31263 - Introduction to Computer Game Programming

Assessment 4 - Report

Group 18

Students:

* William Palmer - 12039499
* John Vitale - 11452594
* Raymond - 99149573

**Table Of Contents**

1 - 1 Title Page

2 - 2 Table of Contents

3 - 3 The Plan

3 - 5 Objectives

6 - 7 Procedural Generation

7 - 8 Gantt Chart

**The Plan**

We started out thinking around where we could go from a 3D point of view, and what we were beginning to learn in Unity. After consulting the intermediate and advanced topic lists, we threw around a few ideas, and eventually stuck on a challenge we thought would be interesting, but doable; a 3D roguelike. We’d implement some fps shooter traits to make the challenge for progressing through the levels, and generate the map randomly for variety on each play-through. Beyond that, we’d see what we can come up with and what we’re capable of as we progress through the subject; a few ideas on grappling hooks, alternate weapons, bosses, randomly selected map conditions were all bandied about, and will be considered further once we meet the assignment spec, and *then* we can have a crack at these stretch goals.

**Objectives**

The intermediate and advanced topics will be covered separately in the report.

***Movement*** - Players must be able to move through the levels. Our levels also feature obstacles, and may include some ground-based hazards, so players must also be able to jump. Movement should be quick enough that it's not a slog to progress through cleared areas, but should be slow enough that it's not hard to control, or make the levels feel tiny.

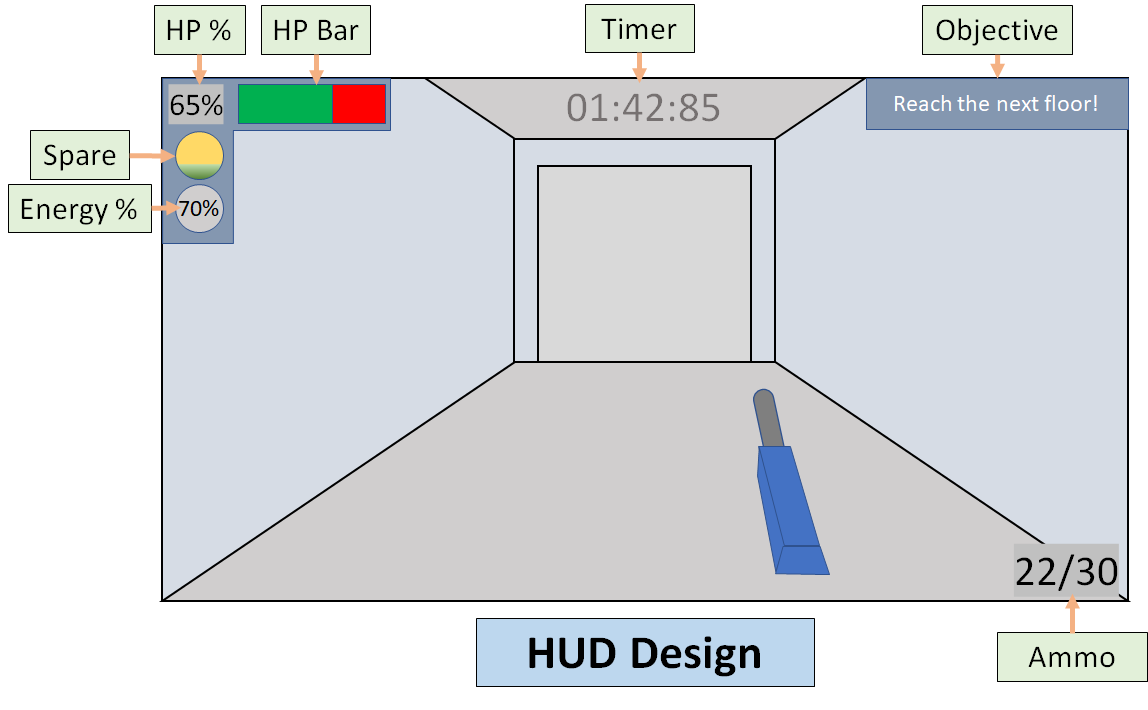
A sprint function that allows the player to move faster, but not shoot, would help satisfy both requirements; players can progress around explored portions of the level quickly, and use a slower speed to deal with enemies. This helps deal with inevitable backtracking that will occur from our procedural level generation.

***Combat*** - Players and enemies will have health, die when it reaches zero, and the player should have methods to replenish their health. The player starts with a ranged weapon which uses ammo and a simple, inexhaustible short-range tool for backup for being short on all ammo supplies. Health packs can be seeded in spawn points throughout floors, encouraging players to explore, and making dead ends feel rewarding instead of an unlucky waste of time. Similarly, ammo pickups can also be scattered using the same spawn points. Extra additions will be any and all weapons we can think of. We’re expecting some basic bullet firing weapons (a pistol, SMG, or rifle are all very similar but for a few tweaked numbers), and will be having grenades and a rocket launcher for certain. The grenade will probably utilise Unity’s physics engine. Other weapons we’re considering are shotguns, bouncing bullets, and a telefrag.

***Enemies (Intermediate Topic - Simple AI)*** - Will depend greatly on what we’re capable of coding in, but at minimum; we’re looking to create simple enemies that can shoot at the player, run to the player to attack them at melee range, and path to the player if they’ve seen the player, but can no longer attack them from where they are. Enemy attacks will need to have their own properties to support this, enemies should be clearly distinguishable from one another so the player can predict and plan for an attack, and enemies should not feel ‘unfair’ for the player; a good design is that a well-prepared and practised player should not take heavy damage from any enemy.

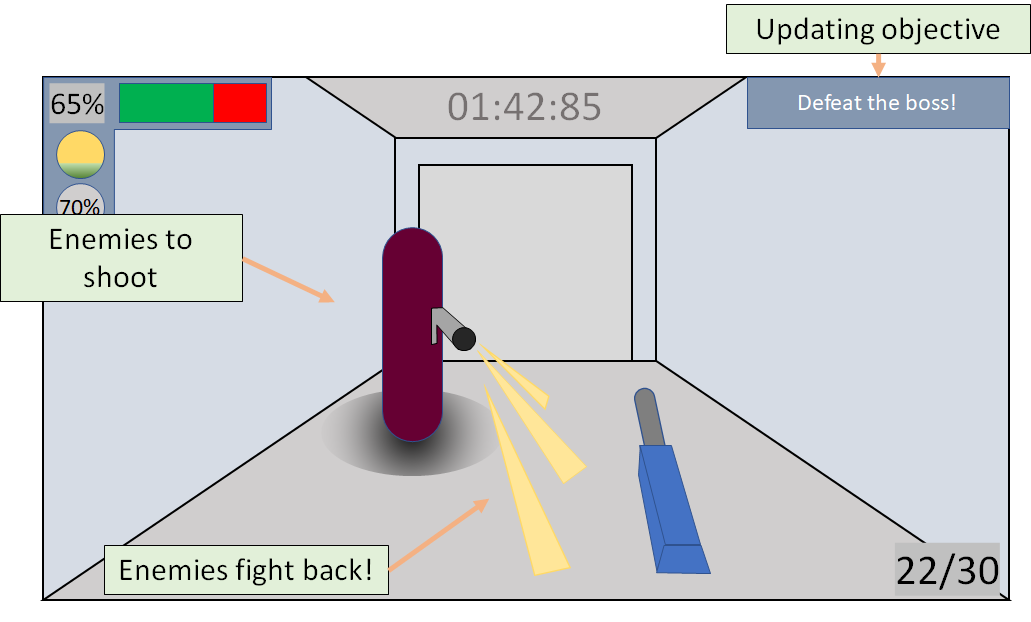
Stretch goals are mostly aimed around ‘boss’ style enemies that exhibit different, more challenging behaviour than the basic enemies. Possibilities for boss enemies include:

* A melee boss that charges faster than the player can retreat, but can only turn slowly, and can be sidestepped.
* A mine deployer which throws mines toward the player, punishing them for standing still, and pressuring them to plan where they walk as they bring it down.
* A sniper who deals heavy damage with a high-speed, but telegraphed attack, forcing the player fight between shots around cover.
* Turrets covering small rooms which may be brute-forced, but easily dealt with by indirect attacks like grenades



***HUD*** - The heads up display should show the player everything they need, in an unobtrusive way. This is why most designs have their info clinging to the edges of the screen, leaving the main view the game uninterrupted, save for a small and translucent firing reticle. Ours will be no exception; we may make some design adjustments depending on how we decide to do our weapons implementation (should we have a permanent name, a separate pop-up for changing weapons, or just rely on the weapon model as satisfactory indication?).

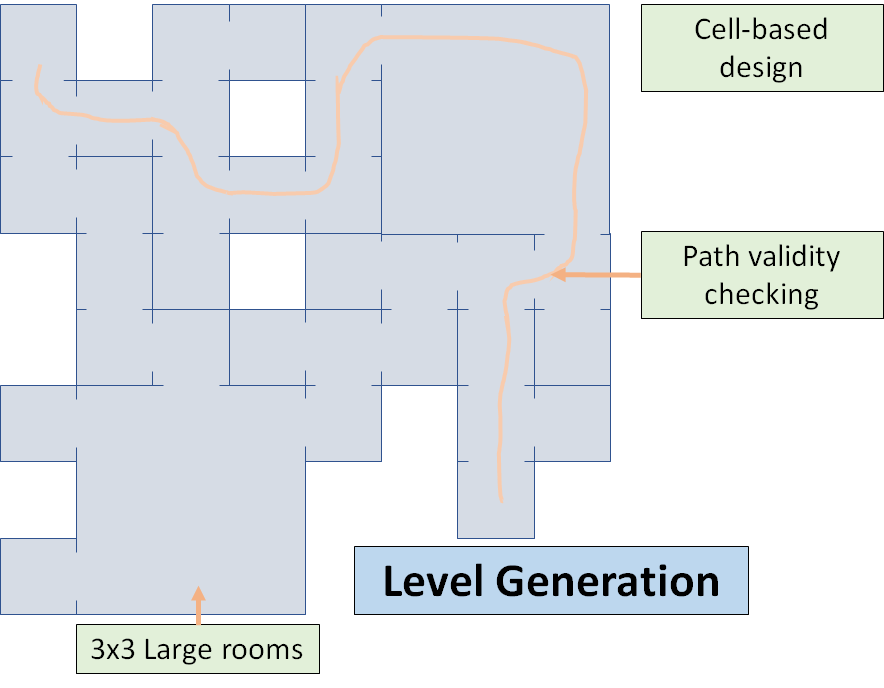
***Player Objective -*** Our current plan is for the player to advance through each floor, survive the obstacles within, ascend to the next one, and repeat until they reach the top of the stack. A timer will record how quickly this is accomplished (or when the player meets an untimely demise). Depending on how testing goes, we may decide to force the player to find and defeat bosses to activate the exit to the next floor, and the objective on-screen must change to inform the player of this.

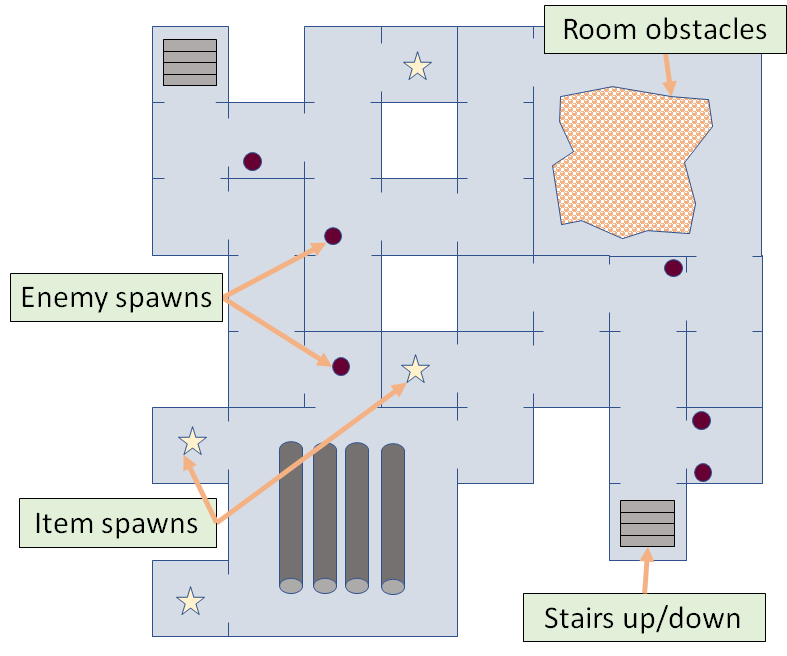


***Procedurally Generated Levels (Advanced Topic - Substantial Procedural Content Generation) -*** Upon calling the algorithm to generate the level, an array of integers is passed in to provide the number of rooms on a per floor basis, with the number of elements in the array being the number of floors. At first, the “map” algorithm generates a ten by ten by number of floors array of strings, before proceeding to set each of the strings in the three-dimensional array to a default value of the number zero. Starting from the height-wise bottom of the map, a random x and z set of coordinates are generated, with the corresponding position on the bottom floor being set to a generic room value, in our case the number one is used. From there, more random coordinates are generated, each of which are tested for x and/or z adjacency to any already placed “rooms”, until the number of rooms that can be generated for the floor has been met. From there, a set of coordinates are generated for the “stairs” in between the floors, except on the last floor, with the same restriction that the stairs must be generated adjacent to an already existing room. After the floor has been completed, we move up to the next floor, where this process repeats itself until each floor has been generated. Once this generation has been complete, a random set of x, y and z coordinates are generated, under the same ruleset of adjacency, to determine the starting location of the level.

For large rooms, a random number is generated to determine if a large room is to be generated. The equation used to justify the building of a large room is one in number of rooms the floor can have minus the number of rooms already generated plus two, such that a large room has at most a one in three chance of being generated. When a large room is being attempted to be generated, a set of x and z coordinates are generated, with a ruleset applying to all eight surrounding “tiles”. The first rule is that any of these eight surrounding tiles must be adjacent to already existing rooms. The second rule is that none of the nine tiles required would, upon selection, overwrite already existing rooms. Thus, simultaneous overlap and adjacency tests are performed on each of the surrounding eight tiles, with only a overlap test being performed on the center tile. Once the process is done and a suitable location for the large room is found, the center tile is designated as lower-case L with the surrounding eight tiles being denoted as a p, or “empty space”.

One of the many additions that will need to be made relates to how items are spawned. Currently, rooms come with built-in spawn points for items, which immediately spawn a random item from a list. While not time-consuming and quick to do, this could potentially leave oddities that can lead to frustration, such as grenades being spawned for the player to use with no grenade launcher being present in the level. A future modification can be easily applied such that the Level Generator has direct access to all of the item spawn points, and thus can dynamically create a list of items as each item is spawned to ensure that all of the resources being generated can be used. This new system would enable players to get more excited about finding, for example, grenades, because then it means that somewhere, one or possibly several grenade launchers exist within the level. While this would take a fraction of a second longer to perform, a fraction of a second is a reasonable exchange to ensure that all the resources can be used within a level.



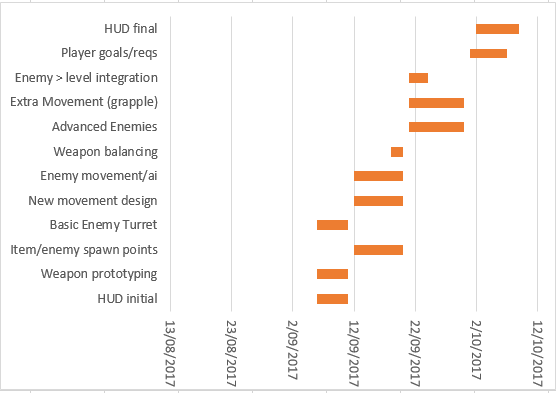


**Activities Undertaken**

|  |  |  |  |
| --- | --- | --- | --- |
| Activity | William’s Contribution | John’s  Contribution | Raymond’s  Contribution |
| Coding | 5 | 5 | 5 |
| Report Writing | 5 | 5 | 5 |
| Presentation | 5 | 5 | 5 |

We need to fill this out. To be honest, with the mess that has been our trouble getting communications going (getting Riot set up and paying attention to it), getting access to the code (only getting the project online this thursday), we’ve had trouble contributing fairly to the project. I’d suggest we drop 5s for all boxes for Assessment 4, and get our plans together for managing a better workload for the final submission (time and personal work). I’ll delete this paragraph before we submit. -John

**Gantt Charts Plan**

****

****