other classes and only extends Drawable. Entity contains the vast majority of functionality required consisting of attributes such as the location, size, velocity, as well as color and shape. The class also provides the corresponding getters and setters for the variables. Two methods to note that are contained within entity are `draw(..)` and `update(..)`. Both of these methods are a result of SFML, draw can be seen in the drawable class and it does what the name implies. It takes the components available, in this case the shape and color of the entity, and draws them on the target provided. Update also does what the name implies. It will update the various attributes of the object as well as then translating the new changes to the entity shape to prepare to be drawn with draw.

The three classes that extend entity, Player, Enemy, and Terrain, serve to differentiate the different types when it comes to actual game elements. Starting with terrain which is not actually any different to entity outside of type alone. Terrain requires no special treatment since its main objective is to sit their and act as the environment for the game world. It still exists though so it can be differentiated from other game elements.

Before discussing the Player and Enemy, the sixth class present in the model needs to be addressed. Healthbars is not part of the hierarchy and does not inherit from Entity or its children. It does however extend Drawable since it is an element that will be drawn onto the window during play. The main purpose of Healthbars is to provide visual representation of the health of various entities within the game. As such, it contains the attributes one would expect such as health, maxHealth, location, size, as well as two shapes to make up the actual visual. It contains the standard getter and setters as well as draw and update. With healthbars covered, Player and Enemy can be addressed. Both these classes have a healthbars object as an attribute so as to visually represent each individual instances health. Alongside the inclusion of a healthbars object, enemy and player have a special boolean as well as a clock. The reason for this is that due to gameplay elements, there are periods where the player or enemy cannot do a certain action for a period of time. In the case of an enemy, if they hit the player or get hit, they are knocked backed and stunned for a couple seconds. The boolean `stunned` in enemy as well as the `stun()` and `isStunned()` methods handle this process by using the clock attribute the enemy has. Stun functions so that if the enemy is not stunned, stun them, and if they are stunned, check the timer, if enough time has passed, unstun and reset the timer for next time. The same applies for player except the player has an invulnerable variation. In the case of a player, they are set to be invuln if they take damage. This lasts a few seconds but during this time, the player cannot take further damage.

6.3 World Generation:

Now addressing the actual game world, specifically its representation as well as its generation. Both components are important towards to final goal of the project.

More specifically the world generation component. How the world is generated will take advantage of the random number generator as well as a very basic AI to convert the numbers from the generator into a proper world. The representation of the world is also an important components but has more impact on the game representation than the project. Regardless, both are required for the game to function as well as for the project to move forward.

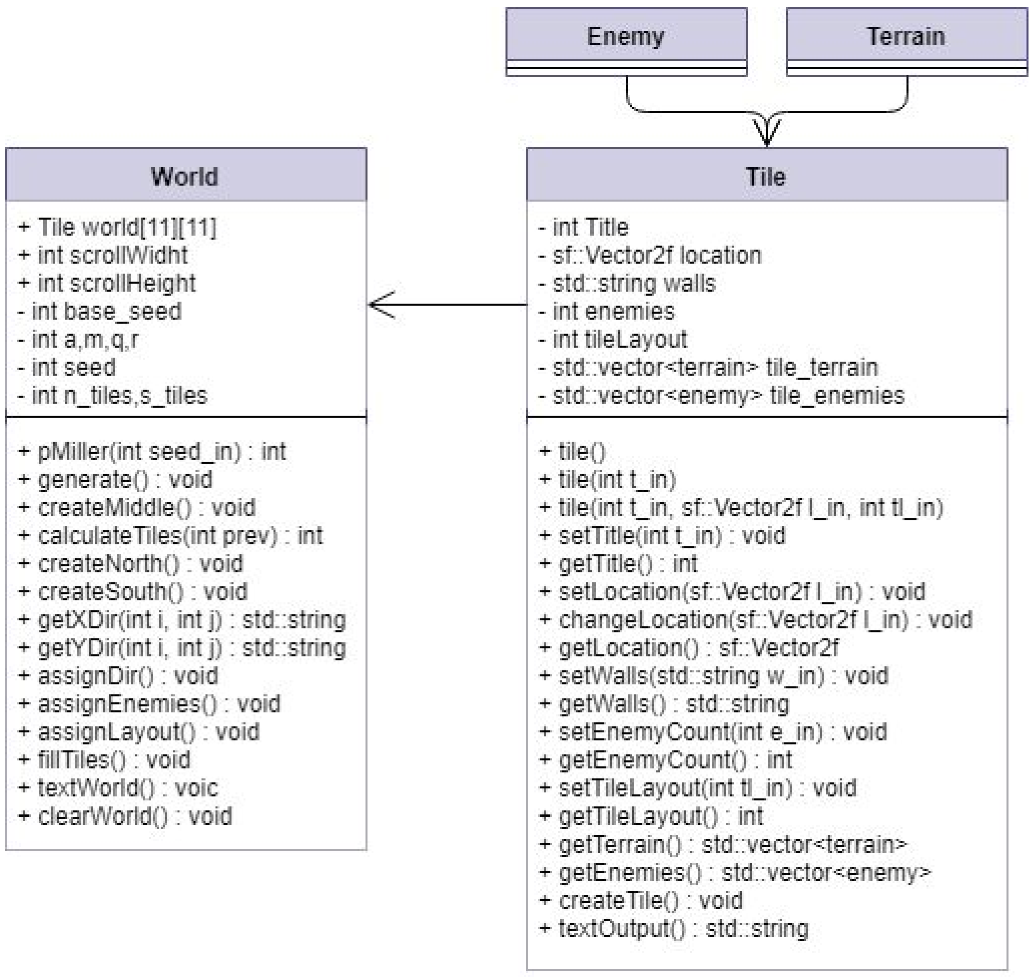

Enemy
Terrain
World
+ Tile world[11][11]
+ int scrollWidht
+ int scrollHeight
- int base_seed
- int a,m,q,r
- int seed
- int n_tiles,s_tiles
+ pMiller(int seed_in) : int
+ generate() : void
+ createMiddle() : void
+ calculateTiles(int prev) : int
+ createNorth() : void
+ createSouth() : void
+ getXDir(int i, int j) : std::string
+ getYDir(int i, int j) : std::string
+ assignDir() : void
+ assignEnemies() : void
+ assignLayout() : void
+ fillTiles() : void
+ textWorld() : voic
+ clearWorld() : void
Tile
- int Title
- sf::Vector2f location
- std::string walls
- int enemies
- int tileLayout
- std::vector<terrain> tile_terrain
- std::vector<enemy> tile_enemies
+ tile()
+ tile(int t_in)
+ tile(int t_in, sf::Vector2f l_in, int tl_in)
+ setTitle(int t_in) : void
+ getTitle() : int
+ setLocation(sf::Vector2f l_in) : void
+ changeLocation(sf::Vector2f l_in) : void
+ getLocation() : sf::Vector2f
+ setWalls(std::string w_in) : void
+ getWalls() : std::string
+ setEnemyCount(int e_in) : void
+ getEnemyCount() : int
+ setTileLayout(int tl_in) : void
+ getTileLayout() : int
+ getTerrain() : std::vector<terrain>
+ getEnemies() : std::vector<enemy>
+ createTile() : void
+ textOutput() : std::string

Figure 9: *World Representation (Oscar W.)*

The model shows the world and how simple it is represented. There only exists two parts when it comes to the world, World and Tile. Each serves a vastly different purpose but they work together to produce the end result of the world. For the game though, the world is represented by a 2D array represented in the World part. Its size is technically adjustable but it has been limited to 11x11 for the sake of consistency. The array itself is specifically public so the game engines can access it and function properly. Other variables within World are base_seed, a,m,q,r, and seed. These are all relevant to the random number generation which is defined in the `pMiller(...)` method contained within world. The core concept regarding the world generation involves the pMiller method as well as the seed variable. Whenever a value of some kind needs to be determined, such as the layout of a tile, number of enemies in the tile, how many tiles north or south to add, the current value of seed is used in combination with modulus to get a value within the range desired. The seed variable is derived initially from pMiller(base_seed) as its initial value. After this point, any time the seed variable is used in any algorithm at any point, the value is updated to be equal to pMiller(seed). The result is that the seed value will contain the current iteration of pMiller when the given input is base_seed.

```
int calculateTiles(int prev)
{
     int tiles;
     if (prev == -1)
     {
          tiles = seed % 5;
          prev = tiles;
          return tiles;
     }
     else
     {
          seed = pMiller(seed);
          tiles = 2 - (seed % 4);
          tiles = prev + tiles;
          if (tiles > scrollHeight / 2)
               tiles = scrollHeight / 2;
          if (tiles < 0)
               tiles = 0;
          prev = tiles;
          return tiles;
     }
}
```

Figure 10: *CalculateTiles Method (Oscar W.)*

Taking the method shown for instance, CalculateTiles(...) is used to determine the height of the next column either north or south based on the height of the previous column. Where prev = -1, this signifies that it is the far left column and as such, has no previous column. Within the else portion of the conditions, seed is immediately assigned to be the next iteration of the pMiller algorithm before being used when calculating values. So every time seed is used in a calculation, it is a unique value for seed based on the output of pMiller(...) with the given base_seed.

Sidetracking to the Tile class for a moment. The Tile class represents and contains all the information relevant to a tile in the world. The result is a fair amount of different information has to be processed in various ways. Starting simple, Title consists of being either 0 or 1, 0 being an empty tile, 1 being an active tile. Any tile with title of 0 has no other attribute values while a title of 1 has all its attributes filled. This is done by the world algorithms. Other attributes include enemies and tileLayout. Enemies is merely the number of enemies in that given tile, where as tileLayout is the numerical representation of the physical layout of the tile. There are 6 possible layouts for a tile, 5 of which contain additional items where as the sixth is a tile with no additional obstacles. Also contained within tile is two C++ vectors that contain lists of the terrain and enemy objects for each tile. These are filled when the createTile() method in a tile is run. This method translates the attributes of the tile into game entities that can be visually and mathematical represented elsewhere in the game. One last variable present is walls which is a string. This is four characters `NSEW` which represent whether that direction from this tile is a empty tile or not. If that tile is a wall, the corresponding letter is set, otherwise it is a '`-`'. For example, a tile that only has an empty tile to the East will be represented with '`--E-`'. A tile then contains all the appropriate getter and setters with a couple extra methods, `createTile()` and `textOutput()`. CreateTile was mentioned earlier but textOutput merely converts the information fo the tile into a concise and easily represented text output. This is less relevant for the game but serves as a way to view the tile without the need of SFML graphical representation.

With the components of world generation addressed, it is time to move on to the actual generation component. The algorithm is based within the generate() method as shown below. Within the method, all the appropriate generation methods also within world are called in the appropriate order.

```
void generate()
{
        clearWorld();
        while (n_tiles + s_tiles >
20)
        {
            n_tiles = 35;
            s_tiles = 35;
            createMiddle();
            createSouth();
            createNorth();
        }
        assignDir();
        assignEnemies();
        assignLayout();
        fillTiles();
        textWorld();
}
```

Figure 11: *Generation Algorithm (Oscar W.)*

As seen, the algorithm consists of seven parts with two housekeeping parts as well. Starting with the core of the process, createMiddle(), createSouth(), and createNorth(). These three methods make up the process that turns the 2D array of tiles from empty to full of active tiles. The process involved starts with taking every tile along the middle horizontal line of the 2D array and turning them into active tiles. The tile on the far left acting as the entrance, and the far right acting as the exit. This means there is always a straight line from the entrance to the exit. After this line is created, the north and south sides of that line are filled with tiles while following some rules.

- *A Column of tiles cannot be greater than two tiles taller than the previous column.*

- *A Column of tiles cannot be greater than one tile shorter than the previous column.*
- *Each side from the centre line will contain at most thirty five total tiles each, totaling to seventy tiles plus the eleve in the middle line to a maximum of eighty one tiles.*
    - *If a side reaches the tile limit before reaching the other side of the world, the rules regarding change in column height are to be ignored.*
- *There is to be a minimum of fifty combined tiles from the north and south sides of the world. Failing to meet this requirement should prompt regeneration of the world until this requirement is met.*
- *The first column on the left is exempt from the first two rules as it has no previous column to compare.*

With the rules taken into account, the world is generated accordingly. As is seen by the for loop, this handles the tile limitations both minimum and maximum by using the variables n_tiles and s_tiles which represent the limit of tiles to be allocated on each side. Once the world has been generated according to the rules laid out, the process moves onto the next step which is `assignDir()`. This method does not actually generate anything but instead takes the world as it is from the initial generation and calculates the direction of walls from each tile. This is mentioned when going over the Tile object, this just does the calculation of the NSEW string of tile and assigns it accordingly. The next step involved now is the assigning of enemies into tiles. The allocation of enemies, like the allocation of tiles, has to follow a list of rules regarding how it is done.

- *A total of fifteen tiles within the world must contain enemies*
- *Any tile that contains enemies must contain between one and five enemies*
- *No enemies can spawn within a two tile radius of the entrance tile*
- *No enemies can spawn in a tile that is adjacent on the horizontal or vertical to another tile that contains enemies.*

The rules regarding the allocation of enemies is less strict than the tile allocation but the rules must be followed and have been properly implemented within the `assignEnemies()` method. The last generation step involved is the assigning of layouts to the tiles. The tilelayout will determine what other elements, specifically terrain, are contained within that tile. As with before, `assignLayout()` has a set of rules to follow regarding its generation.

- *The Entry and Exit Tile must be of an Empty Layout*
- *There should be a 4/9 chance that a tile is of an Empty Layout*
- *A Tile of that is adjacent to a tile of Layout five has a 3/8 chance of being a tile of Layout five*
    - *The remaining 5/8 chance will be assigned as normal*

The rules regarding layouts are also not exceedingly strict but are more precise in their purpose. WIth regards to the Layout five Rule, this is a design choice based on the layout of tile five specifically. After this step, the remaining steps include `fillTiles()` and textWorld(). Neither of these methods generate new content. The fillTiles method turns the information assigned from the previous methods into workable game elements by calling the `createTile()` method as described earlier within each Tile. With textWorld, that merely prints out a text representation of the world for observation and testing purposes.

6.4 AI:

6.4.1 Entity AI:

Beginning with the AI surrounding the Entities within the game. For said entities, they employ a state machine in order to decide what they should be doing. The parameters of which include their current percentage of health as well as their

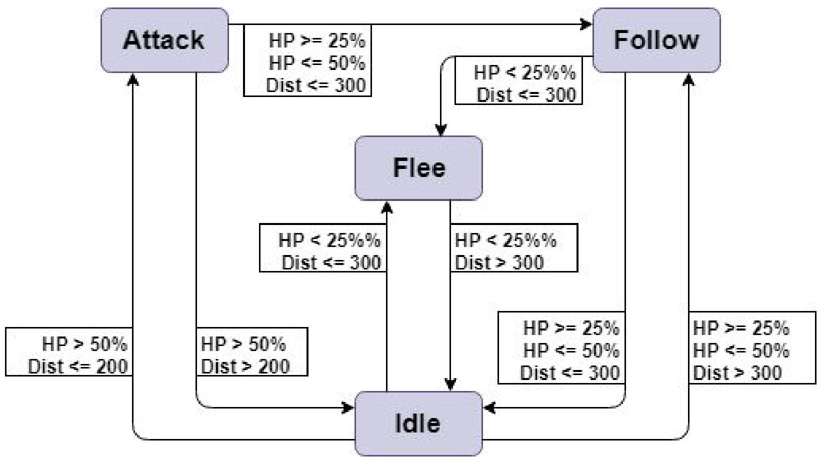


***Figure 12:** Entity State Machine (Oscar W.)*

distance to the player. Further variables could be taken into consideration such as line of sight or other similar aspects but at present, no further values are considered. The State machine itself is comprised of four different states, one of which is the default state. The states include ‘Attack’, ‘Follow’, ‘Flee’, and ‘Idle’.

As their names imply, the action the entity will perform corresponds directly to the name of the state. Idle is of course the default state which occurs primarily when the player is outside of the ranges defined by the state machine. When idle, the entity will at random intervals, pick a random direction to travel and move its velocity times one in that direction. This results in the AI occasionally taking a step in some direction if the player is not within range. The purpose of such is to give the entities some semblance of life rather than motionless entities that only move if the player is close. With regards to the other three states, they all make use of one or two of the other defined methods available to them.

```
sf::Vector2f calcEnemyVel(player _p, enemy& _e)
{
     enemySpeed = baseSpeed * speedModifier;
     int x = (_p.getLocation().x + _p.getSize().x / 2) - 
(_e.getLocation().x + _e.getSize().x / 2);
     int y = (_p.getLocation().y + _p.getSize().y / 2) - 
(_e.getLocation().y + _e.getSize().y / 2);
     double d = sqrt((pow(x, 2)) + (pow(y, 2)));
     sf::Vector2f vel = (sf::Vector2f((enemySpeed / 
d)*x, (enemySpeed / d)*y));
     return vel;
}
```

***Figure 13:** calcEnemyVel (Oscar W.)*

Shown here, calcEnemyVel(...) is used to determine the velocity in which the entity will move. No intricate path finding is being used and instead entities will attempt to move in a straight line towards the player regardless of any entities in its way. Attack and Flee both use this method directly, Attack will set the velocity of the given entity to the value calculated from this method where as Flee sets the velocity to negative the method output. This produces the effect of Attack heading straight towards the player and Flee, the opposite, running directly away from the player.

Following also uses the same math in this method but not this method specifically as additional math is required. The reason is that with Follow, its objective is to stay 100 units of distance away from the player at all times, staying near the player, but not attacking nor running away as long as it’s within range. The primary difference between the method above and the method Follow uses, is that Follow has additional mathematical steps in order to calculate the point one hundred units away from the player in the direction of the entity. One additional aspect of the AI is the enemy's special trait of being stunned. As mentioned previously, Enemies will be stunned upon interaction with the player so as to avoid being a constant threat and providing no breathing room. When it comes to how this interacts with the state machine, being stunned turns the state machine off until unstunned. As long as the entity is stunned, it will not be in any of the states shown above and instead be in a ‘limbo’ of no directive.

6.4.2 Director AI:

The Director AI was more of a problem to implement than initially believed. Due to little experience in the field of AI as well as a limitation of the game elements to be observed, the implemented state of the Director is simple. Despite that, it does contribute to the intended purpose of adding replay value and dynamic changes to the game as the user plays. Given further implementation and development of the game, the Director would grow and improve alongside it.

Onto how the Director was implemented at present. The Director consists of a State Machine based on weighted probability as well as methods which dynamically and statically change game values based on both the Director State Machine and observed game values. The values observed by the Directory consist of the players current health (0-1), the number of enemies remaining in the level (0-1) and the time elapsed in the current level (0-300s). The values observed from those three game elements make up the processes and modifications of the game values to be altered. The Values that can be modified include the health of Enemies, the damage of enemies, the speed of enemies, the range in which enemies will interact with the player, and the damage of the player. The roll the state machine fills is it provides another value to modify other values by depending on the ‘mood’ of the Director.

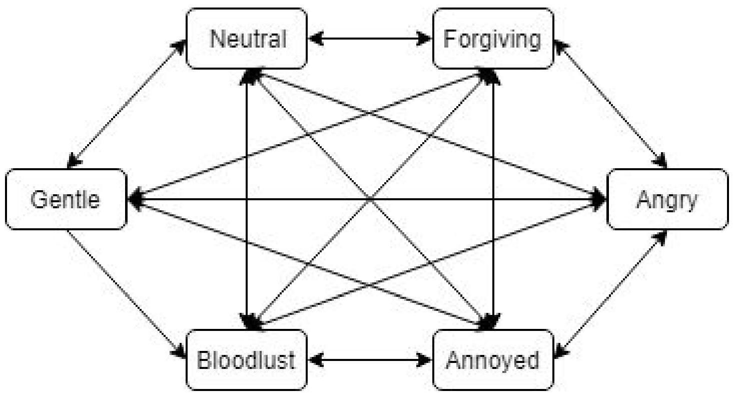
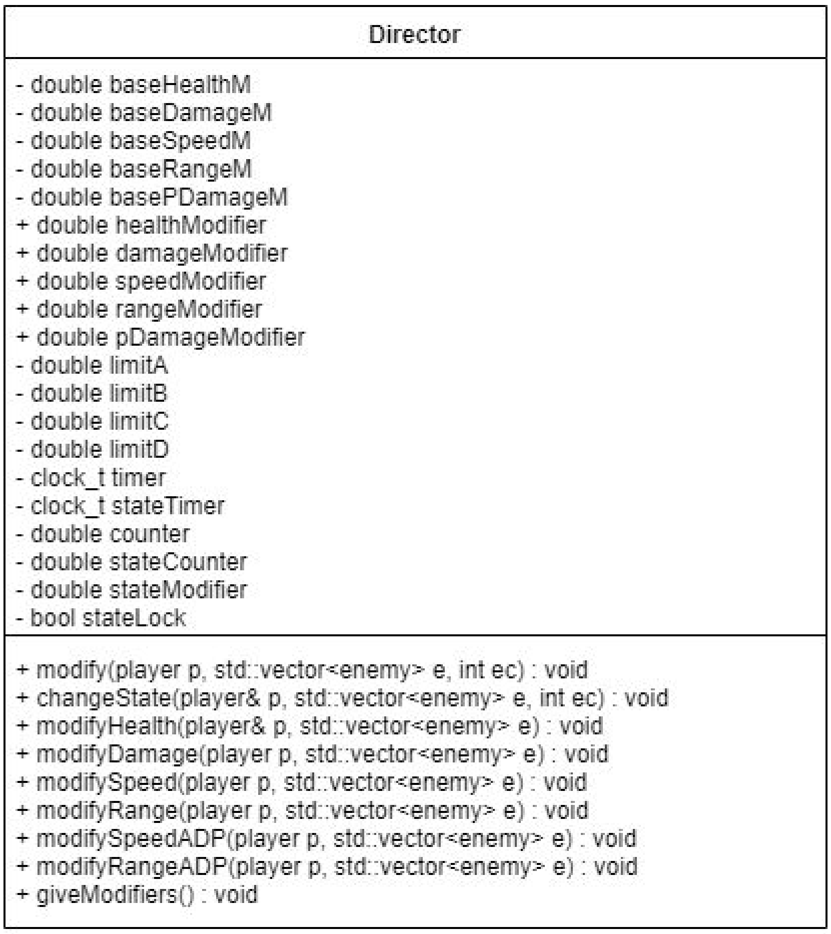


Figure 14: *Director State Machine (Oscar W.)*

As shown, the Director consists of 6 states that are all interchangeable with one another. The conditions for the change are not consistent and are instead reliant on observable game values. The way this works is that each state has a percentage chance to be picked by using a random number, with Neutral being the base state. Every fifteen seconds, the Director rolls the dice and switches to the corresponding state. The actual chances though adapt and change based on the game values the Director observes. For example, Assuming All enemies remain and the player is at max health, the Director will have roughly a seventy percent chance of being Neutral. However, as your health lowers, the chance of the Director switching to the other states increases. This works in conjunction with enemies remaining as well. Assume the player is at low health but all enemies remain, the chance of the Director being forgiving or gentle go up higher than angry or annoyed do. Another example is if the player is at max health but the remaining enemies is low, the chance of gentle or forgiving will go down while the chance of Angry, Annoyed, and Bloodlust will go up accordingly. The chance for neutral will always be the remaining percentage after the chance for the other five states is calculated. The combination of using weighted chance with the observable game values means the Director will make different choices on its state every time you play. The result of each state though is changing an internal value to the corresponding multiplier. Neutral is a 1.0 multiplier where as

Bloodlust is 2.0 and Gentle is 0.8. The resulting modifier is used when changing the game values either dynamically or statically.



```
Director
- double baseHealthM
- double baseDamageM
- double baseSpeedM
- double baseRangeM
- double basePDamageM
+ double healthModifier
+ double damageModifier
+ double speedModifier
+ double rangeModifier
+ double pDamageModifier
- double limitA
- double limitB
- double limitC
- double limitD
- clock_t timer
- clock_t stateTimer
- double counter
- double stateCounter
- double stateModifier
- bool stateLock
+ modify(player p, std::vector<enemy> e, int ec) : void
+ changeState(player& p, std::vector<enemy> e, int ec) : void
+ modifyHealth(player& p, std::vector<enemy> e) : void
+ modifyDamage(player p, std::vector<enemy> e) : void
+ modifySpeed(player p, std::vector<enemy> e) : void
+ modifyRange(player p, std::vector<enemy> e) : void
+ modifySpeedADP(player p, std::vector<enemy> e) : void
+ modifyRangeADP(player p, std::vector<enemy> e) : void
+ giveModifiers() : void
```

Figure 15: *Director File (Oscar W.)*

Moving onto the actual implementation of the Director, it consists of a large amount of variables as well as the methods to modify game values. The main variables are all the baseXXXXM variables as well as the XXXXModifier variables. These variables keep track of the current modifier. The base variables keep track of static modifiers while the Modifier variables track it dynamically. What this means is that whenever the game moves onto a new level, it will adjust all the base modifiers accordingly and only when the game progresses. The Modifier variables are always equal to the corresponding base multiplied by the various other modifiers the Director observes and uses. When the value of a Modifier variable is changed, it will effect the game immediately. So if the speedModifier is increased, the speed of all enemies at that given time will go up appropriately. Changes to the base variables only occur and take effect at specific times and not during direct gameplay. This is also highlighted in the methods Director contains. Methods containing ADP at the end mean that these methods are run during gameplay to change the modifier values and not the base values. Methods not containing ADP only run at specific times and only change the base value and not the modifier value.

How the Director modifies game values is mostly just mathematical calculations use the various observed data as well as the modifier obtained from the Directors state. Doing so means that game values will change somewhat consistently assuming the same situation occurs. The prime game value the director controls is the speed of enemies. What the Director does is take the players health into account and adjust the speed accordingly. Starting from max health to 62.5% health, the Director will slow down enemies proportionally before reaching the their slowest at a health of 62.5%. After this point, the Director will start raising their speed back up all the way till 25% health. At 25%, the enemies will be moving at normal speed again. Any health lower than 25% causes the enemies to move proportionally faster.

7.0 Testing:

The testing of the software was quite scarce in some areas for various reasons as will be covered here. Most of the in depth testing involved was surrounding the world generation algorithm which used the random number generators. Example tests will be given as and where possible, otherwise the process of how components were tested will be given with or without proper representation depending.

7.1 Game Engine Testing:

The testing of the game engine is a difficult task to represent. The primary reason is that the engine being used in the project is an engine that was developed previously in freetime nearly half a year prior to the project starting. As a result, the testing for a range of the functionality of the engine goes undocumented. The method of how components were tested though is something that can be explained.

For the most part, the engine focused on the translation of the values and attributes of components into visual representation. This being the case, most if not all the testing required the observation of the graphical output and looking for behaviour of components according to how is intended. One example of a case where this would occur would have been during the testing of the collision algorithms. This would have been achieved by simply having two entities move towards one another and seeing if they stop before they clip into one another. The values for components such as position and velocity were also listed on the screen to observe for any abnormalities where the graphical representation mismatched the internal representation. This process of observational testing would repeat for most of the engine components. It should be noted that testing of the engine during the scope of the project was done in the same manner primarily due to habit and failure to come up with alternative, documentable forms of testing.

7.2 World Generation Testing:

Unlike the game engine, the testing of the world generation was easier to document and record as it was developed during the project as opposed to beforehand. The order of testing was done in accordance to various milestones of the world generation as well as when one or multiple of the predefined world generation rules was achieved.

7.2.1 createMiddle, createNorth, createSouth part 1:

The first task that was implemented and tested was the ability to fill the 11x11 grid representation of the world with tiles according to how specified. In particular, the middle row consisting of entirely tiles and then the north and south of that row being filled with tiles based on the random number generator outputs. The first step was exactly that, filling the middle row with tiles and then the north and south columns with pseudo-random amount of tiles outwards from the middle. While the rule defined constraints for the column changes, the first step of testing involved the implementation of merely the proper order of allocation.

Test A	Test B	Test C	Test D

Seed: 6183 Alpha: 98100 11:11 ----------- ----XX-X--- XX--XX-XX-- XX-XXXXXXX- XXXXXXXXXX- XXXXXXXXXXX -X-XXXXXXXX -X-XXXXXXXX -X-XXXXXXXX -X-X-XXXXXX -X-X----XX-	Seed: 20173 Alpha: 98100 11:11 ----------- -----X-X--- -----XXXXX- -XXX-XXXXX- -XXX-XXXXXX XXXXXXXXXXX -XXXXXXXXXX -XXXXXXXXXX -XX-X-XXXXX -X--X--XXXX -X-----XXXX	Seed: 20173 Alpha: 742 11:11 ----------- XX-XX-XXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX --XXXXXXXXX --X-XXXXXXX ----XXXXXXX -----XXXXXX	Seed: -100 Alpha: 972 11:11 ----------- --X-XXXXXXX --XXXXXXXXX --XXXXXXXXX --XXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX --XXXXXXXXX ----XXXXXXX ----XXXXXXX
Test E	**Test F**	**Test G**	**Test H**
Seed: 9360 Alpha: 742 11:11 ----------- ---XX-XXXXX ---XXXXXXXX X--XXXXXXXX XXXXXXXXXXX XXXXXXXXXXX -XXXXXX-XXX --X-XXX-XXX --X-XX--XXX ----XX--XXX ----X---XXX	Seed: 35 Alpha: 3 11:11 ----------- --XXXXXXXXX --XXXXXXXXX -XXXXXXXXXX -XXXXXXXXXX XXXXXXXXXXX -XXXXXXXXXX -XXXXXXXXXX --XXXXXXXXX --XXXXXXXXX ---XXXXXXXX	Seed: 7 Alpha: 64 11:11 ----------- -------XXX- -------XXXX XX-----XXXX XXXX--XXXXX XXXXXXXXXXX XXXXX-XXXXX XXXXX--XXXX XX-----XXXX -------XXXX -------X-XX	Seed: -655 Alpha: -78 11:11 ----------- XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX --X-XXXXXX- --X-XXXXXX- -----X--X-- ----------- -----------

Table 8: *World Gen Tests 1 (Oscar W.)*

The results of the implementation of the world gen is shown within the 8 tests given above. Each test shows the output 11x11 grid where 'X' represents a tile and '-' represents a blank space. It can be observed that the objectives of the current tests was achieved. The middle row is filled with tiles every run and tiles to the north and south are allocated pseudo-randomly according the the random number algorithm being used.

7.2.2 createMiddle, createNorth, createSouth part 2:

With the base world generation sorted, it is now the time to add the restrictions to the createNorth and createSouth methods so that the change in column heights adheres to the rules specified before. The process involved in adding the restrictions was not a difficult task and was completed quite quickly. The other rule implemented at this time was the tile limit. The limit of tiles on the north and south side was added during this time as well. At this point, all the rules regarding how the tiles are allocated into the world have been fulfilled.

Test A	Test B	Test C	Test D
Seed: 6183 Alpha: 98100 11:11 ----------- -XXXXXXXX-- -XXXXXXXX-- XXXXXXXXX-- XXXXXXXXXX- XXXXXXXXXXX -XXXXXXXXXX -X---XXXXXX -----XXXXXX ------XXXXX -------XXXX	Seed: 20173 Alpha: 98100 11:11 ----------- -----XXXXXX -----XXXXXX X---XXXXXXX XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXX-- XXXXXXXX--- XXXXXXXX--- X--XXXXX--- ---X-XXX---	Seed: 20173 Alpha: 742 11:11 ----------- X-XXXXXXX-- XXXXXXXXX-- XXXXXXXXX-- XXXXXXXXX-- XXXXXXXXXXX XXXXXXXXXX- -XXXXXXXX-- --XXXXXXX-- ----XXXXX-- ----XXXXX--	Seed: -100 Alpha: 972 11:11 ----------- ------XXXXX -----XXXXXX ----XXXXXXX ---XXXXXXXX XXXXXXXXXXX XXXXXXXXXXX X-XXXX-XXXX -------XXXX --------XXX --------X-X
Test E	**Test F**	**Test G**	**Test H**
Seed: 9360 Alpha: 742 11:11 ----------- -XXXXXXXX-- XXXXXXXXX-- XXXXXXXXX-- XXXXXXXXX-- XXXXXXXXXXX --XXXXXXXXX --XXXXXXXXX ----XXXXXX- ----XXXXXX- -----XXXXX-	Seed: 35 Alpha: 3 11:11 ----------- ----XXXXXXX ---XXXXXXXX --XXXXXXXXX --XXXXXXXXX XXXXXXXXXXX ------XXXXX ------XXXXX -------XXXX -------XXXX --------XX-	Seed: 7 Alpha: 64 11:11 ----------- -XXXXXXXX-- -XXXXXXXX-- XXXXXXXXX-- XXXXXXXXXX- XXXXXXXXXXX XXXXXXXX--- XXXXXXXX--- XXXXXXXX--- -XXXXXX---- -XXX-XX----	Seed: -655 Alpha: -78 11:11 ----------- X--XXXXXX-- XXXXXXXXX-- XXXXXXXXX-- XXXXXXXXXX- XXXXXXXXXXX --XXXXXXXXX --XXXXXXXX- ---XXXXXX-- ---XXXXX--- ----XXX----

Table 9: *World Gen Tests 2 (Oscar W.)*

The validity of the statement that the world generation tile allocation rules were implemented can be seen in the testing shown above. Along with the qualities of the first tests from earlier, the testing here now shows that the rules regarding the tile limit and the height variance are implemented. At no point does the height of a tile exceed the previous plus two nor does the height lower by more than one tile from the previous except when the tile limit has been reached as seen in a number of the tests. With all this now in place, the world in its most basic form can be generated and represented within the game according the basic rules. The next stages of testing cover the allocation of enemies as well as the allocation of tile layouts.

7.2.3 allocateEnemies:

With how to allocate enemies, the actual implementation is easy as well as the actual testing. All it involves is picking a valid tile, checking its surroundings, and assigning a number of enemies should the rules be fulfilled. When testing of the allocation was done, a list of the tiles where enemies were allocated as well as how many enemies in that tile were printed out. This can be seen below with four tests of the allocation. As can be observed, the enemies within a tile range from one to six, and there are only fifteen tiles that contain any enemies. What can also be observed is that no two adjacent tiles contain enemies. All these points comply with the rules set out with the allocation of enemies within the world.

Test A	Test B	Test C	Test D
Seed: -655	Seed: 20173	Seed: 20173	Seed: -100
Alpha: -78	Alpha: 98100	Alpha: 742	Alpha: 972
11:11	11:11	11:11	11:11
Tile [8,4]	Tile [8,6]	Tile [5,3]	Tile [5,3]
Enemies:5	Enemies:4	Enemies:3	Enemies:1
Tile [2,4]	Tile [7,3]	Tile [1,2]	Tile [4,4]
Enemies:2	Enemies:3	Enemies:4	Enemies:1
Tile [1,7]	Tile [7,11]	Tile [7,3]	Tile [6,0]
Enemies:5	Enemies:1	Enemies:6	Enemies:6
Tile [5,4]	Tile [7,5]	Tile [4,8]	Tile [1,9]
Enemies:5	Enemies:3	Enemies:3	Enemies:1
Tile [7,3]	Tile [8,2]	Tile [2,8]	Tile [1,7]
Enemies:5	Enemies:3	Enemies:4	Enemies:2
Tile [5,6]	Tile [6,8]	Tile [7,7]	Tile [8,7]
Enemies:3	Enemies:3	Enemies:5	Enemies:6
Tile [0,11]	Tile [1,5]	Tile [9,4]	Tile [3,5]
Enemies:2	Enemies:2	Enemies:1	Enemies:2
Tile [4,2]	Tile [2,7]	Tile [2,6]	Tile [6,7]
Enemies:1	Enemies:1	Enemies:3	Enemies:4
Tile [4,0]	Tile [3,0]	Tile [2,0]	Tile [4,9]
Enemies:5	Enemies:6	Enemies:4	Enemies:4
Tile [5,9]	Tile [4,11]	Tile [1,4]	Tile [8,10]
Enemies:5	Enemies:1	Enemies:1	Enemies:2

Tile [9,5]	Tile [4,6]	Tile [9,6]	Tile [10,8]
Enemies:5	Enemies:6	Enemies:5	Enemies:5
Tile [6,7]	Tile [1,10]	Tile [6,1]	Tile [3,10]
Enemies:1	Enemies:4	Enemies:1	Enemies:1
Tile [9,3]	Tile [5,3]	Tile [4,1]	Tile [6,2]
Enemies:4	Enemies:6	Enemies:5	Enemies:5
Tile [3,7]	Tile [9,5]	Tile [4,4]	Tile [6,4]
Enemies:6	Enemies:6	Enemies:4	Enemies:4
Tile [3,5]	Tile [4,1]	Tile [5,9]	Tile [6,10]
Enemies:3	Enemies:6	Enemies:5	Enemies:5

Table 10: *Enemy Allocation Test (Oscar W.)*

7.2.4 assignLayout:

Before addressing the assignment of layouts, it should be noted that at this stage of implementation, the text layout of the world was changed to include more information compared to earlier. Compared to previously where the world only consisted of 'X's and '-'s, the new representation contains the direction of walls, the number of enemies, as well as the tile layout. An example of the new representation is shown below.

```
[ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ N--W 2 5 ][ N--- - 1 ][ N--- 2 3 ][ N--- - 2 ][ N-E- - 5 ][ -------- ]
[ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---W - 5 ][ ---- - 5 ][ ---- - 5 ][ ---- 3 7 ][ ---- - 6 ][ N-E- - 3 ]
[ -------- ][ N-EW - 4 ][ -------- ][ N--W - 6 ][ N--- - 2 ][ ---- 2 5 ][ ---- - 2 ][ ---- - 2 ][ ---- - 1 ][ ---- - 2 ][ --E- 3 7 ]
[ -------- ][ ---W - 3 ][ N--- - 1 ][ ---- 3 9 ][ ---- - 8 ][ ---- - 2 ][ ---- 4 6 ][ ---- - 9 ][ ---- - 5 ][ ---- 2 1 ][ --E- - 7 ]
[ N--W - 9 ][ ---- - 3 ][ ---- - 4 ][ ---- - 9 ][ ---- - 1 ][ ---- 1 2 ][ ---- - 2 ][ ---- - 3 ][ ---- - 5 ][ -S-- - 5 ][ -SE- - 9 ]
[ ---W - 8 ][ ---- - 8 ][ ---- - 4 ][ ---- 2 8 ][ ---- - 3 ][ ---- - 1 ][ ---- - 2 ][ ---- 2 4 ][ -SE- - 5 ][ -------- ][ -------- ]
[ ---W - 7 ][ ---- - 5 ][ ---- - 1 ][ ---- - 9 ][ ---- 5 1 ][ ---- - 1 ][ ---- - 8 ][ -SE- - 2 ][ -------- ][ -------- ][ -------- ]
[ ---W - 9 ][ -S-- - 9 ][ -S-- 5 8 ][ ---- - 7 ][ -S-- - 2 ][ ---- - 4 ][ -SE- - 9 ][ -------- ][ -------- ][ -------- ][ -------- ]
[ -SEW - 3 ][ -------- ][ -------- ][ -SEW 4 3 ][ -------- ][ -SEW 3 9 ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
```

Figure 16: *World Text Representation 1 (Oscar W.)*

With the new text representation addressed, the testing of the layout allocation can be discussed. The results of the testing can be observed even in the above figure because it contains the tile layout in the output. Since the layout is just a pseudo-random number between one and nine, that part of the implementation was simple to do and can be seen in the above figure with all the tiles containing a tile layout within that range. Looking at the other rules specified in the design, both the entrance and exit tiles are given a blank tile layout, specifically tile layout nine. Any tile layout between five and nine inclusive is considered a empty tile. All other tile layout values indicate a specific tile layout. The one remaining rule regarding adjacency to tiles with tile layout five is harder to observe properly but can be observed to some degree. Specifically where an abnormal amount of tiles with layout five are adjacent to one another. The world above does not show this quality very well. Another output shown below does a better job of representing the fact the rule regarding layout five is implemented.

```
[ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[ -------- ][ N--W - 9 ][ N--- - 5 ][ N--- - 5 ][ N-E- - 5 ][ -------- ][ -------- ][ N--W - 6 ][ N-E- - 4 ][ -------- ][ -------- ]
[ N--W 2 4 ][ ---- - 2 ][ ---- - 4 ][ ---- - 5 ][ ---- 4 5 ][ N--- - 6 ][ N--- - 6 ][ ---- 5 3 ][ --E- - 8 ][ -------- ][ -------- ]
[ ---W - 8 ][ ---- - 6 ][ ---- - 2 ][ ---- 3 5 ][ ---- - 1 ][ ---- - 1 ][ ---- - 2 ][ ---- - 8 ][ ---- 1 2 ][ N-E- - 4 ][ -------- ]
[ ---W - 7 ][ ---- - 5 ][ ---- - 5 ][ ---- - 5 ][ ---- - 9 ][ ---- - 4 ][ ---- 4 3 ][ ---- - 6 ][ ---- - 2 ][ --E- - 7 ][ -------- ]
[ ---W - 9 ][ ---- - 8 ][ ---- - 5 ][ ---- - 5 ][ ---- - 5 ][ ---- 1 6 ][ ---- - 4 ][ ---- - 4 ][ -S-- - 2 ][ -S-- 4 3 ][ NSE- - 9 ]
[ ---W - 5 ][ ---- - 5 ][ ---- - 7 ][ ---- 2 1 ][ ---- - 5 ][ ---- - 4 ][ ---- - 4 ][ --E- - 9 ][ -------- ][ -------- ][ -------- ]
[ ---W - 4 ][ ---- - 6 ][ ---- - 6 ][ ---- - 3 ][ ---- - 3 ][ ---- - 8 ][ ---- - 3 ][ --E- 2 1 ][ -------- ][ -------- ][ -------- ]
[ -S-W - 1 ][ ---- - 3 ][ ---- 5 7 ][ ---- - 3 ][ ---- - 5 ][ ---- - 5 ][ ---- - 4 ][ --E- - 4 ][ -------- ][ -------- ][ -------- ]
[ -------- ][ ---W 5 7 ][ ---- - 8 ][ ---- - 5 ][ -S-- 5 1 ][ -S-- - 5 ][ -S-- 2 2 ][ --E- - 2 ][ -------- ][ -------- ][ -------- ]
[ -------- ][ -S-W - 6 ][ -S-- 4 4 ][ -SE- - 5 ][ -------- ][ -------- ][ -------- ][ -SEW - 6 ][ -------- ][ -------- ][ -------- ]
```

Figure 17: *World Text Representation 2 (Oscar W.)*

With all the above world gen testing covered, all the rules laid out in the design have been fulfilled and the world generation component can be deemed finished for the given design specified.

7.3 AI Testing:

With regards to how the AI testing was done, it was primarily done by checking the values of attributes whenever the State machine of the AI changes. Such as the health of an entity when it changes state and similar. This was also combined with observational testing as mentioned before where how the AI acted during a run of the system was observed to see if it matched with expectations.

7.3.1 Entity AI:

The AI behind the Entities consists of a state machine so the process of testing is checking the conditions of the values that decide when a state is changed. Temporary print statements were added into the system that would print out Whenever the state of an entity was changed. Also included was the State being changed to, the entities max health, current health, and distance to the player. This allows viewing of what caused a state to change. The conditions for state changes were already described prior. Shown now is a run of the entity changing states between its possible states.

```
INITIAL STATE SET
- STATE CHANGE -
state = Attack
maxHealth = 50
currentHealth = 50
distance = 200
- STATE CHANGE -
state = Idle
maxHealth = 50
currentHealth = 50
distance = 201
- STATE CHANGE -
state = Attack
maxHealth = 50
currentHealth = 50
distance = 200
- STATE CHANGE -
state = Follow
maxHealth = 50
currentHealth = 20
distance = 57
- STATE CHANGE -
state = Idle
maxHealth = 50
currentHealth = 20
distance = 301
- STATE CHANGE -
state = Follow
maxHealth = 50
currentHealth = 20
distance = 300
- STATE CHANGE -
state = Flee
maxHealth = 50
currentHealth = 10
distance = 62
- STATE CHANGE -
state = Idle
maxHealth = 50
currentHealth = 10
distance = 301
```

Figure 18: *Entity AI State Change (Oscar W.)*

As can be observed from the results, the AI correctly swaps its states the appropriate times. Specifically whenever the player is either in or out of range, and the percentage remaining health of the entity. The above was repeated a few times just to confirm and every test featured the same results as expected. In combination with the testing shown, observational testing was done to visually confirm the AI did indeed act how it should. Specifically that when attacking, the entity would directly move to the player, this was achieved. When following, the entity would move towards the player until it reached the predefined distance away from the player. This behaviour was observed to take place. And finally fleeing where the entity should move directly away from the player until out of range. Just like the other behaviours, this was observed to be the case when the state was as such. The same tests were done on multiple occasions as and when internal values were tweaked to suite the gameplay and pace but no other forms of testing were done as they were deemed not necessary.

7.3.2 Director AI:

When it comes to testing the Director AI, specifically the state machine, it proves difficult since the state machine behind the Director does not change on specific conditions but rather based on a random chance using weighted probability. Each state the director can be is given a chance to swap to based on the observable values. The state is then picked at random using the chances they have been assigned. As such, specifically checking whether the director correctly swaps to a given state is not easy since there are no concrete values that dictate what it swaps to. Regardless though, it can be tested that the Director does swap between its stated at a seemingly random rate. The AI is set to change mood every fifteen seconds. Over time, the change of state can be observed given enough time. The other element to tests are the change of the base modifiers every time the level is changed as well as the adaptive speed modification. With regards to the base modifiers, this can be tested by checking the modifier values every time a level is generated. Assuming it functions correctly, the modifiers will be changed according to the observed values

```
INITIAL STARTUP
Health Modifier: 1.0
Damage Modifier: 1.0
Player Modifier: 1.0
Speed Modifier : 1.0
Range Modifier : 1.0
```

```
State = Neutral
STATE CHANGE
State = Neutral
STATE CHANGE
State = Neutral
STATE CHANGE
State = Annoyed
STATE CHANGE
State = Neutral
STATE CHANGE
State = Forgiving
STATE CHANGE
State = Neutral
LEVEL CHANGE
State on Change = Neutral
Health Modifier: 1.53239
Damage Modifier: 1.24684
Player Modifier: 1.29412
Speed Modifier : 1.0
Range Modifier : 1.0
```

Figure 19: *Director State and Modifier Test (Oscar W.)*

The first two qualities to test are shown within the figure above. The changing of the state on a seemingly random basis as well as the change of modifiers according to the observed values when the level is changed. The modifiers all sit at the base 1.0 on startup and can be seen to have changed when the level changes. Speed and Range remain unchanged as they do not have modifications based on observable values but rather on the state of the director on level change. With the state changing, every time the state is changed, the new state is printed. Though only a small amount of changes were presented, the testing was done over a large period of time to determine whether state changed as described.

```
1.0308
1.031
1.03114
1.03131
TAKEN DAMAGE
0.939793
0.939934
0.940084
0.940239
…
```

```
0.9492
0.94936
0.94951
TAKEN DAMAGE
0.857019
0.857146
…
0.743838
0.743955
0.744069
TAKEN DAMAGE
0.840218
0.840338
0.840467
```

Figure 20: *Director Adaptive Speed Modifier (Oscar W.)*

Now presented is the testing involved when determining whether the Director modified the speed dynamically. The way in which this was tested was by continuously outputting the speed modifier to see it change. From the test, it can be seen that the modifier changes firstly, over time. The longer that is spent on a single level, the faster everything will move as a whole. The other quality observed is that when the player takes damage, as signified by 'TAKEN DAMAGE' the speed modifier is decreased accordingly. Both of the qualities observed are qualities the Director is meant to have which means that the methods function correctly. Also observed in the testing above is that at later points, taking damage causes the speed modifier to increase. This is desired as the health of the player modifies the speed in a parabola as described previously. The testing done on the director shows that the intended functions to function as design and specified.

8.0 Evaluation:

Going over the results of the project as well as the developed artefact, the results were mixed as a whole. The Objective was the development of a game that implemented Random Number Generators and Artificial Intelligence in order to increase replayability of the game.

8.1 What was Achieved and could be Improved:

In regards to how well those objectives were achieved, it can safely be said that the objectives were most definitely achieved. The system developed during the project serves as the framework for a game but lacks a large amount of defining features. Features include proper graphics, menus, proper user interfaces, as well as more in depth game mechanics. At the start of the development, it was intended to develop deeper gameplay mechanics such as skills and items into the game. As time went on, this became less and less likely to occur as more and more time was taken in other fields of development. The result is what the game is at the present, it contains basic gameplay mechanics that can be played to a degree. Compared to actual games in the Roguelike genre, the project system is sorely lacking from a commercial standpoint. Thankfully that the purpose of the project was not for a commercially available product but rather seeing how gameplay elements can be combined with AI and RNG to 'improve' the game.

Moving onto the objective of using RNG within the game. This was done to a great deal specifically when it came to the game world. The usage of RNG was already decided before design even began. The roguelike genre of games has always made use of procedural generation to provide a different experience every time you play. This was no different in the scope of the project. What the project focused on was on what elements could RNG be applied to. This was achieved and a number of elements were picked, including size and shape of the world, where enemies are, etc. However, problems arise from what was mentioned earlier in that the game engine does not facilitate a large amount of game elements. This being the case, further effort was done during development to apply the RNG to the elements available at the time. The focus of using RNG had changed throughout the project, most specifically how it was going to affect various game elements. There were plans to have RNG not only influence enemy locations but their health, speed, and damage. This was put to the side while more development time went into the AI of the game.

Speaking of the AI, the AI proved to be the biggest problem with the project. Specifically the Directory AI. Designing and Developing the AI of the entities was simple as they were given a simple state machine. Of course given more development time, the AI behind entities could have been vastly improved. Moving onto the Director AI, this was a large hurdle to do with the given game engine. The problem that arose was a lack of experience with AI as well as a lack of data within the game that the Director could observe. Compared to the Director AI as described in Left 4 Dead (Valve Corporation, 2008), the one in the project is lacking in complexity and depth. Other reasons that contribute to the difficulty to the development of the Director was how late into development the idea of the Director was decided. Previously it had been decided that AI would be combined with the world generation to change aspects but the methods and processes behind it were never decided. A long time was spent struggling on how to decide the implementation. It was when additional research into AI where mentioned of the Director AI was brought up. At this point it was decided to not use World Generation based AI but instead AI that controls the pace and difficulty of the game as the game is played. The plan was good and progress was made but limitations were reached due to the game engine. Because of the depth of the game engine, what could be modified and observed was limited. It was also quite late into development to suddenly design and implement a range of new elements that could be modified or observed. As a result, it was decided that the Director AI would have to be simpler than initially intended.

The overarching issue that arose from the project was how in depth the game engine was and the gameplay elements. There was enough to develop the basics of AI and world generation but not enough to properly expand on those fields. As a result, the effect the project had at achieving its objective is limited. While ways to implement AI and RNG into gameplay elements were found and achieved, they are not at a level that would be deemed well. With all this, most of the issues can merely be solved with with additional development time or additional manpower. The base premise was to develop a game, which is not an easy task as a single person in the given timeframe, though what was achieved can be said to be a good amount of progress.

8.2 Results of the Project:

The results of the project primarily include how the World Generation was developed as well as the Director AI and the impacts on the game. When considering other examples of World Generation and the Director AI, the results of the project do not contribute a great deal. World Generation in games is nothing new, especially within the roguelike genre. The project merely contributes to the notion that RNG can be used for world generation to great effect. With the Director AI on the other hand, such a form of AI is not widely used in a many games, far less in Roguelike games where content remains unchanging from when its generated. Though the AI developed within the project is simple and lacks great depth for what it is designed to do, it signifies that it can be developed and applied to a roguelike game in a way that doesn’t directly change static elements like the world shape or size, but can influence various internal values the change how the game plays. The significance of the projects AI would be much greater had there been more time and resources put into the AI. The projects primary disadvantage is lack of depth in the fields desired. Perhaps if only AI or only RNG was picked as the focal point of the project, more usable and significant information would be produced.

Though the project can serve to provide other benefits, less so from a research standpoint but perhaps from a development standpoint. The artefact produced by the project is a game by an ametuar, written from the ground up. Typically writing a game engine yourself is no easy task, as observed within the project. The fact this was done in this project can serve as an example to those intending to do the same. Though the engine may not be in depth or the most efficient, it does the tasks it's designed to do well. The project perhaps serves to contribute more to how to develop similar artefacts rather than the original premise of the project. Whether this is an advantage of disadvantage is up to the reader.

8.3 Professional, Ethical, Social, Security, and Legal Issues:

Addressing the various issues within the categories above now. With the given state of the project being a non-commercial artefact, the range of issues is very limited in what needs to be addressed.

Looking at professional issues, the primary issues in this area are to do with the design practice of the software in order to comply to various standards set out. Adhering to a standard was done to some level, various conventions involved with coding have been met for the most part but some sections deviate a bit at times. Given additional time to address, the artefact could be adjusted to comply with all conventions and standards being used.

With regards to ethical issues, no obvious ones were relevant to the project due to its nature. There was no inclusion of other persons nor the usage of content that could be deemed sensitive. The content of the project consisted of code and basic graphics. The graphics in question consisted of squares and rectangles. As such, there were no steps required to adhere to any ethical issues as none existed.

Social issues is similar to how ethical issues were handled. The project did not feature any content that involved others outside of usage of the artefact designed. In that regard, effort had to be made for the artefact to be prepared to be used by others. Otherwise though, no steps were taken to handle the non-existent social issues.

Security issues is a different category though, as the artefact is a piece of software, any and all measures must be taken into consideration whenever sensitive data is handled, such as a persons name, address, or other similar data. Should such data have been used by the system, then all the steps would have been taken. However, the artefact of the project uses primarily data and information represented and calculated internally. The only non-internal data required by the project is the clock time on the system. That form of data is openly accessible even within the c++ libraries so required no special handling to obtain. As a result, no special security steps were taken outside of any standard ones involved when writing software.

Finally with legal issues, the most obvious is the concern of copyright. A great deal was made to avoid using outside code or similar within the project and it was primarily written personally. Times when outside sources were used were specifically as and where math was involved that required a specific process. In most cases though, the math involved was well established methods or equations such as some physics equations used. The only other potential legal issues would arise if the artefact was designed for commercial distribution, though since at present times this is not the case, no special steps were required.

9.0 Conclusion:

The project at this stage achieved in developing a game that includes Random Number Generators as well as Artificial Intelligence into some of its core elements to improve the replay value of the game. Though the game is not as detailed it intended. Regardless, the implementation of the AI and RNG into the game achieves in displaying how said components can be implemented into a game to achieve the desired effect.

If this work were to be taken further, it would likely rely on the artefact itself. Primarily by continue development of the artefact while constantly improving and expanding on the RNG and AI components. Since the RNG and AI are somewhat restricted by the game elements, further game development will allow further development on the RNG and AI. Though it is not necessary to use the artefact produced by the project, using something similar is possible as long as the core principles of how the RNG and AI are implemented are similar. As long as what is being looked into constitutes a Roguelike Game, the results of the project can be expanded on by what is achieved with further development.

With regards to how the artefact was designed, the focus was in three parts, the game engine, the world generation in combination with random number generators, and the usage of Artificial intelligence on game elements. The focus on RNG and AI was decided to be a majority of the focus as they are two areas of interest in computing as well as their wide usage in games already. Though the choice could have been made to pick one over the other. It was decided to use RNG with the world generation specifically because this was already a common practice when it came to games, especially roguelike games. Procedural generation of content has been a part of various games in all forms and this project is no different. When it came to the AI, the choice was made to focus more on AI that influences the game rather than AI of entities within the game. Entities within the game can be simple and rely only on a state machine, as they do in the project. AI that can influence the game however, is more complex and more interesting to look into and develop. At multiple stages, the form of AI intended for the project had changed due to either complexity issues or lack of resources. The usage of the Director AI was not initially decided until later research into AI prompted the idea to implement such an AI.

The third focus of the Game engine was perhaps more of a special goal to improve coding skills and learning techniques to achieve various effects. It was entirely possibly to use one of the well established game engines out there such as Unreal or Unity. It was decided though to write the engine from the base up. Though it contributes less to the objectives of the project itself, it contributes to other objectives regarding coding as a whole. Looking back over the project, It may have been beneficial to use said engines in order to provide more development time into the AI and RNG. Though it is too late to even consider such an action. Should the project development be done again from the ground up, it would be decided to use a established game engine instead.

Looking back to the aims of the project, there are gaps in research which have resulted from lack of development time. The project does achieve in addressing the aim of the project though there is room for further research. In general, the project was able to achieve the desired objectives to a basic level. There is still room to further achieve the objective of the project and can be achieved with additional development and time.

10.0 Appendices:

Appendix A:

References

Ankur, Divyanjali and Pareek, V. 'A New Approach to Pseudorandom Number Generation'. *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, 8-9 Feb. 2014, 290-295.

Cellar Door Games, 2013, *Rogue Legacy*, video game, Cellar Door Games

J. Lan, W. L. Goh, Z. H. Kong, and K. S. Yeo, "A random number generator for low power cryptographic application," *2010 International SoC Design Conference*, 2010, 328-331.

J. Togelius and G. N. Yannakakis, "General general game AI," *2016 IEEE Conference on Computational Intelligence and Games (CIG)*,2016, 1-8.

Lait, J. (2008). *The Berlin Interpretation*. [online] Groups.google.com. Available at: https://groups.google.com/forum/#!msg/rec.games.roguelike.development/Orq2_7HhMjI/8DYjoMPBeF4J [Accessed 5 Feb. 2019].

L. Milinković, M. Antić, and Z. Čiča, "Pseudo-random number generator based on irrational numbers," *2011 10th International Conference on Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS)*, 2011, 719-722.

M. O. Riedl and A. Zook, "AI for game production," *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013, 1-8.

Mega Crit Games, 2019, *Slay the Spire*, video game, Mega Crit Games

Motion Twin, 2018, *Dead Cells*, video game, Motion Twin

Solovay, A. (July 27, 1993). *RFD: rec.games.dungeon.* hierarchy*. [online] Groups.google.com. Available at: https://groups.google.com/forum/#!topic/news.groups/CdWOd-M6g-w%5B1-25%5D [Accessed 5 Feb. 2019].

Stephen K. Park; Keith W. Miller. "Random Number Generators: Good Ones Are Hard To Find" *Communications of the ACM*. 1988, 31 (10): 1192–1201.

Valve Corporation, 2008, Left 4 Dead, video game, Valve Corporation

Appendix B:

Detailed Project Proposal

Section 1: Defining the Project

1.1 Detailed Research question/problem

In what ways can the use of random number generation combined with game AI affect the replayability of a roguelike game. Using random number generation combined with numerous algorithms within the game to generate the game world in different ways for each time played, allowing multiple playthroughs to feel and play differently from one another. Combined with AI aimed at a game world to control characters within the game to add a new and repeatable element to the game world for players to experience multiple times over.

1.2 Keywords

Game; Random Number Generation(RNG); Tree/Game AI;

1.3 Project Title

The usage of Procedural Generation (RNG) and game AI to develop gameplay with the focus of replayability.

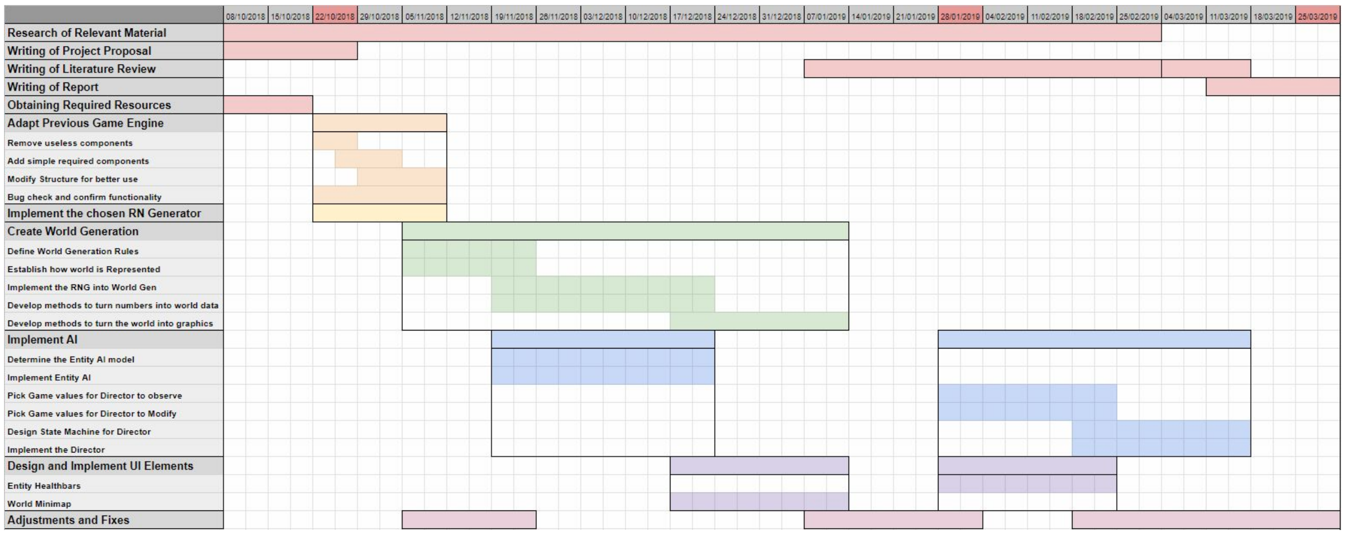
1.4 Client, Audience, and Motivation

For the project, I would say that in the area of Client/Audience, it is somewhat limited due to the nature of it being the development of a game. In the scheme of a game, there would likely be a player base or a group of people who would actively play the game. In this case, they would be the target audience of the project outcome (a piece of software). Beyond this scope, there is a limitation to who the client/audience may be. For the project matter itself, such as research and observed data or similar, such information may prove beneficial to those in the realm of developing games within the same genre. I would believe this to be unreliable to consider a client as the potential outcome of information can be very limited as it is not the immediate focus of the project.

Beyond the scope of those who would use the software/game, it is possible to consider employers as people of great interest in regards to the piece of software. For instance, it could be considered as a representation of skill to employers, providing clear evidence that one would have the experience and knowledge in the field. While a potential area to consider, it is perhaps more reliant on the process and success of the development of the software in question. Though one would hope this stage of the project goes well as it makes a large portion of the outcome.

As for motivation for the project, it is primarily a personal desire to develop a game of some form of another. As a person who plays a large number of games and enjoys writing software, it seemed logical to combine the two into something I would likely enjoy more. Beyond this, I had desired to focus my project on software more than theoretical research as my strong points primarily lie in software and code. Other than these points, it was to get experience writing software in a more semi-serious environment with deadlines.

1.5 Project plan



Section 2: Mini Literature review

2.1 Abstract

The roguelike genre of games is the focus on games in which a player is put into a procedural generation environment with 1 life to play for as long or as far as they are able to. In the process, the player would receive upgrades of punishments based on their decisions within the game world. These elements are the common basis for a roguelike game. This project will be looking into ways in which the procedural generation and the ai within the game can be developed and integrated in order to allow replayability of the game much like other games in the same genre. This will be accomplished by the actual development of a game using the information in question. Said information will be gathered from games within the genre to expand knowledge of key components of them and overlapping ideas to better understand what makes the game replayable and makes the player keep trying again and again.

2.2 Initial/Mini lit review (500 words max)

The focus of the project is the development of a game which features procedural generation and AI to create a interactive game environment. This being the basis, additional research will be done in the area of roguelike games to understand what makes these types of games successful and replayable (a key component of roguelike games). With these being the key details of the project, an effort has been made into the research of random number generation (RNG) and AI, primarily aimed at games, in order to provide a base for the development of the project.

RNG is a key feature within roguelike games and is a key component to the replayability by providing an element(s) of the game that changes constantly which each attempt to play. The process of doing so allows the player to interact with a different game world in each playthrough. This being the case, the methodology used to create the ‘key’ or ‘seed’ for the algorithms of the game generation is important to create a constant level of RNG to avoid a player seeing similar aspects over and over again. Two papers stood out greatly in this area of preliminary research. The paper by Ankur, Divyanjali and Pareek, V which looks at an algorithm and process of creating a random number generation with a larger focus on universal applicability rather than focusing on cryptographic generation. I deemed this paper to be valuable as it provided a solid example of an algorithm capable of generating pseudorandom numbers. On the flip side, the paper by Yang, Y. and Guang, Z provided a different method/algorithm into the matter. The combination of these two papers allowed for a better understanding of the workings of these kinds of methods. The 2nd paper, in particular, focused on a more complex method as it had an application into cryptography. Of course, the first paper is largely more relevant towards the project, the inclusion of the 2nd paper is to provide additional information into how pseudorandom numbers are generated in other environments.

The other field of research I went into was the area of AI, particularly game AI and similar areas. The first paper by Riedl, M. O. and Zook, A looked at the more theoretical aspect and more about objectives and notions to do with AI and games. While valuable as a base understanding of AI, a more practical example would provide better information. This is where the paper by Pepels, T., Winands, M. H. M. and Lanctot, M came into being. This paper looks directly at the Monte Carlo Tree within Ms Pac Man. This paper provided a larger deal of useful information compared to the first as it put forward a explanation of a well-known game. While the 1st paper is indeed still useful into the base information regarding ai in this field, the actual explanation of a functioning ai in the 2nd paper was deemed vastly more important.

2.3 Relevant Professional/Social/Ethical/Legal/Security issues

Professional:

Professional issues within the project will primarily consist of keeping to professional standards of software development and documentation. Though the specific standard to be used throughout the project is still somewhat undecided, I will likely be using the ISO or the IEEE standards for the project. Once the appropriate standard has been decided, all proper steps will be taken throughout the project in order to follow the standards and maintain a professional level of work throughout.

Social:

The social implications of the project are small if any. The premise of the project is the development of a roguelike game, as such, the social aspect is incredibly limited. While this would be an area of interest if the game were multiplayer or similar, the current plan behind it limits it to a singleplayer experience. There is likely to be no social aspects of the game within the scope of the project. However, in the event, the development continues on after the project and the scope expands, there are areas in which social implications are important to take into consideration. But within the scope of the project, the plans behind the project do not indicate any area for social issues to arise.

Ethical:

The premise of ethical issues is deciding between right and wrong within a situation or on a topic. In regards to the project, such a situation will unlikely occur on a level that would pose an issue to the general populace. Though the player will be presented choices within the game, these are only limited to statistical gain within the game world and are in no way connected to any potential topics that pose an ethical dilemma.

Also important to take into consideration is the content of the game and whether it is acceptable to the general populace. This should not pose a problem as the current plans of the project do not include any potential images or ideas that would be deemed unethical in the eyes of others. It will follow a standard game development practice and will not be stretching out of typical boundaries for indie developed games.

This all being the case, there are no ethical issues with the project at the current point in time. Should this change, it will be documented with the date of change and the rationality behind said change.

Legal:

The legal area surrounding the project primarily involve copyright and the various laws associated with it. One of the key premises of copyright in software is the expression of an idea rather than the idea itself. This would mean that the usage of a copyrighted methodology of an idea is prohibited by law given that it does not fall into public use.

The project, while researching into two key areas of software, RNG and AI, both fields consist of widely accepted practices which can be adapted and used in software as seen fit. Of course, there are exceptions to this but the research material related to the project is limited to methodology or ideas for use by others. Despite this information, attempts will be made to develop my own ideas and applications of an idea as much as physically possible in order to reduce any potential issues arising with copyright.

It is also important to note that the current state of the project is limited to the scope of the project and not to outside distribution. This provides a safer environment to handle copyright as in most cases, no real issues will arise as the project may not necessarily be made publically available. This is still no excuse to ignore copyright and all practices will be made in order to adhere to laws.

Security:

Security for the project is essentially covered in the BCS Code of Practice and the Code of Practice for Information Security Management. As such, all possible steps will be taken in order to maintain software development to the standards put out in said documents.

These details being the case, the nature of the project does not inherently require much in terms of security. As the goal of the project is the development of a game, there is no current need to obtain and use private information from a user. Though this is the case at the present time, should such information be used at a later stage of development, it will be taken into consideration and appropriate security measures will be applied to maintain standards.

Otherwise, the project is no demanding on the security front and as such, no extra work will be taken to make the software more secure than it has to be within the scope of data and information it uses.

2.4 Bibliography

Ankur, Divyanjali and Pareek, V. 'A New Approach to Pseudorandom Number Generation'. *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, 8-9 Feb. 2014, 290-295.

Browne, C. and Maire, F. (2010) 'Evolutionary Game Design', *IEEE Transactions on Computational Intelligence and AI in Games,* 2(1), pp. 1-16. doi: 10.1109/TCIAIG.2010.2041928.

Gabriela, M. (2016) 'Quantum random number generator vs. random number generator', pp.

Liu, Z., Huang, M. and Zhu, S. 'The Design and Implementation of a Pseudo Random Number Generation Algorithm'. *2009 International Conference on Computational Intelligence and Natural Computing*, 6-7 June 2009, 126-129.

Marghescu, A., Svasta, P. and Simion, E. 'High speed and secure variable probability Pseudo/True Random Number Generator using FPGA'. *2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 22-25 Oct. 2015, 323-328.

Meliones, A. and Plas, I. 'Developing video games with elementary adaptive artificial intelligence in unity: An intelligent systems approach'. *2017 Intelligent Systems Conference (IntelliSys)*, 7-8 Sept. 2017, 104-111.

Mogos, G. 'Quantum random number generator vs. random number generator'. *2016 International Conference on Communications (COMM)*, 9-10 June 2016, 423-426.

Pepels, T., Winands, M. H. M. and Lanctot, M. (2014) 'Real-Time Monte Carlo Tree Search in Ms Pac-Man', *IEEE Transactions on Computational Intelligence and AI in Games,* 6(3), pp. 245-257. doi: 10.1109/TCIAIG.2013.2291577.

Riedl, M. O. and Zook, A. 'AI for game production'. *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 11-13 Aug. 2013, 1-8.

Togelius, J. and Yannakakis, G. N. 'General general game AI'. *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 20-23 Sept. 2016, 1-8.

Yang, Y. and Guang, Z. 'A New Type of Random Number Generator for Software Implementation'. *2008 International Symposium on Information Science and Engineering*, 20-22 Dec. 2008, 236-238.

Appendix C:

Presentation Slides with Screenshots of Demo

Appendix D:

Project Log

🟩 Completed	🟨 Ongoing	🟧 Progress but Incomplete	🟥 No Progress

Weeks	**Tasks Todo**	**Done**
08/10/18 - 22/10/18	- Research and Find papers about the topic of the project.	🟨
	Begin research into papers and other resources about areas of interesting in regards to the project. This likely will include papers and resources on Artificial Intelligence as well as Random Number Generators.	
	- Write out the Detailed Project Proposal.	🟩
	Using the resources and plans at this time, write out a project proposal to provide a decent direction and plan for the project as a whole.	
	- Obtain software and libraries required for the development of the project.	🟩
	Determine if any special libraries of software are required to be able to develop the project and obtain them through legal means, though if any special resources are required, searches will be made in order to find free versions.	
22/10/18 - 05/11/18	- Adapt the prewritten game engine developed previously to accommodate the design intended for the project.	🟩
	Using a game engine previously developed in my free time, adapt and improve it in order to be able to be used for the project. Potentially scrapping and/or rewriting entire sections if required.	

	- Pick and Implement a Random Number Algorithm to provide the basis of the world generation.	🟩
	Ideally should be done as part of the process of picking an algorithm in order to test its effectiveness and its difficulty to implement as well as obtaining test runs of the algorithm.	
	- Establish and determine how the game world will be represented and rules to decide how to generate it.	🟩
	Decide what game elements the world will include such as items, enemies, etc, and come up with rules regarding how elements within the game are generated, such as only 1 item per tile, or no enemies near spawn.	
	- Modify existing Entity classes in the engine to remove useless attributes and add relevant ones.	🟩
	Using the Entity class hierarchy from the previously developed game engine, tinker with it to only contain the attributes and methods that are relevant to this project in particular. Any useless components are to be removed as well as useless classes.	
05/11/18 - 19/11/18	- Design world representation objects and methods to be able to be used when world generation algorithms are designed.	🟩
	This is the actual development and implementation of the world in a form usable by the software that includes all the attributes it requires. This representation won't be used until the world gen algorithms are done.	
	- Translate world generation rules and attributes into algorithms using text representation while the actual world is being developed.	🟩
	Writing and developing the world generation algorithms as according to the rules decided previously. The algorithms will work with text representations of the world since the actual representations are not ready at this point.	
	- Confirm that all changes to the game engine have been completed and are functioning correctly as they were before.	🟩
	Primarily confirming that the engine and its components function as they did prior to the adjustments made for the project. This includes the physics engine as well as controls.	

19/11/18 - 17/12/18	- Using the resources and papers researched, begin writing a proper literature review as well as an introduction for the project.	🟨
	In order to keep up with what is required from the report, ample time here is given before the deadline to write this part.	
	- Map the world representation onto the world generation algorithms and use a text output of the world representation to confirm it works.	🟩
	With world representation and world gen algorithms done from the previous time period, now it is mapping the world representation onto the world gen instead of the textual representation. This involves adjusting the specific methods and processes the algorithms use to turn the calculations into world elements but the world gen algorithms should produce the same exact output with proper world representation as compared to the previous text form. The world representation also needs to be able to be in a text form to confirm the above.	
	- Start development of a basic State machine AI for the enemy entities within the world.	🟩
	Mostly will consist of deciding how the AI will act and implementing a basic state machine to cover the states. May not necessarily have any AI components to cover actual interaction with other entities such as the player but will cover basic motor functionality.	
	- Address the bugs that occur within the physics engine while also implemented additional collision algorithms to handle collision between different types of entities.	🟧
	A Number of bugs exist from the physics engine, primarily concerning the collision of entities. This will be where those bugs are addressed and hopefully fixed. One example bug is when a collision occurs on the perfect diagonal of another entity, allowing clipping into the entity. The other task is to address the fact that an animate entity colliding with an inanimate wall will react differently than an animate entity colliding with an animate entity. As such, different methods will be added to accommodate the difference.	
17/12/18 - 07/01/19	- Develop and implement methods and functionality to translate the world representation into the graphical output.	🟩
	Using the world representation as generated from the world gen algorithms, translate the information generated into graphical representation in the form of entities types with their respective locations and sizes. Once the world is translated into entities, already defined methods that translate entities into graphics can be	

	used.	
	- Implement health functionality into entities as well as the interaction between entities. (specifically player x enemy)	🟩
	This handles with the notion that enemies and the player take damage when they come into contact with one another. Implementing the health into the base entity which can be adjusted based on collision detection between entities.	
	- Develop and implement methods that translate the world representation into a separate graphical output that can be adjusted.	🟩
	Writing methods that will take the graphical output of the world as defined above and allow a duplicate of the output to be modified to represent the entire map in a smaller area (minimap)	
07/01/19 - 28/01/19	- Using the resources and papers researched, begin writing a proper literature review as well as an introduction for the project.	🟩
	In order to keep up with what is required from the report, ample time here is given before the deadline to write this part. This is merely just writing the required documents as specified to the best of the current ability.	
	- Make improvements to the current stage of development primarily focusing on improvement of functionality and fixing of bugs.	🟩
	Primarily addressing how some functions work to either improve or change due to requiring different functionality. As well as attempting to handle some of the various bugs that arose from previous stages of development.	
28/01/19 - 18/02/19	- Design the implementation for a Director AI as well as determine what game elements it will observe and modify.	🟩
	Before implementing the AI, a focus will be made into determining how the AI will act and react according to what it observes. This also involves determining what values will influence it. Some minor code will be involved mostly surrounding the gathering of the observable data as well as the values that will be modified.	
	- Improve and rewrite the previous literature review and introduction based on feedback.	🟨
	This will be an ongoing task to constantly improve the literature	

	review and introduction in anticipation of putting them in the final report.	
	- Implement a visual representation of the health of entities within the game window so players can understand what's going on.	🟩
	Involves implementing a new class that will represent the health bar of a given entity. The methods of visually representing these will be built into it and the visuals will adjust based on the actual health of the entity its representing.	
	- Add a method in which the player can attack and damage enemy entities without coming in direct contact.	🟩
	Implementing a special hitbox that reacts to the player's mouse so that a special form of collision can be detected and handled that represents the player hitting the enemy. This is not the same as other collisions and requires its own method since the hitbox is unique.	
18/02/19 - 11/03/19	- Improve and rewrite the previous literature review and introduction based on feedback.	🟨
	This will be an ongoing task to constantly improve the literature review and introduction in anticipation of putting them in the final report.	
	- Implement the Director AI with a state machine and automatic modification of values.	🟩
	The Director will consist of a state machine that determines its mood. The state will be determined by weighted chance using the observable game elements to adjust the probability of a given mood. The mood will then effect to what degree the Director adjusts game values. The Director will adjust game values based on various conditions using game values such as health percentage, the time taken, etc.	
	- Make any more major changes that are deemed required during this time period.	🟩
	Make any changes to the system that requires large amounts of work or major changes to the code. This will be the last point to make such major changes since the remaining time period will be used to address the report and writing it properly with a state of the system that is mostly finished to the state it can be.	
11/03/19 - 22/03/19	- Compile and Right the final report.	🟩

	Gather all the appropriate papers as well as documentation about the project into one document, including source code and other stuff. Then Begin writing of the report according to the the specifications provided.	
	- Handle minor adjustments to areas within the system without drastically changing major components.	🟩
	This time period is primarily to handle small changes that need to be made and refinement of what is currently implemented. No major additions or changes will be made at this point that would require high amounts of additional workings to accommodate within the system and report. Essentially fine tuning of what is developed.	

Appendix E:

Ethics Form

TDE Form E1

OXFORD BROOKES UNIVERSITY

Faculty of Technology, Design and Environment - Ethics Review Form E1

- This form should be completed jointly by the **Supervisor and Student** who is undertaking a research/major project which involves human participants.
- It is the **Supervisor** who is responsible for exercising appropriate professional judgement in this review.
- Before completing this form, please refer to the University **Code of Practice for the Ethical Standards for Research involving Human Participants**, available at http://www.brookes.ac.uk/Research/Research-ethics/ and to any guidelines provided by relevant academic or professional associations.
- Note that the ethics review process needs to fully completed and signed **before fieldwork commences**.

(i) **Project Title: Usage of Random Number Generators and Artificial Intelligence to improve replayability of a Roguelike Game**

(ii) **Name of Supervisor and School in which located: Clare Martin (Wheatley Campus)**

(iii) **Name of Student and Student Number: Oscar Witney (16063222)**

(iv) **Brief description of project outlining where human participants will be involved (30-50 words):**

		Yes	**No**
1.	Does the study involve participants who are unable to give informed consent (e.g. children, people with learning disabilities)?	☐	☐
2.	If the study will involve participants who are unable to give informed consent (e.g. children under the age of 18, people with learning disabilities), will you be unable to obtain permission from their parents or guardians (as appropriate)?	☐	☐
3.	Will the study require the cooperation of a gatekeeper for initial access to groups or individuals to be recruited (e.g. students, members of a self-help group, employees of a company)?	☐	☐
4.	Are there any problems with the participants' right to remain anonymous, or to have the information they give not identifiable as theirs?	☐	☐
5.	Will it be necessary for the participants to take part in the study without their knowledge/consent at the time? (e.g. covert observation of people in non-public places?)	☐	☐
6.	Will the study involve discussion of or responses to questions the participants might find sensitive? (e.g. own traumatic experiences)	☐	☐
7.	Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?	☐	☐

8.	Will blood or tissue samples be obtained from participants?	☐	☐
9.	Is pain or more than mild discomfort likely to result from the study?	☐	☐
10.	Could the study induce psychological stress or anxiety?	☐	☐
11.	Will the study involve prolonged or repetitive testing of participants?	☐	☐
12.	Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?	☐	☐
13.	Will deception of participants be necessary during the study?	☐	☐
14.	Will the study involve NHS patients, staff, carers or premises?	☐	☐

Signed:		**Supervisor**
Signed:		**Student**
Date:		

What to do now:

1. If you have answered **'no'** to all the above questions:

   (a) The student must **send** the completed and fully signed E1 form to their **Dissertation Module Leader.**
   (b) The student must keep a copy of the E1 form which must be bound into their dissertation as an appendix.
   (c) The supervisor must keep a copy of the E1 form as they are responsible for monitoring compliance during the fieldwork.

2. If you have answered **'yes'** to **any** of the above questions:

   (a) The supervisor and student must complete the TDE E2 form available at http://www.brookes.ac.uk/Research/Research-ethics/Ethics-review-forms/
   (b) Note that the information in the E2 must be in **sufficient detail** for the ethical implications to be clearly identified.
   (c) The signed E2 and signed E1 Form must be emailed to Bridget Durning (bdurning@brookes.ac.uk) who is the Faculty Research Ethics Officer (FREO) for review. Please allow **at least two weeks** for this review process.
   (d) If/when approved the FREO will issue an E3 Ethics Approval Notice.
   (e) The student must send the E1, E2 and E3 Notice **to the Dissertation Module Leader**.
   (f) The student must also keep copies which must be bound into their dissertation as an appendix.
   (g) The supervisor must keep a copy of documentation to monitor compliance during field work.

3. If you answered 'yes' to any of questions 1-13 and 'yes' to question 14, an application must be submitted to the appropriate NHS research ethics committee. This is an onerous and time consuming process so the supervisor should liaise early with the FREO if the student is considering this.

Appendix F:

Source Code

Stdafx.h

```
#pragma once

#include <stdio.h>
#include <tchar.h>
#include <iostream>
#include <SFML/Graphics.hpp>
```

Stdafx.cpp

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

Main.cpp

```
#include "stdafx.h"
#include "core.h"
#include "SFML\Graphics.hpp"

sf::RenderWindow window(sf::VideoMode(window_width, window_height), "Window");

int main()
{
	window.setPosition(sf::Vector2i(20, 20));
	window.setFramerateLimit(30);
	window.requestFocus();

	initialise();
	while (window.isOpen())
	{
		sf::Event event;
		while (window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
				window.close();
			if (event.type == sf::Event::Resized)
			{
				// update the view to the new size of the window
				sf::FloatRect visibleArea(0.f, 0.f, event.size.width,
event.size.height);
				window.setView(sf::View(visibleArea));
			}
		}
		if (window.hasFocus())
		{
			window.clear();

			online(event, window);

			window.display();
		}

	}
	return 0;
}
```

<u>Entity.h</u>

```
#ifndef ENTITY_H
#define ENTITY_H

#include "stdafx.h"

class entity : public sf::Drawable
{
public:
	entity();
	entity(sf::Vector2f l_in, sf::Vector2f s_in);

	sf::Vector2f getLocation();
	void setLocation(sf::Vector2f l_in);
	void changeLocation(sf::Vector2f l_in);

	sf::Vector2f getSize();
	void setSize(sf::Vector2f s_in);
	void changeSize(sf::Vector2f s_in);

	sf::Vector2f getVelocity();
	void setVelocity(sf::Vector2f v_in);
	void changeVelocity(sf::Vector2f v_in);

	int getMaxHealth();
	void setMaxHealth(int h_in);
	int getHealth();
	void setHealth(int h_in);
	void changeHealth(int h_in);

	sf::Color getColor();
	void setColor(sf::Color c_in);
```

```
		void update(sf::Event& e, sf::RenderWindow& window);
		void draw(sf::RenderTarget & target, sf::RenderStates states) const;

		sf::RectangleShape getShape();
private:
		sf::Vector2f location;
		sf::Vector2f size;
		sf::Vector2f velocity;

		int maxHealth;
		int health;

		sf::Color color;
		sf::RectangleShape shape;


};

#endif
```

Entity.cpp

```
#include "entity.h"

entity::entity()
{
}

entity::entity(sf::Vector2f l_in, sf::Vector2f s_in)
{
		location = l_in;
		size = s_in;
		health = -1;
		velocity = sf::Vector2f(0, 0);
		color = sf::Color::White;
}

sf::Vector2f entity::getLocation()
{
		return location;
}

void entity::setLocation(sf::Vector2f l_in)
{
		location = l_in;
}
```

```
void entity::changeLocation(sf::Vector2f l_in)
{
	location += l_in;
}

sf::Vector2f entity::getSize()
{
	return size;
}

void entity::setSize(sf::Vector2f s_in)
{
	size = s_in;
}

void entity::changeSize(sf::Vector2f s_in)
{
	size += s_in;
}

sf::Vector2f entity::getVelocity()
{
	return velocity;
}

void entity::setVelocity(sf::Vector2f v_in)
{
	velocity = v_in;
}

void entity::changeVelocity(sf::Vector2f v_in)
{
	velocity += v_in;
}

int entity::getMaxHealth()
{
	return maxHealth;
}

void entity::setMaxHealth(int h_in)
{
	maxHealth = h_in;
	health = maxHealth;
}
```

```
int entity::getHealth()
{
	return health;
}

void entity::setHealth(int h_in)
{
	health = h_in;
}

void entity::changeHealth(int h_in)
{
	health += h_in;
}

sf::Color entity::getColor()
{
	return color;
}

void entity::setColor(sf::Color c_in)
{
	color = c_in;
}

void entity::update(sf::Event & e, sf::RenderWindow & window)
{
	changeLocation(getVelocity());
	shape.setPosition(getLocation());
	shape.setSize(getSize());
	shape.setFillColor(getColor());
}

sf::RectangleShape entity::getShape()
{
	return shape;
}

void entity::draw(sf::RenderTarget & target, sf::RenderStates states) const
{
	target.draw(shape, states);
}
```

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H
```

```
#include "entity.h"
#include <time.h>
#include "healthbars.h"

class player : public entity
{
public:
	player();
	player(sf::Vector2f l_in, sf::Vector2f s_in);

	void invulnerable();
	bool isInvuln();

	void update(sf::Event& e, sf::RenderWindow& window);
private:
	healthbars myHealth;
	clock_t thisTime;
	double counter = 0;
	bool invuln;

	void draw(sf::RenderTarget & target, sf::RenderStates states) const;
};

#endif
```

Player.cpp

```
#include "player.h"

player::player()
{
	invuln = false;
}

player::player(sf::Vector2f l_in, sf::Vector2f s_in) : entity(l_in,s_in)
{
	myHealth = healthbars();
	setMaxHealth(100);
	setHealth(getMaxHealth());
	setColor(sf::Color::Green);
	myHealth.setMaxHealth(getMaxHealth());
	myHealth.setCurrentHealth(getMaxHealth());
	invuln = false;
}

void player::invulnerable()
```

```
{
	if (!invuln)
	{
		invuln = true;
		thisTime = clock();
	}
	else if (counter > (double)(2 * CLOCKS_PER_SEC))
	{
		invuln = false;
		counter = 0;
	}
	else
	{
		counter = (double)(clock() - thisTime);
	}
}

bool player::isInvuln()
{
	return invuln;
}

void player::update(sf::Event & e, sf::RenderWindow & window)
{
	entity::update(e, window);
	myHealth.setLocation(sf::Vector2f(30,window.getSize().y - 70));
	myHealth.setSize(sf::Vector2f(400,40));
	//myHealth.setMaxHealth(getMaxHealth());
	myHealth.setCurrentHealth(getHealth());
	myHealth.update(e, window);
}

void player::draw(sf::RenderTarget & target, sf::RenderStates states) const
{
	entity::draw(target, states);
	myHealth.draw(target, states);
}
```

Enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H

#include "entity.h"
#include "player.h"
#include <time.h>
#include "healthbars.h"
```

```
class enemy : public entity
{
public:
	enemy();
	enemy(sf::Vector2f l_in, sf::Vector2f s_in);

	void stun();
	bool isStunned();

	healthbars getHealthBar();
	void update(sf::Event& e, sf::RenderWindow& window);
private:
	healthbars myHealth;
	clock_t thisTime;
	double counter = 0;
	bool stunned;

	void draw(sf::RenderTarget & target, sf::RenderStates states) const;
};

#endif
```

Enemy.cpp

```
#include "enemy.h"

enemy::enemy()
{
}

enemy::enemy(sf::Vector2f l_in, sf::Vector2f s_in) : entity(l_in,s_in)
{
	myHealth = healthbars();
	setColor(sf::Color::Red);
	setMaxHealth(50);
	setHealth(getMaxHealth());
	myHealth.setMaxHealth(getMaxHealth());
	myHealth.setCurrentHealth(getMaxHealth());
	stunned = false;
}

void enemy::stun()
{
	//std::cout << "CALLED" << std::endl;
	if (!stunned)
	{
			stunned = true;
			thisTime = clock();
		}
		else if (counter > (double)(1.5 * CLOCKS_PER_SEC))
		{
			stunned = false;
			counter = 0;
		}
		else
		{
			counter = (double)(clock() - thisTime);
		}
}

bool enemy::isStunned()
{
		return stunned;
}

healthbars enemy::getHealthBar()
{
		return myHealth;
}

void enemy::update(sf::Event & e, sf::RenderWindow & window)
{
		entity::update(e, window);
		myHealth.setLocation(sf::Vector2f(getLocation().x - 5, getLocation().y - 15));
		myHealth.setSize(sf::Vector2f(getSize().x + 10, 10));
		myHealth.setMaxHealth(getMaxHealth());
		myHealth.setCurrentHealth(getHealth());
		myHealth.update(e, window);
}

void enemy::draw(sf::RenderTarget & target, sf::RenderStates states) const
{
		entity::draw(target, states);
		myHealth.draw(target, states);
}
```

Terrain.h

```
#ifndef TERRAIN_H
#define TERRAIN_H

#include "entity.h"
```

```
class terrain : public entity
{
public:
	terrain();
	terrain(sf::Vector2f l_in, sf::Vector2f s_in);
private:
};

#endif
```

Terrain.cpp

```
#include "terrain.h"

terrain::terrain()
{
}

terrain::terrain(sf::Vector2f l_in, sf::Vector2f s_in) : entity(l_in,s_in)
{
	setColor(sf::Color::White);
}
```

Tile.h

```
#ifndef TILE_H
#define TILE_H

#include "stdafx.h"
#include "terrain.h"
#include "enemy.h"
#include <iostream>

class tile
{
public:
	tile();
	tile(int t_in);
	tile(int t_in, sf::Vector2f l_in, int tl_in);

	void setTitle(int t_in);
	int getTitle();

	void setLocation(sf::Vector2f l_in);
	void changeLocation(sf::Vector2f l_in);
	sf::Vector2f getLocation();

	void setWalls(std::string w_in);
	std::string getWalls();

	void setEnemyCount(int e_in);
	int getEnemyCount();

	void setTileLayout(int tl_in);
	int getTileLayout();

	std::vector<terrain> getTerrain();
	std::vector<enemy> getEnemies();

	void createTile();

	std::string textOutput();
private:
	int title;
	sf::Vector2f location;
	std::string walls;
	int enemies;
	int tile_layout;

	std::vector<terrain> tile_terrain;
	std::vector<enemy> tile_enemies;
};

#endif
```

Tile.cpp

```
#include "tile.h"

tile::tile()
{
}

tile::tile(int t_in)
{
	title = t_in;
	location = sf::Vector2f(0, 0);
	tile_layout = 0;
	enemies = 0;
	walls = "----";
}

tile::tile(int t_in, sf::Vector2f l_in, int tl_in)
{
	title = t_in;
	location = l_in;
	tile_layout = tl_in;
	enemies = 0;
	walls = "----";
}

void tile::setTitle(int t_in)
{
	title = t_in;
}

int tile::getTitle()
{
	return title;
}

void tile::setLocation(sf::Vector2f l_in)
{
	location = l_in;
}

void tile::changeLocation(sf::Vector2f l_in)
{
	location += l_in;
}

sf::Vector2f tile::getLocation()
{
	return location;
}

void tile::setWalls(std::string w_in)
{
	walls = w_in;
}

std::string tile::getWalls()
{
	return walls;
}

void tile::setEnemyCount(int e_in)
{
	enemies = e_in;
}
```

```
int tile::getEnemyCount()
{
	return enemies;
}

void tile::setTileLayout(int tl_in)
{
	tile_layout = tl_in;
}

int tile::getTileLayout()
{
	return tile_layout;
}

std::vector<terrain> tile::getTerrain()
{
	return tile_terrain;
}

std::vector<enemy> tile::getEnemies()
{
	return tile_enemies;
}

void tile::createTile()
{
	//Tiles are assumed to be 360x240
	tile_terrain.clear();
	tile_enemies.clear();

	if (getWalls().find("N") != std::string::npos)
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x, getLocation().y -
240), sf::Vector2f(360, 240)));
	if (getWalls().find("S") != std::string::npos)
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x, getLocation().y +
240), sf::Vector2f(360, 240)));
	if (getWalls().find("E") != std::string::npos)
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 360,
getLocation().y), sf::Vector2f(360, 240)));
	if (getWalls().find("W") != std::string::npos)
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x - 360,
getLocation().y), sf::Vector2f(360, 240)));

	switch (tile_layout)
	{
```

```
	case 1:
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 40, 
getLocation().y + 20), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 200, 
getLocation().y + 40), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 100, 
getLocation().y + 120), sf::Vector2f(60, 60)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 220, 
getLocation().y + 160), sf::Vector2f(60, 40)));
		switch (enemies)
		{
		case 5:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 300, 
getLocation().y + 140), sf::Vector2f(20, 20)));
		case 4:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 260, 
getLocation().y + 80), sf::Vector2f(20, 20)));
		case 3:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 40, 
getLocation().y + 180), sf::Vector2f(20, 20)));
		case 2:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 120, 
getLocation().y + 40), sf::Vector2f(20, 20)));
		case 1:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 220, 
getLocation().y + 120), sf::Vector2f(20, 20)));
		default:
			break;
		}
		break;
	case 2:
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 40), sf::Vector2f(60, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 160, 
getLocation().y + 40), sf::Vector2f(120, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 60), sf::Vector2f(20, 60)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 160), sf::Vector2f(40, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 160, 
getLocation().y + 160), sf::Vector2f(40, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 220, 
getLocation().y + 160), sf::Vector2f(60, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 140), sf::Vector2f(20, 20)));
```

```
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 260, 
getLocation().y + 140), sf::Vector2f(20, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 260, 
getLocation().y + 60), sf::Vector2f(20, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 300, 
getLocation().y + 80), sf::Vector2f(20, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 40, 
getLocation().y + 220), sf::Vector2f(20, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 20, 
getLocation().y + 20), sf::Vector2f(20, 20)));
			switch (enemies)
			{
			case 5:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 120, 
getLocation().y + 200), sf::Vector2f(20, 20)));
			case 4:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 120, 
getLocation().y + 80), sf::Vector2f(20, 20)));
			case 3:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 300, 
getLocation().y + 140), sf::Vector2f(20, 20)));
			case 2:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 320, 
getLocation().y + 40), sf::Vector2f(20, 20)));
			case 1:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 220, 
getLocation().y + 120), sf::Vector2f(20, 20)));
			default:
				break;
			}
			break;
		case 3:
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 60), sf::Vector2f(40, 40)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 100, 
getLocation().y + 80), sf::Vector2f(40, 40)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 300, 
getLocation().y + 20), sf::Vector2f(40, 40)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 220, 
getLocation().y + 140), sf::Vector2f(40, 40)));
			switch (enemies)
			{
			case 5:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 200), sf::Vector2f(20, 20)));
			case 4:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 40, 
getLocation().y + 160), sf::Vector2f(20, 20)));
			case 3:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 240, 
getLocation().y + 80), sf::Vector2f(20, 20)));
			case 2:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 180, 
getLocation().y + 40), sf::Vector2f(20, 20)));
			case 1:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 160, 
getLocation().y + 120), sf::Vector2f(20, 20)));
			default:
				break;
			}
			break;
		case 4:
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 140, 
getLocation().y + 20), sf::Vector2f(40, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 100, 
getLocation().y + 40), sf::Vector2f(100, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 60, 
getLocation().y + 60), sf::Vector2f(160, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 40, 
getLocation().y + 80), sf::Vector2f(180, 40)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80, 
getLocation().y + 120), sf::Vector2f(80, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 260, 
getLocation().y + 160), sf::Vector2f(40, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 220, 
getLocation().y + 180), sf::Vector2f(100, 20)));
			tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 240, 
getLocation().y + 200), sf::Vector2f(60, 20)));
			switch (enemies)
			{
			case 5:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 280, 
getLocation().y + 100), sf::Vector2f(20, 20)));
			case 4:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 260, 
getLocation().y + 60), sf::Vector2f(20, 20)));
			case 3:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 160, 
getLocation().y + 180), sf::Vector2f(20, 20)));
			case 2:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 100, 
getLocation().y + 180), sf::Vector2f(20, 20)));
```

```
		case 1:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 60,
getLocation().y + 160), sf::Vector2f(20, 20)));
		default:
			break;
		}
		break;
	case 5:
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 20,
getLocation().y + 20), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80,
getLocation().y + 40), sf::Vector2f(80, 60)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 200,
getLocation().y + 20), sf::Vector2f(20, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 240,
getLocation().y + 40), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 300,
getLocation().y + 80), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 0, getLocation().y
+ 120), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 100,
getLocation().y + 140), sf::Vector2f(40, 20)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 200,
getLocation().y + 140), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 260,
getLocation().y + 160), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 80,
getLocation().y + 180), sf::Vector2f(40, 40)));
		tile_terrain.push_back(terrain(sf::Vector2f(getLocation().x + 160,
getLocation().y + 200), sf::Vector2f(40, 40)));
		switch (enemies)
		{
		case 5:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 40,
getLocation().y + 80), sf::Vector2f(20, 20)));
		case 4:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 220,
getLocation().y + 200), sf::Vector2f(20, 20)));
		case 3:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 260,
getLocation().y + 100), sf::Vector2f(20, 20)));
		case 2:
			tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 180,
getLocation().y + 80), sf::Vector2f(20, 20)));
		case 1:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 40,
getLocation().y + 180), sf::Vector2f(20, 20)));
			default:
				break;
			}
			break;
		default:
			switch (enemies)
			{
			case 5:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 220,
getLocation().y + 80), sf::Vector2f(20, 20)));
			case 4:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 60,
getLocation().y + 160), sf::Vector2f(20, 20)));
			case 3:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 120,
getLocation().y + 60), sf::Vector2f(20, 20)));
			case 2:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 240,
getLocation().y + 180), sf::Vector2f(20, 20)));
			case 1:
				tile_enemies.push_back(enemy(sf::Vector2f(getLocation().x + 160,
getLocation().y + 120), sf::Vector2f(20, 20)));
			default:
				break;
			}
			break;
		}
}

std::string tile::textOutput()
{
		std::string output = "[ ";
		output += getWalls() + " ";
		if (enemies == 0)
			output += "- ";
		else
			output += std::to_string(enemies) + " ";
		output += std::to_string(tile_layout) + " ]";
		return output;
}
```

World.h

```
#ifndef WORLD_H
#define WORLD_H
```

```
#include "stdafx.h"
#include "tile.h"

extern tile world[11][11];

static int scrollWidth = sizeof(world) / sizeof(world[0]);
static int scrollHeight = sizeof(world[0]) / sizeof(tile);

int pMiller(int seed_in);

void generate();

void createMiddle();

int calculateTiles(int prev);
void createNorth();
void createSouth();

std::string getXDir(int i, int j);
std::string getYDir(int i, int j);

void assignDir();
void assignEnemies();
void assignLayout();

void fillTiles();

void textWorld();

void clearWorld();

#endif
```

World.cpp

```
#include "world.h"
#include <chrono>

tile world[11][11];


int base_seed = std::abs(std::chrono::duration_cast<std::chrono::milliseconds>(
	std::chrono::system_clock::now().time_since_epoch()).count()) % 1048576;
int a = std::abs(std::chrono::duration_cast<std::chrono::milliseconds>(
	std::chrono::system_clock::now().time_since_epoch()).count()) % 500;
int m = 2147483647;
int q = m / a;
int r = m % a;

int seed = pMiller(base_seed);
int n_tiles = 35;
int s_tiles = 35;

int pMiller(int seed_in)
{
	int lo, hi, test;

	hi = seed_in / q;
	lo = seed_in % q;
	test = a * lo - r * hi;
	if (test > 0)
		seed_in = test;
	else
		seed_in = test + m;
	return seed_in;
}

void generate()
{
	clearWorld();
	std::cout << "[ NOTICE ] Generate Called" << std::endl;
	while (n_tiles + s_tiles > 20)
	{
		n_tiles = 35;
		s_tiles = 35;
		createMiddle();
		createSouth();
		createNorth();
	}
	std::cout << "[ NOTICE ] While Loop Finished" << std::endl;
	assignDir();
	std::cout << "[ NOTICE ] assignDir() Finished" << std::endl;
	assignEnemies();
	std::cout << "[ NOTICE ] assignEnemies() Finished" << std::endl;
	assignLayout();
	std::cout << "[ NOTICE ] assignLayout() Finished" << std::endl;
	fillTiles();
	std::cout << "[ NOTICE ] fillTiles() Finished" << std::endl;
	textWorld();
}

void createMiddle()
```

```
{
	for (int i = 0; i < scrollWidth; i++)
	{
		for (int j = 0; j < scrollHeight; j++)
		{
			if (i == scrollWidth / 2)
				world[scrollHeight / 2][j] = tile(1);
			else
				world[i][j] = tile(0);
		}
	}
}

int calculateTiles(int prev)
{
	int tiles;
	if (prev == -1)
	{
		tiles = seed % 5;
		prev = tiles;
		return tiles;
	}
	else
	{
		seed = pMiller(seed);
		tiles = 2 - (seed % 4);
		tiles = prev + tiles;
		if (tiles > scrollHeight / 2)
			tiles = scrollHeight / 2;
		if (tiles < 0)
			tiles = 0;
		prev = tiles;
		return tiles;
	}
}

void createNorth()
{
	int prev_UP = -1;
	for (int i = 0; i < scrollWidth; i++)
	{
		int tiles = calculateTiles(prev_UP);
		prev_UP = tiles;
		for (int j = scrollHeight / 2 - 1; j > 0; j--)
		{
			if (tiles != 0)
```

```
                {
                    if (n_tiles > 0)
                    {
                        world[j][i] = tile(1);
                        tiles--;
                        n_tiles--;
                    }
                }
            }
        }
}

void createSouth()
{
        int prev_UP = -1;
        for (int i = 0; i < scrollWidth; i++)
        {
            int tiles = calculateTiles(prev_UP);
            prev_UP = tiles;
            for (int j = scrollHeight / 2 + 1; j < scrollHeight; j++)
            {
                if (tiles != 0)
                {
                    if (s_tiles > 0)
                    {
                        world[j][i] = tile(1);
                        tiles--;
                        s_tiles--;
                    }
                }
            }
        }
}

std::string getXDir(int i, int j)
{
        std::string dir = "";
        if (world[i][j + 1].getTitle() == 0 || j == 10)
            dir += "E";
        else
            dir += "-";
        if (world[i][j - 1].getTitle() == 0 || j == 0)
            dir += "W";
        else
            dir += "-";
        return dir;
```

```
}

std::string getYDir(int i, int j)
{
	std::string dir = "";
	if (world[i - 1][j].getTitle() == 0 || i == 0)
		dir += "N";
	else
		dir += "-";
	if (world[i + 1][j].getTitle() == 0 || i == 10)
		dir += "S";
	else
		dir += "-";

	return dir;
}

void assignDir()
{
	for (int i = 0; i < scrollHeight; i++)
	{
		for (int j = 0; j < scrollWidth; j++)
		{
			if (world[i][j].getTitle() == 1)
			{
				world[i][j].setWalls(getYDir(i, j) + getXDir(i, j));
			}
		}
	}
}

void assignEnemies()
{
	int count = 15;
	//std::cout << "Assign Enemies Started" << std::endl;
	while (count > 0)
	{
		//std::cout << "While Loop Run" << std::endl;
		int x = pMiller(seed) % 11;
		int y = pMiller(pMiller(seed)) % 11;
		seed = pMiller(pMiller(seed));

		if (y >= 8 || y <= 2 || x >= 3)
		{
			//std::cout << "First Condition met" << std::endl;
			if (world[y][x].getTitle() == 1)
```

```
                        {
                                //std::cout << "Second Condition met" << std::endl;
                                if (world[y - 1][x].getEnemyCount() == 0 && world[y +
1][x].getEnemyCount() == 0
                                        && world[y][x - 1].getEnemyCount() == 0 && world[y][x
+ 1].getEnemyCount() == 0
                                        && world[y][x].getEnemyCount() == 0)
                                {
                                        //std::cout << "Third condition Met" << std::endl;
                                        int enemies = 1 + (pMiller(seed) % 5);
                                        world[y][x].setEnemyCount(enemies);
                                        count--;
                                }
                        }
                }
        }
}

void assignLayout()
{
        //ADD IN LIMITATIONS TO AVOID THE SAME TILE BEING NEXT TO THE SAME
TILE EXCEPT IN THE CASE OF BLANK
        seed = pMiller(pMiller(seed));
        for (int i = 0; i < scrollHeight; i++)
        {
                for (int j = 0; j < scrollWidth; j++)
                {
                        if (world[i][j].getTitle() == 1)
                        {
                                seed = pMiller(seed);
                                if (i == 5 && (j == 0 || j == 10))
                                {
                                        world[i][j].setTileLayout(9);
                                }
                                else if(world[i - 1][j].getTileLayout() == 5 ||
                                        world[i][j - 1].getTileLayout() == 5)
                                {
                                        if (pMiller(seed) % 8 > 4)
                                                world[i][j].setTileLayout(5);
                                        else
                                                world[i][j].setTileLayout(pMiller(seed) % 9 + 1);
                                }
                                else
                                        world[i][j].setTileLayout(pMiller(seed) % 9 + 1);
                        }
                }
```

```
	}
}

void fillTiles()
{
	for (int i = 0; i < scrollHeight; i++)
	{
		for (int j = 0; j < scrollWidth; j++)
		{
			if (world[i][j].getTitle() == 1)
			{
				//EDIT IF WORLD SIZE CHANGES
				world[i][j].setLocation(sf::Vector2f(360 * (j+2), 240 * (i-4)));
				world[i][j].createTile();
			}
		}
	}
}

void textWorld()
{
	std::cout << "[  INFO  ]  Seed: " << base_seed << "\n[  INFO  ] Alpha: " << a <<
std::endl;
	std::cout << "    0  1  2  3  4  5  6  7  8  9  10" << std::endl;
	for (int i = 0; i < scrollHeight; i++)
	{
		if (i == 10)
			std::cout << i << " ";
		else
			std::cout << i << "  ";
		for (int j = 0; j < scrollWidth; j++)
		{
			if (world[i][j].getTitle() == 0)
				std::cout << "[-]";
			else
				std::cout << "[X]";
		}
		std::cout << "\n";
	}
	std::cout << std::endl;
	for (int i = 0; i < scrollHeight; i++)
	{
		for (int j = 0; j < scrollWidth; j++)
		{
			if (world[i][j].getTitle() == 0)
				std::cout << "[ -------- ]";
```

```
					else
					{
						std::cout << world[i][j].textOutput();
					}
				}
				std::cout << "\n";
		}
}

void clearWorld()
{
		for (int i = 0; i < scrollHeight; i++)
		{
				for (int j = 0; j < scrollWidth; j++)
				{
						world[i][j] = NULL;
				}
		}
		n_tiles = 35;
		s_tiles = 35;
}
```

Healthbars.h

```
#ifndef HEALTHBARS_H
#define HEALTHBARS_H

#include "stdafx.h"

class healthbars :public sf::Drawable
{
public:
		healthbars();
		healthbars(sf::Vector2f s_in, sf::Vector2f l_in);
		void setMaxHealth(int h_in);
		void setCurrentHealth(int h_in);
		void changeCurrentHealth(int h_in);

		void setSize(sf::Vector2f s_in);
		void setLocation(sf::Vector2f l_in);

		sf::Vector2f getSize();
		sf::Vector2f getLocation();

		int getMaxHealth();
		int getCurrentHealth();
```

```
        void update(sf::Event& e, sf::RenderWindow& window);
        void draw(sf::RenderTarget & target, sf::RenderStates states) const;
private:
        int currentHealth;
        int maxHealth;

        sf::Vector2f size;
        sf::Vector2f location;

        sf::RectangleShape health;
        sf::RectangleShape container;
};

#endif
```

Healthbars.cpp

```
#include "healthbars.h"

healthbars::healthbars()
{
        maxHealth = 1;
}

healthbars::healthbars(sf::Vector2f s_in, sf::Vector2f l_in)
{
        size = s_in;
        location = l_in;
        maxHealth = 1;
}


void healthbars::setMaxHealth(int h_in)
{
        maxHealth = h_in;
}

void healthbars::setCurrentHealth(int h_in)
{
        currentHealth = h_in;
}

void healthbars::changeCurrentHealth(int h_in)
{
        currentHealth += h_in;
}
```

```
void healthbars::setSize(sf::Vector2f s_in)
{
	size = s_in;
}

void healthbars::setLocation(sf::Vector2f l_in)
{
	location = l_in;
}

sf::Vector2f healthbars::getSize()
{
	return size;
}

sf::Vector2f healthbars::getLocation()
{
	return location;
}

int healthbars::getMaxHealth()
{
	return maxHealth;
}

int healthbars::getCurrentHealth()
{
	return currentHealth;
}

void healthbars::update(sf::Event & e, sf::RenderWindow & window)
{
	container.setFillColor(sf::Color::White);
	container.setSize(size);
	container.setPosition(location);
	health.setFillColor(sf::Color::Green);
	health.setSize(sf::Vector2f(size.x * (currentHealth/ (double)(maxHealth)),size.y));
	health.setPosition(location);
}

void healthbars::draw(sf::RenderTarget & target, sf::RenderStates states) const
{
	target.draw(container, states);
	target.draw(health, states);
}
```

Core.h

```
#ifndef CORE_H
#define CORE_H

#include "stdafx.h"
#include "control.h"
#include "physics.h"
#include "entity.h"
#include "player.h"
#include "terrain.h"
#include "enemy.h"
#include "world.h"
#include "director.h"

extern int window_width;
extern int window_height;

extern sf::RectangleShape entryPoint;
extern sf::RectangleShape exitPoint;

extern player _player;
extern std::vector<terrain> walls;
extern std::vector<enemy> enemies;

extern entity mapP;
void initialise();
void regen();
void online(sf::Event& e, sf::RenderWindow& window);

void extractItems();
void drawEntities(sf::Event& e, sf::RenderWindow& window);
void scrollScreen();

void attack(double rotation);

#endif
```

Core.cpp

```
#include "core.h"

int window_width = 1560;
int window_height = 720;
std::vector<terrain> walls;
std::vector<enemy> enemies;
std::vector<entity> map;
```

```
player _player = player(sf::Vector2f(window_width / 2, window_height / 2), sf::Vector2f(20,
20));

sf::RectangleShape attackBox;
sf::RectangleShape entryPoint;
sf::RectangleShape exitPoint;

entity mapP = entity();

int enemyCount;

void initialise()
{
	mapP.setSize(_player.getSize() * 0.15f);
	mapP.setColor(sf::Color::Green);
	giveModifiers();
	generate();
	extractItems();
	enemyCount = enemies.size();
	attackBox.setSize(sf::Vector2f(25, 40));
	attackBox.setOrigin(-5.f, 20.f);
	attackBox.setFillColor(sf::Color::Transparent);
	attackBox.setPosition(window_width / 2, window_height / 2);

}

void regen()
{
	walls.clear();
	map.clear();

	modify(_player, enemies, enemyCount);
	enemies.clear();
	giveModifiers();
	generate();
	extractItems();
	enemyCount = enemies.size();
}

void online(sf::Event& e, sf::RenderWindow& window)
{
	mousePress(e,window);
	moveEntities(e, window);
	modifySpeedADP(_player, enemies);
	changeState(_player, enemies, enemyCount);
	drawEntities(e, window);
```

```
		exitLevel();
}

void extractItems()
{
		for (int i = 0; i < scrollHeight; i++)
		{
				for (int j = 0; j < scrollWidth; j++)
				{
						if (world[i][j].getTitle() == 1)
						{
								if (i == 5 && j == 0)
								{
										entryPoint.setPosition(sf::Vector2f(360 * (j + 2.05), 240
* (i - 3.666)));
										entryPoint.setSize(sf::Vector2f(80, 80));
										entryPoint.setFillColor(sf::Color::Magenta);

								}
								if (i == 5 && j == 10)
								{
										exitPoint.setPosition(sf::Vector2f(360 * (j + 2.5), 240 * (i
- 3.666)));
										exitPoint.setSize(sf::Vector2f(80, 80));
										exitPoint.setFillColor(sf::Color::Blue);
								}
								//for (terrain& t : world[i][j].getTerrain())

								for (terrain& t : world[i][j].getTerrain())
								{
										walls.push_back(t);
										t.setSize(t.getSize()*0.075f);
										t.setColor(sf::Color::Blue);
										t.setLocation(t.getLocation() * 0.075f);
										t.setLocation(sf::Vector2f(t.getLocation().x -
27,t.getLocation().y + (18*5)));
										if (i == 5 && j == 0)
												mapP.setLocation(sf::Vector2f(t.getLocation().x
+ (t.getSize().x *1.15) - (mapP.getSize().x /2),
														t.getLocation().y+ (t.getSize().y/2) -
(mapP.getSize().y / 2)));
										map.push_back(t);
								}
								for (enemy& e : world[i][j].getEnemies())
								{
										e.setMaxHealth(e.getMaxHealth() * healthModifier);
```

```
					enemies.push_back(e);
				}

			}
		}
	}
}

void drawEntities(sf::Event& e, sf::RenderWindow& window)
{
	window.draw(entryPoint);
	window.draw(exitPoint);
	for (terrain& _i : walls)
	{
		_i.update(e, window);
		window.draw(_i);
	}
	for (enemy& _a : enemies)
	{
		_a.update(e, window);
		window.draw(_a);
	}

	_player.update(e, window);

	window.draw(attackBox);
	window.draw(_player);

	for (entity& _e : map)
	{
		_e.update(e, window);
		window.draw(_e);
	}
	mapP.update(e, window);
	window.draw(mapP);
	if (attackBox.getFillColor() == sf::Color::Cyan)
		attackBox.setFillColor(sf::Color::Transparent);

}

void scrollScreen()
{
	_player.setLocation(sf::Vector2f(window_width / 2 - _player.getSize().x / 2,
window_height / 2 - _player.getSize().y / 2));
	for (entity& e : walls)
	{
			e.setLocation(sf::Vector2f(e.getLocation().x - _player.getVelocity().x, 
e.getLocation().y - _player.getVelocity().y));
		}
		for (entity& e : enemies)
		{
			e.setLocation(sf::Vector2f(e.getLocation().x - _player.getVelocity().x, 
e.getLocation().y - _player.getVelocity().y));
		}
		entryPoint.setPosition(sf::Vector2f(entryPoint.getPosition().x - _player.getVelocity().x, 
entryPoint.getPosition().y - _player.getVelocity().y));
		exitPoint.setPosition(sf::Vector2f(exitPoint.getPosition().x - _player.getVelocity().x, 
exitPoint.getPosition().y - _player.getVelocity().y));
		_player.setVelocity(sf::Vector2f(0, 0));
}

void attack(double rotation)
{
		attackBox.setRotation(rotation);
		attackBox.setFillColor(sf::Color::Cyan);
		for (enemy& e : enemies)
			hitCollision(attackBox, e);
}
```

Physics.h

```
#ifndef PHYSICS_H
#define PHYSICS_H

#include "stdafx.h"
#include "core.h"
#include "entity.h"
#include "enemy.h"
#include "baseai.h"

void moveEntities(sf::Event& e, sf::RenderWindow& window);
void inanimateCollision(entity& _a, entity& _e);
void animateCollision(entity& _a, enemy& _e);
void exitLevel();
void hitCollision(sf::RectangleShape _a, enemy& _e);

#endif
```

Physics.cpp

```
#include "physics.h"

double calcEnemyDist(enemy _e)
{
	int x = (_player.getLocation().x + _player.getSize().x /2) - (_e.getLocation().x +
_e.getSize().x/2);
	int y = (_player.getLocation().y + _player.getSize().y/2) - (_e.getLocation().y +
_e.getSize().y/2);
	// distance = sqrt((x2 - x1) ^ 2 + (y2 - y2) ^ 2)
	// velocity = spd/distance * (x2 - x1)
	// velocity = spd/distance * (y2 - y1)
	double d = sqrt((pow(x, 2)) + (pow(y, 2)));
	return d;
}

sf::Vector2f calcEnemyVel(enemy& _e)
{
	int x = (_player.getLocation().x + _player.getSize().x / 2) - (_e.getLocation().x +
_e.getSize().x / 2);
	int y = (_player.getLocation().y + _player.getSize().y / 2) - (_e.getLocation().y +
_e.getSize().y / 2);
	double d = sqrt((pow(x, 2)) + (pow(y, 2)));
	sf::Vector2f vel = (sf::Vector2f((5 / d)*x, (5 / d)*y));
	return vel;
}

void moveEntities(sf::Event& e, sf::RenderWindow& window)
{
	move();
	for (int i =0;i< enemies.size();i++)
	{
		enemy& _a = enemies[i];

		if (_a.getHealth() > 0)
		{
			setAI(_player, _a);
		}
		else
			enemies.erase(enemies.begin() + i);

		animateCollision(_player, _a);
		for (terrain& _i : walls)
		{
			inanimateCollision(_a, _i);
		}
	}

	for (terrain& _i : walls)
		inanimateCollision(_player, _i);
	mapP.setVelocity(_player.getVelocity() * 0.075f);
	if(_player.isInvuln())
		_player.invulnerable();
	scrollScreen();
}

void inanimateCollision(entity& _a, entity& _e)
{
	if (_a.getLocation().x + _a.getSize().x > _e.getLocation().x
		&& _a.getLocation().x < _e.getLocation().x + _e.getSize().x)
	{
		if (_a.getLocation().y + _a.getSize().y + _a.getVelocity().y >=
_e.getLocation().y
			&& _a.getLocation().y + _a.getVelocity().y <= _e.getLocation().y +
_e.getSize().y)
		{
			if (_a.getVelocity().y > 0)
			{
				_a.setVelocity(sf::Vector2f(_a.getVelocity().x,
_e.getLocation().y - (_a.getLocation().y + _a.getSize().y)));
			}
			if (_a.getVelocity().y < 0)
			{
				_a.setVelocity(sf::Vector2f(_a.getVelocity().x,
(_e.getLocation().y + _e.getSize().y) - _a.getLocation().y));
			}
		}
	}
	if (_a.getLocation().y + _a.getSize().y > _e.getLocation().y
		&& _a.getLocation().y < _e.getLocation().y + _e.getSize().y)
	{
		if (_a.getLocation().x + _a.getSize().x + _a.getVelocity().x > _e.getLocation().x
			&& _a.getLocation().x + _a.getVelocity().x <= _e.getLocation().x +
_e.getSize().x)
		{
			if (_a.getVelocity().x > 0)
			{
				_a.setVelocity(sf::Vector2f(_e.getLocation().x -
(_a.getLocation().x + _a.getSize().x), _a.getVelocity().y));
			}
			if (_a.getVelocity().x < 0)
			{
				_a.setVelocity(sf::Vector2f((_e.getLocation().x + _e.getSize().x)
- _a.getLocation().x, _a.getVelocity().y));
			}
		}
	}
	// COLLISION FOR THE DIAGONAL - ONLY WORKS ASSUMING SIZE OF ENTITY
IS DIVISIBLE BY VELOCITY OF PLAYER
	if (_a.getLocation().x + _a.getVelocity().x + _a.getSize().x > _e.getLocation().x
		&& _a.getLocation().x + _a.getVelocity().x < _e.getLocation().x +
_e.getSize().x
		&& _a.getLocation().y + _a.getVelocity().y + _a.getSize().y >
_e.getLocation().y
		&& _a.getLocation().y + _a.getVelocity().y < _e.getLocation().y +
_e.getSize().y)
	{
		_a.setVelocity(sf::Vector2f(0, 0));
	}
}

void animateCollision(entity& _a, enemy& _e)
{
	if (_a.getLocation() == _e.getLocation())
	{
		_e.setHealth(0);
	}
	if (_a.getShape().getGlobalBounds().intersects(_e.getShape().getGlobalBounds()))
	{
		std::cout << _player.isInvuln() << std::endl;
		if (!_player.isInvuln())
			_a.changeHealth(-10 * damageModifier);
		std::cout << _player.getHealth() << std::endl;
		if (_a.getHealth() <= 0)
			std::exit(0);
		_e.setVelocity(-(calcEnemyVel(_e)*2.5f));
		_e.stun();
		_player.invulnerable();
	}
}

void hitCollision(sf::RectangleShape _a, enemy& _e)
{
	sf::FloatRect box = _a.getGlobalBounds();
	if (box.intersects(_e.getShape().getGlobalBounds()) && !_e.isStunned())
	{
		_e.stun();
		_e.changeHealth(-10 * pDamageModifier);
		_e.setVelocity(-(calcEnemyVel(_e)*2.5f));
	}
}
```

```
void exitLevel()
{
	if(exitPoint.getGlobalBounds().intersects(_player.getShape().getGlobalBounds()))
		regen();
}
```

Control.h

```
#ifndef CONTROL_H
#define CONTROL_H

#include "stdafx.h"
#include "core.h"

static bool up, down, left, right, mouse;

void keyPress();
void move();
void mousePress(sf::Event& e, sf::RenderWindow& window);

#endif
```

Control.cpp

```
#include "control.h"

void keyPress()
{
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
		left = true;
	else
		left = false;
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))
		right = true;
	else
		right = false;
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))
		up = true;
	else
		up = false;
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::S))
		down = true;
	else
		down = false;
}

void move()
{
	int velocity = 10;
	keyPress();
	if (left && !right)
	{
		_player.setVelocity(sf::Vector2f(-velocity, _player.getVelocity().y));
	}
	else if (!left && right)
	{
		_player.setVelocity(sf::Vector2f(velocity, _player.getVelocity().y));
	}
	if ((left && right) || (!left && !right))
	{
		_player.setVelocity(sf::Vector2f(0, _player.getVelocity().y));
	}
	if (up && !down)
	{
		_player.setVelocity(sf::Vector2f(_player.getVelocity().x, -velocity));
	}
	else if (!up && down)
	{
		_player.setVelocity(sf::Vector2f(_player.getVelocity().x, velocity));
	}
	if ((up && down) || (!up && !down))
	{
		_player.setVelocity(sf::Vector2f(_player.getVelocity().x, 0));
	}
}

void mousePress(sf::Event& e, sf::RenderWindow& window)
{
	if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
	{
		double angle = atan2(sf::Mouse::getPosition(window).y -
(_player.getLocation().y + _player.getSize().y / 2),
			sf::Mouse::getPosition(window).x - (_player.getLocation().x +
_player.getSize().x / 2));
		attack(angle * 57.2958);
	}
}
```

Baseai.h

```
#ifndef BASEAI_H
#define BASEAI_H

#include "stdafx.h"
#include "player.h"
```

```
#include "entity.h"
#include "enemy.h"
#include "terrain.h"
#include <time.h>
#include "director.h"

extern clock_t thisTime;
extern double counter;

void setAI(player p, enemy& e);
void runIdle(player p, enemy& e);
void runAttack(player p, enemy& e);
void runFollow(player p, enemy& e);
void runFlee(player p, enemy& e);
void stunned(enemy& e);

#endif
```

Baseai.cpp

```
#include "baseai.h"

clock_t thisTime;
double counter = 0;
double baseSpeed = 5.0;
double enemySpeed = baseSpeed;

double calcEnemyDist(player _player, enemy _e)
{
	int x = (_player.getLocation().x + _player.getSize().x / 2) - (_e.getLocation().x +
_e.getSize().x / 2);
	int y = (_player.getLocation().y + _player.getSize().y / 2) - (_e.getLocation().y +
_e.getSize().y / 2);
	// distance = sqrt((x2 - x1) ^ 2 + (y2 - y2) ^ 2)
	// velocity = spd/distance * (x2 - x1)
	// velocity = spd/distance * (y2 - y1)
	double d = sqrt((pow(x, 2)) + (pow(y, 2)));
	return d;
}

sf::Vector2f calcEnemyVel(player _p, enemy& _e)
{
	enemySpeed = baseSpeed * speedModifier;
	int x = (_p.getLocation().x + _p.getSize().x / 2) - (_e.getLocation().x + _e.getSize().x /
2);
	int y = (_p.getLocation().y + _p.getSize().y / 2) - (_e.getLocation().y + _e.getSize().y /
2);
```

```
	double d = sqrt((pow(x, 2)) + (pow(y, 2)));
	sf::Vector2f vel = (sf::Vector2f((enemySpeed / d)*x, (enemySpeed / d)*y));
	return vel;
}

sf::Vector2f calcEnemyVel2(player _p, enemy& _e)
{
	enemySpeed = baseSpeed * speedModifier;
	double x1 = (_p.getLocation().x + _p.getSize().x / 2);
	double y1 = (_p.getLocation().y + _p.getSize().y / 2);
	double x2 = (_e.getLocation().x + _e.getSize().x / 2);
	double y2 = (_e.getLocation().y + _e.getSize().y / 2);

	double d = sqrt((pow(x1 - x2, 2)) + (pow(y1 - y2, 2)));

	//God I hate math
	sf::Vector2f temp = sf::Vector2f((x1 - x2), (y1 - y2));
	sf::Vector2f temp2 = sf::Vector2f(temp.x / d, temp.y / d);
	sf::Vector2f dist = temp2 * 100.f;
	sf::Vector2f temp3 = sf::Vector2f(x1, y1) - dist;
	double d3 = sqrt((pow(temp3.x - x2, 2)) + (pow(temp3.y - y2, 2)));
	sf::Vector2f vel = (sf::Vector2f((enemySpeed / d3)*(temp3.x - x2), (enemySpeed /
d3)*(temp3.y - y2)));
	return vel;
}


void setAI(player p, enemy& e)
{
	if (!e.isStunned())
	{
		if (e.getHealth() > e.getMaxHealth() * 0.5f)
		{
			if (calcEnemyDist(p, e) <= 200 * rangeModifier)
				runAttack(p, e);
			else
				runIdle(p, e);
		}
		else if (e.getHealth() >= e.getMaxHealth() * 0.25f && e.getHealth() <=
e.getMaxHealth() * 0.5f)
		{
			if (calcEnemyDist(p, e) <= 300 * rangeModifier)
				runFollow(p, e);
			else
				runIdle(p, e);
```

```
                }
                else if (e.getHealth() < e.getMaxHealth() * 0.25f)
                {
                        if (calcEnemyDist(p, e) <= 300 * rangeModifier)
                                runFlee(p, e);
                        else
                                runIdle(p, e);
                }
        }
        else if (e.isStunned())
                stunned(e);
        else
                runIdle(p, e);
}


void runIdle(player p, enemy& e)
{
        int val = rand() % 100 + 1;
        int yDir = (rand() % 3 - 1);
        int xDir = (rand() % 3 - 1);
        if (val == 1)
        {
                e.setVelocity(sf::Vector2f(enemySpeed * xDir, enemySpeed * yDir));
        }
        e.setVelocity(e.getVelocity()*0.8f);
}

void runAttack(player p, enemy & e)
{
        e.setVelocity(calcEnemyVel(p, e));
}

void runFollow(player p, enemy & e)
{
        if (calcEnemyDist(p, e) <= 95 || calcEnemyDist(p, e) >= 105)
                e.setVelocity(calcEnemyVel2(p, e));
        else
                e.setVelocity(sf::Vector2f(0, 0));
}

void runFlee(player p, enemy & e)
{
        e.setVelocity(-calcEnemyVel(p, e));
}
```

```
void stunned(enemy & e)
{
	e.stun();
	e.setVelocity(e.getVelocity()*0.8f);
}
```

Director.h

```
#ifndef DIRECTOR_H
#define DIRECTOR_H

#include "stdafx.h"
#include "player.h"
#include "enemy.h"
#include <time.h>

extern double healthModifier;
extern double damageModifier;
extern double speedModifier;
extern double rangeModifier;
extern double pDamageModifier;

void modify(player p, std::vector<enemy> e, int ec);
void changeState(player& p, std::vector<enemy> e, int eT);
void modifyHealth(player& p, std::vector<enemy> e);
void modifyDamage(player p, std::vector<enemy> e);
void modifySpeed(player p, std::vector<enemy> e);
void modifyRange(player p, std::vector<enemy> e);

void modifySpeedADP(player p, std::vector<enemy> e);
void modifyRangeADP(player p, std::vector<enemy> e);

void giveModifiers();
#endif
```

Director.cpp

```
#include "director.h"

double baseHealthM = 1.0;
double baseDamageM = 1.0;
double baseSpeedM = 1.0;
double baseRangeM = 1.0;
double basePDamageM = 1.0;
double healthModifier = baseHealthM;
double damageModifier = baseDamageM;
double speedModifier = baseSpeedM;
double rangeModifier = baseRangeM;
double pDamageModifier = basePDamageM;

int enemyTotal;
clock_t myTime;
double myCounter = 0;
clock_t stateTime;
double stateCounter = 0;
double stateModifier = 1.0;
bool stateLock = false;

double limitA = (double)(60 * CLOCKS_PER_SEC);
double limitB = (double)(120 * CLOCKS_PER_SEC);
double limitC = (double)(240 * CLOCKS_PER_SEC);
double limitD = (double)(300 * CLOCKS_PER_SEC);


void modify(player p, std::vector<enemy> e, int ec)
{
	enemyTotal = ec;
	modifyHealth(p, e);
	modifyDamage(p, e);
	modifySpeed(p, e);
	modifyRange(p, e);
	myTime = clock();
	myCounter = 0;
	stateCounter = 0;
	stateModifier = 1.0;
	std::cout << enemyTotal << std::endl;
}

void temp()
{
	srand(time(NULL));
}

bool t = false;

void changeState(player& p, std::vector<enemy> e, int eT)
{
	double a = p.getHealth();
	double b = p.getMaxHealth();
	double internalHModifier = a / b;
	double internalEModifier = e.size() / eT;

	if (!t)
	{
		temp();
			t = true;
	}

	if (!stateLock)
	{
		stateLock = true;
		stateTime = clock();
	}
	else if (stateCounter >= (double)(20 * CLOCKS_PER_SEC))
	{
		stateLock = false;
		stateCounter = 0;

		if (internalHModifier < 0.25)
			internalHModifier = 0.25;
		if (internalEModifier < 0.3)
			internalEModifier = 0.3;
		double annoyed = 20 * (0.5 / internalHModifier) * internalEModifier;
		double angry = 15 * (0.5 / internalHModifier) * internalEModifier;
		double forgiving = 20 * (1 / internalHModifier) * (0.1 / internalEModifier);
		double bloodLust = 5 * (0.5 / internalHModifier) * (0.5 / internalEModifier);
		double gentle = 20 * (0.5 / internalHModifier) * (0.25 / internalEModifier);

		double limitA = annoyed;
		double limitB = limitA + angry;
		double limitC = limitB + forgiving;
		double limitD = limitC + bloodLust;
		double limitE = limitD + gentle;
		int chance = rand() % 101;
		if (chance <= limitA)// Annoyed Stated
			stateModifier = 1.1;
		else if (chance > limitA && chance <= limitB)// Angry State
			stateModifier = 1.3;
		else if (chance > limitB && chance <= limitC)// Forgiving State
			stateModifier = 0.9;
		else if (chance > limitC && chance <= limitD)// Bloodlust State
			stateModifier = 2.0;
		else if (chance > limitD && chance <= limitE)// Gentle State
			stateModifier = 0.8;
		else// Neutral State
			stateModifier = 1.0;
		std::cout << "A: " << limitA << std::endl;
		std::cout << "B: " << limitB << std::endl;
		std::cout << "C: " << limitC << std::endl;
		std::cout << "D: " << limitD << std::endl;
		std::cout << "E: " << limitE << std::endl;
		std::cout << "CHANCE: " << chance << std::endl;
		std::cout << "MODIFIER: " << stateModifier << std::endl;
	}
	else
		stateCounter = (double)(clock() - stateTime);
}

double timeModifier()
{
	myCounter = (double)(clock() - myTime);
	if (myCounter <= limitD)
		if (myCounter <= limitC)
			if (myCounter <= limitB)
				if (myCounter <= limitA)
					return 1 + ((myCounter / limitA) * 0.3); // 1.0 - 1.3
				else
					return 1 + ((myCounter / limitB) * 0.6); // 1.3 - 1.6
			else
				return 1 + ((myCounter / limitC) * 1.2); // 1.6 - 2.2
		else
			return 2.2 * ((limitA + ((myCounter)-limitC)) / limitA); // 2.2 - 4.4
	else
		return 4.4;
}


void modifyHealth(player& p, std::vector<enemy> e)
{
	double a = p.getHealth();
	double b = p.getMaxHealth();
	double internalHModifier = a / b;
	double internalTModifier = timeModifier();
	double internalEModifier = e.size() / enemyTotal;
	baseHealthM += 0.5*internalHModifier * internalTModifier * internalEModifier;
	baseHealthM *= stateModifier;
	healthModifier = baseHealthM;


	double heal = 20 * ((1.1 - internalEModifier) / internalHModifier) * stateModifier;
	std::cout << "HEAL " << heal << std::endl;
	if (a + heal > 85)
		if (internalEModifier <= 0.3)
			if (a + heal > b)
				p.setHealth(b);
			else
```

```
				p.changeHealth(heal);
	else
		p.changeHealth(heal);
	std::cout << p.getHealth() << " " << p.getMaxHealth() << std::endl;
}

void modifyDamage(player p, std::vector<enemy> e)
{
	double a = p.getHealth();
	double b = p.getMaxHealth();
	double internalHModifier = a / b;
	double internalTModifier = timeModifier();
	double internalEModifier = e.size() / enemyTotal;
	baseDamageM += internalHModifier * internalEModifier * (internalTModifier / 4.4);
	baseDamageM *= stateModifier;
	damageModifier = baseDamageM;

	double temp = (0.3/internalHModifier) + (1/internalEModifier) - 1;
	if (temp < 0)
		basePDamageM += 0.25;
	else if (basePDamageM + temp > baseHealthM)
		basePDamageM = baseHealthM * 0.75;
	else
		basePDamageM += temp;
	basePDamageM *= stateModifier;
	pDamageModifier = basePDamageM;
}

void modifySpeed(player p, std::vector<enemy> e)
{
	baseSpeedM *= stateModifier;
	speedModifier = baseSpeedM;
}

void modifyRange(player p, std::vector<enemy> e)
{
	baseRangeM *= stateModifier;
	rangeModifier = baseSpeedM;
}

void modifySpeedADP(player p, std::vector<enemy> e)
{
	double a = p.getHealth();
	double b = p.getMaxHealth();
	double internalHModifier = a/b;
	double internalTModifier = timeModifier();
	if (internalHModifier >= 0.25)
	{
		double temp = a - (b * 0.25);
		double temp2 = temp - ((b * 0.75) / 2);
		double temp3 = std::abs(temp2) / ((b * 0.75) / 2);
		if(internalHModifier >= 3.0)
			speedModifier = baseSpeedM * (((2.0 / 3.0) * 3.0) + (temp3 / 3)) *
stateModifier;
		else
			speedModifier = baseSpeedM * ((2.0 / 3.0) + (temp3 / 3)) *
stateModifier;
	}
	else
	{
		double c = b * 0.25;
		speedModifier = baseSpeedM * (((c - (-c + a)) / c)) * stateModifier;
	}
	if (internalTModifier > 1.75)
		speedModifier *= 1.75;
	else
		speedModifier *= internalTModifier;

}

void modifyRangeADP(player p, std::vector<enemy> e)
{

}

void giveModifiers()
{
	std::cout << "[  INFO  ] Health Modifier: " << healthModifier << std::endl;
	std::cout << "[  INFO  ] Damage Modifier: " << damageModifier << std::endl;
	std::cout << "[  INFO  ] Player Modifier: " << pDamageModifier << std::endl;
	std::cout << "[  INFO  ] Speed Modifier: " << speedModifier << std::endl;
	std::cout << "[  INFO  ] Range Modifier: " << rangeModifier << std::endl;
}
```