COMPUTING & GAMES DEVELOPMENT WITH PLYMOUTH UNIVERSITY

School of Computing, Electronics, and Mathematics

PRCO304
Final Stage Computing Project
2016/2017

BSc (Hons) Computing & Games Development

Sam Perriton-Branch

12345678

Dungeon VR

Acknowledgements

Firstly, I'd like to thank Dr. Dan Livingstone for providing excellent support and feedback over the course of the project and the rest of the ISS team for all of their support and guidance during my time at university.

Secondly, I'd like to thank my family for their unwavering support throughout my time at university.

Finally, I'd like to thank all of my classmates and the people I have met during my time at university and offered me help and kindness.

Abstract

This report covers the development of a local multiplayer procedurally generated roguelike dungeon crawler in virtual reality. This project was undertaken as an opportunity to improve skills in programming for the input of multiple users on one system and programming and design for VR games.

The report starts by going over the background for the project, including a description of the project and the setting; It also covers the aims and goals of the project. It then moves onto comparing the key elements of the project to similar already existing products to provide a quick overview of possible mechanics that could be used in the project. This comparison is proceeded by a review of related literature covering the topics of procedural level generation and locomotion in VR experiences.

The next section of the report discusses the software used for development and the reasons it was used over other alternatives along with the methodologies used for the development and the reasons why they were used over alternatives. The legal, social, ethical and professional issues caused by the content of the project and how they are to be mitigated are then discussed, followed by the choice of project management tools and how they are used.

The penultimate section of the report covers the actual development of the project, starting by describing the actual architecture of how some the more important parts of the project are set up. Then the report covers how the main features of the project itself were implemented and how testing was carried out during the project.

The final section of the report reviews the project's goals and if they were completed along with a post-mortem of the project discussing how the project went as a whole. The report ends with the author's final conclusions on how the project went.

Table of Contents

List of Tables and Figures

Introduction

This project was undertaken as a method of improving skills in programming for multiple local players, using Xbox 360 controllers or similar, as well as improving other programming skills for games based in virtual reality. The project also allows for a game to be created that can be improved upon after the author has finished university and eventually sold by the company the author has started and as such, this project does not have a client.

For the project to meet the requirements of the minimum viable product, it needed to have one pre-made dungeon that's design would showcase the full potential of procedurally generated dungeons. One player should be able to use a virtual reality headset and motion controllers to explore the dungeon in the first person, with at least one used weapon, to find the treasure and win the game. At least one other player should also be able to use a gamepad to interact with the dungeon by being able to take control of two different types of traps and be able to spawn in and control two different types of enemies.

The rest of the report will cover the background of the project, software used, the method of production and the actual architecture of the project. It will also detail the projects management tools and legal and ethical issues raised by the project.

Background, objectives, and deliverables

The project is a local multiplayer roguelike dungeon crawler in virtual reality. Local multiplayer means there will be one player in a virtual reality headset trying to achieve one goal while other players use standard game controllers, such as Xbox 360 controllers, to control characters in an attempt to stop the player in virtual reality from achieving their goal. The roguelike genre is one categorized by having a procedurally generated dungeon that the player has to progress through fighting enemies to achieve a goal. They are also often permadeath which means once the player has died they are dead and to start a whole new dungeon if they wish to play again.

The game will be set in a medieval dungeon and will have two main goals, one for the player in VR and one for the players not in VR. The goal for the player in the VR headset will be to progress through the dungeon and reach the treasure without dying. The goal for the players not in VR is to kill the player in VR and stop them from reaching the treasure. The VR player will be able to pick up a range of weapons that they can use to kill the enemies attacking them. The players not in VR will be able to spawn in one of four enemies that they can control and use to attack the player in VR. The players not in VR will also be able to take control of traps and the dungeon and can activate them in order to do a lot of damage to the player in VR.

The following tables show a brief competitor analysis. Due to the project having quite a unique USP there are not many products that are similar to the project in multiple ways, as such the tables below compare combat in VR games and locomotion in VR games.

	Trickster VR	Compound	GORN	Freedom Locomotion VR
Rating	Very Positive	5 stars	5 stars	5 stars
Type/s	Teleport	Teleport, Grip to move in direction of controller	Moving around play area only	Head and controllers tracked
Advantages	Easy to implement, standard type of movement in VR	Easy to implement, relatively intuitive	Basically no implementation need, quite immersive	Very immersive, not very nauseating
Disadvantages	Not very immersive	Can be quite nauseating	Very limiting	Very tiring, can play a game for very long, hard to implement

Table 1: Comparison of locomotion in VR games.

	Trickster VR	Compound	GORN	Raiders of Erda
Rating	Very Positive	5 stars	5 stars	N/A
Melee	Sword in holster, trigger to hold, can be thrown, reappears in holster after thrown	N/A	Melee weapon spawned near player, trigger on weapon to pick up, some need 2 hands to be useable	Trigger to pick up found melee weapons
Ranged	One hand holds bow, trigger to hold arrow in the other hand, place in centre and pull back, release trigger to shoot	Hand always holding gun, up on the touchpad to swap between pistol and SMG, trigger to shoot	One hand holds bow, trigger to pick up arrow with the other hand, place in centre and pull back, release trigger to shoot	One hand holds bow, trigger on bowstring to hold arrow and pull back, release trigger to shoot
Advantages - Melee	Easy to implement	N/A	Feels very natural	Very realistic as takes into account velocity
Advantages - Ranged	Very intuitive	Easy to implement, makes controls relatively intuitive	Very intuitive	Very easy to use, relatively easy to implement
Disadvantages - Melee	Doesn’t have much oomph, makes killing enemies very easy by spamming weapons up/down by small amount	N/A	Harder to implement, weapons don’t feel like they have much impact	Very complex to implement
Disadvantages - Ranged	Can be fiddly to use	When enemies are close, within ~1m, fights become impossible	Hard to implement, makes using harder than it needs to be	Can ruin immersion

Table 2: Comparison of combat in VR games.

The project deliverables are as follows:
A complete working Unity application, with the project having no bugs that stop the application from running and no bugs that would affect the user experience drastically. The project should also be optimised to the best of the author's ability to ensure that the VR player has and enjoyable experience that causes no ill effects from a result of lag, screen tearing or similar.

The game itself should meet at least the requirements of the minimum viable product but ideally be closer to the most awesome product. This means the game should have at least one pre-made dungeon that demonstrates the capability of the procedural dungeon generator with the VR player being able to explore it using at least one weapon to find the treasure and win the game. There should also be the ability for at least one non-VR player to use a controller to view the dungeon from the top down to control at least two different types of enemy, that can only be spawned in a sensible area, and at least two different types of trap. For the game to meet the requirements of the most awesome product it should meet the requirements of the minimum viable product but also have two premade dungeons, and have working procedural generation that creates fair (they are complete-able) and fun dungeons. The VR player should be able to use at least four weapons that have reasons for using one over another depending on how the user wants to play. There should also be the ability for two non-VR players to be able to view the dungeon from the top down, on the same screen, and be able to spawn in at least 3 different types of enemies and control 4 different types of traps.

Literature review

Due to the diversity of the areas covered by the project this review of related literature has been split into two main topic sections. The first being procedural generation and the second being movement in VR.

There several different methods used to procedurally generate levels for games. Cellular Automata is one of these methods and it works by first filling a 2-dimensional array with a randomly selected state, such as Wall or Floor. Each cell is then iterated over and a set of rules are applied to determine each cell's new state for example, a rule could be if the current cell has 4 or more walls in its eight surrounding cells it becomes a wall. This process of applying rules to each cell is then repeated for a given number of generations to produce cave like systems.

In [1] a cellular automata algorithm is used to produced cave like systems. The state of a given cell can be either Floor, Wall, or Rock and the rules that are applied to each cell over the generations state that: 1) a cell is rock if the neighbourhood value is equal to or greater than a given threshold value and is currently floor; 2) a rock cell that has a neighbouring floor cell is a wall cell. There are several merits to their implementation, such as it is very efficient and can be used to generate more of the level as the level is being played meaning that the levels can be infinitely big. However, there are some issues, one of the main problems being that this solution to procedural generation of content only really works well for 2-dimensional cave like systems. There is also issues with room connectivity as it is not guaranteed that all rooms generated in the cave system are reachable.

Genetic algorithms can also be used generate dungeons and select the most appropriate dungeons for use. In a general sense, genetic algorithms work by randomly creating a series of individuals that have traits represented by one or more parameters. How good each of the individuals is then calculated by a given fitness functions and the best, e.g. highest fitness, are then selected to potentially swap parts of their traits with other good individuals. There is then a small chance that random parts of an individual's traits are changed slightly. This process is then repeated for several generations.

In [2] a genetic algorithm is used to generate dungeons, with a tree structure representing the levels. The nodes in the tree represent a levels rooms and the edges represent connections between these rooms.

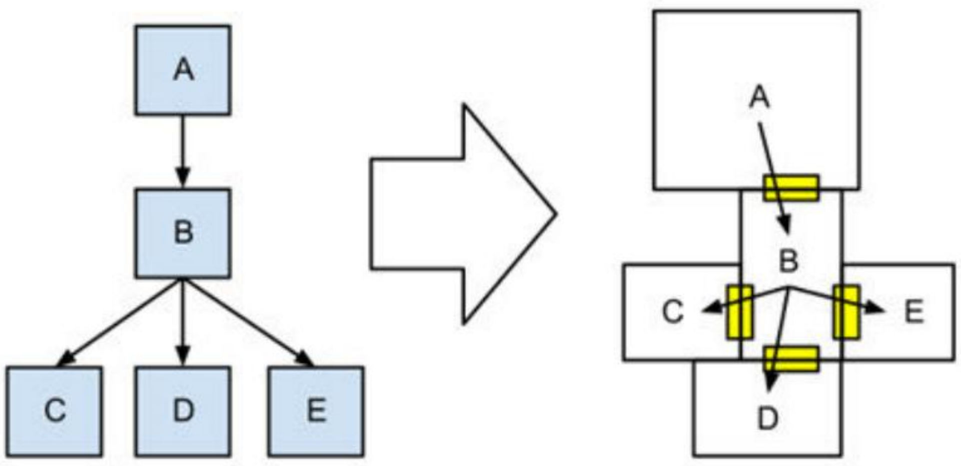


Figure 1: Translation from tree structure to map [2].

It’s worth noting that the fitness function used favours maps that are made up of tightly compact rooms with three special rooms near the edges of the level. In order to achieve levels that favour tightly compact rooms, the fitness function increases the fitness of a given level more if the hallways connect more than two rooms together. To favour three special rooms on the edges of the dungeon the fitness function increases the fitness of a level by a large amount for each special room placed that is further than a given minimum distance from the origin and drastically decreases to fitness of a level if there is more than the given maximum special rooms, ensuring levels with more than the maximum are not used. One of the main problems with this form of procedural generation is that it can lead to many of the rooms being redundant in terms of actually game play. It is also suited more to larger sprawling dungeons than small compact and concise dungeons.

There are several different types of movement in VR some of which have been briefly discussed in the background, objectives and deliverables section. As such the literature reviewed will cover other more interesting approaches.

The first of these is movement control via hand gesture input from a leap motion. This works by moving the player in a direction depending on what one or both of their hands are doing. In [3] hand gestures are used for movement control in an Oculus Rift VR walkthrough demo. To do forward and backward movement they detected the direction the hand's palm is facing and move the user in that direction, e.g. open palm facing forward moves the user forward. To side step to the left or right, the user moves their hand in the direction they wish to side step, e.g. moving a hand from left to right to side step right. To stop moving at all so you can look around at your surroundings the user simply makes a closed fist or removes all hands from the sensors view. To control the speed of the movement the user pushes their hand more in the direction they wish to travel, e.g. push hand forward to increase forward speed. There are advantages to this type of movement such as users find it to be quite intuitive and generally more immersive, however using hand gestures to control movement does mean input via motion controllers such as the Oculus Touch controllers is not possible which can make interacting with the virtual world quite lacking.

The second form of movement is an on-rails movement in which the user is moved along a predetermined path at a set speed and the user must interact with their surroundings before they are out of sight. This method of movement is used very commonly in mobile VR experiences as the amount of inputs are very limited. One of the main issues with on-rails movement is that the user's experience can be worsened as they may not be able to spend as much time as they want looking at parts of a VR experience they find interesting. This is caused by not being able to control their movement speed as a result of the lack of inputs. However [4] have come up with a novel solution to this issue by varying the acceleration of the user depending on where in the virtual world they are looking. To do this they check the direction of the user's head is looking in relation to the direction of travel and then increase or decrease the acceleration depending on the angle between the direction of travel and the direction of movement, for example if the user is looking between 40 and 75 degrees to the right of the direction of travel the users will decelerate at a moderate rate. One key feature of [4] implementation is that if the angle between the direction of travel and the direction the user is looking is greater than 90 degrees the direction of travel is reversed which means users can intuitively explore the virtual world. When exploring the virtual world the user does, however, have to travel along the predetermined route, by travelling forwards or backwards along it, making this method not very suitable for experiences in which content is generated at runtime as a route cannot be defined.

Software

The project was created primarily in Unity3D with C# in Visual Studio used for code. Unity3D was used over the similar engines like Unreal as the author has roughly three years experience with working in Unity3D so is very accustomed to creating projects with this engine. One key reason for the use of Unity3D over Unreal is that the projects art style is cartoon esque, as opposed to a more photorealistic style, which Unity3D is considered to be more proficient at achieving this art style in games.

The CryEninge could have been used as it also has a free tier that allows for the project to be commercialised in the future, however, it was not used as it is much more of a niche engine that is used to create very high-end photorealistic games. It also has a very steep learning curve and as the author has no experience with it, it would have been unwise to use as most of the projects limited time would have had to be spent learning how to use the engine.

As all the assets were acquired from Unity's assets store there was very little need to do a lot of 3d modelling, however, there were a few occasions where some models needed to be adjusted slightly. Blender was the program used to make such adjustments and was chosen over programs such as 3DS Max or Maya as Blender is free and models made with it can be used commercially with no need for licences meaning the project as a whole will still be sellable.

The projects user interface elements and any textures that required editing were done using Photoshop. Photoshop was used over similar tools such as Paint.net as the author is more familiar with Photoshop and is considered to be better for producing such content.

Method of Approach

The methodology that was used over the development cycle of the project was a mixture of scrums and kanban. This means there were weekly scrums, in which the author acted as project manager, lead artist, and lead programmer. The previous weeks work would first be assessed to see if it had been completed, then the coming weeks work would be looked at on the project's roadmap to evaluate if there was enough time in the coming week's work to finish necessary tasks or if the following week's work would have to be pushed back. If the planned week's work could be completed as intended the tasks would then be broken down into smaller subtasks. These would then be allocated an amount time and put on the projects kanban board.

There are other methodologies that could have been used that have their own positive and negatives. The waterfall model is one of these approaches to development and its main advantages are that it's easy to understand, use and manage as every stage should have specific deliverables and a review process. However, there are some disadvantages such as once the project is in the testing stage it can be difficult to go back and change features. This could cause a lot of issues with a VR project as a good portion of the learning has to happen as the project progresses due to the lack of information on how to implement a variety of VR features.

Test-driven development is another methodology that could have been used. This method works by first writing a series of tests for the function being implemented. Next, the function itself is implemented and the previously written tests are performed on the function. If the function passes all the tests the next function is moved onto, however, if it fails any of the tests the function is not complete and must be reworked and improved until it passes all of its tests. While this method produces very robust functional code it adds a fair amount of time to the project not spent on actual development of features and as this project has a relatively short time period in which the whole project needs to be completed it was felt that this method was not the most appropriate.

Legal, social, ethical and professional issues

The majority of the assets, e.g. scripts, UI, etc, have been created for the project. As such the author will own the copyright and will have no issues using these assets. In the project there are several assets being used, all of which are art based, that come from the Unity asset store, as how the game looks is a key part of the VR experience and using professional assets is the most effective way of achieving this. I will be using [5] for the walls, floors, and the interior of the dungeon. This asset costs $15 and is being used as it is relatively cheap and fits the art direction for the project well. It is also modular so works well with procedural generation. [6] and [7] are being used for the spawn-able enemies and the control-able traps in the dungeon as they are free and both fit the art style. [6] is especially good as the creators also offer paid version of each individual monster containing larger forms. This adds the potential to add extra features such as the spawn-able enemies being able to level up. The weapons used in the project are from [8]; the project only uses some of the weapons so this asset offers great potential for expansion.

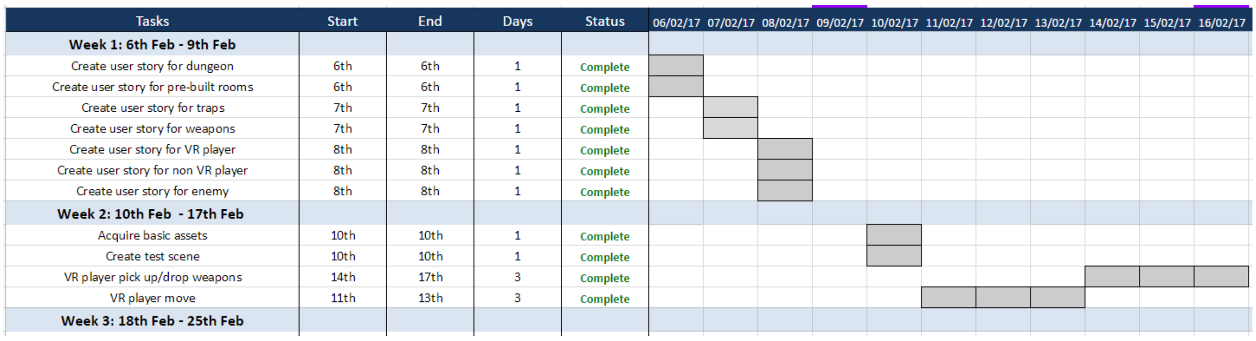
The PEGI (Pan European Game Information) rating system works by creating a list of the worst things that happen in the game and then depending on what is on this list a different rating is given. For example, if a game contains violence, bad language, and horror it would receive a rating of 12. As the project contains violence and is set in a dark dungeon, so could be considered to be fear inducing, it has been given an unofficial PEGI rating of 7. This rating would need to be made official for it to be attached to the game when it is released, however, the game could be released without a PEGI rating.

There are some health and safety issues with VR such as movement within VR causing motion sickness, people wearing a VR headset walking into objects in the real world and spending extended amounts of time using a VR headset. To avoid users walking into objects they should have a clear play area and use Steam's chaperone to alert the users in VR that they are near the edge of their play area. Motion sickness caused by movement in VR can vary drastically from user to user and how much time a user has spent using VR, as such it is impossible to completely remove any chance of causing motion sickness. One of the main causes of motion sickness in VR is the game lagging as people look around the virtual world; To combat this issue the game has been optimised to the best of the author's ability. As the game is local multiplayer it is intended that a dungeon should not take much longer than 25 - 30 minutes and then the player in VR would swap with one of the other players. This encourages users to take regular breaks from using the VR headset alleviating issues caused by extended use of a VR headset.

Project Management

To manage the project various tools were used in conjunction to one another to assure all the various aspects of the project that needed to be completed could and were completed in a reasonable time frame. There are two main tools being used, a project roadmap in the form of a Google spreadsheet and a Kanban board.

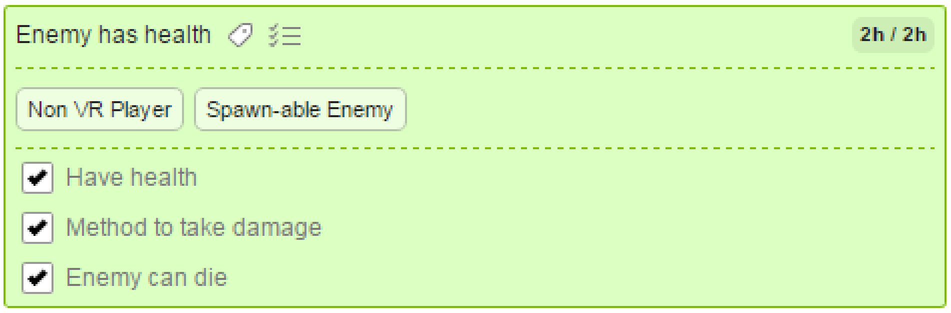
The project roadmap was used as more of an overview look of the project outlining what would need to be completed each week along with how many days and which days of that week are allocated to each task. There is also a box that contains a task's current state; A task's state can either be Complete, Overdue, In progress or Not started. If a task is to implement a feature the tasks are only marked as complete once the author has tested it himself and is working as it should be.



Tasks	Start	End	Days	Status	06/02/17	07/02/17	08/02/17	09/02/17	10/02/17	11/02/17	12/02/17	13/02/17	14/02/17	15/02/17	16/02/17
Week 1: 6th Feb - 9th Feb															
Create user story for dungeon	6th	6th	1	Complete	▨										
Create user story for pre-built rooms	6th	6th	1	Complete	▨										
Create user story for traps	7th	7th	1	Complete		▨									
Create user story for weapons	7th	7th	1	Complete		▨									
Create user story for VR player	8th	8th	1	Complete			▨								
Create user story for non VR player	8th	8th	1	Complete			▨								
Create user story for enemy	8th	8th	1	Complete			▨								
Week 2: 10th Feb - 17th Feb															
Acquire basic assets	10th	10th	1	Complete					▨						
Create test scene	10th	10th	1	Complete					▨						
VR player pick up/drop weapons	14th	17th	3	Complete									▨	▨	▨
VR player move	11th	13th	3	Complete						▨	▨	▨			
Week 3: 18th Feb - 25th Feb															

Figure 2: Start of project roadmap.

At the beginning of each week, all of the tasks for that week were put onto a Kanban board and each task expanded into greater detail in the form of a checklist of things that need to be done for the task to be complete. Each task is also given a time allocation for how long the author believes it will take to complete and labelled with relevant tags so tasks can be found later easily. Once an item on the checklist is completed it is checked off and when all items on the checklist are checked off the task is marked as completed on the roadmap, if the task was the implementation of a feature it can only be marked as complete on the roadmap once it has been tested and works as expected. At this point, the time taken to perform said task is recorded and noted on the task on the Kanban board.



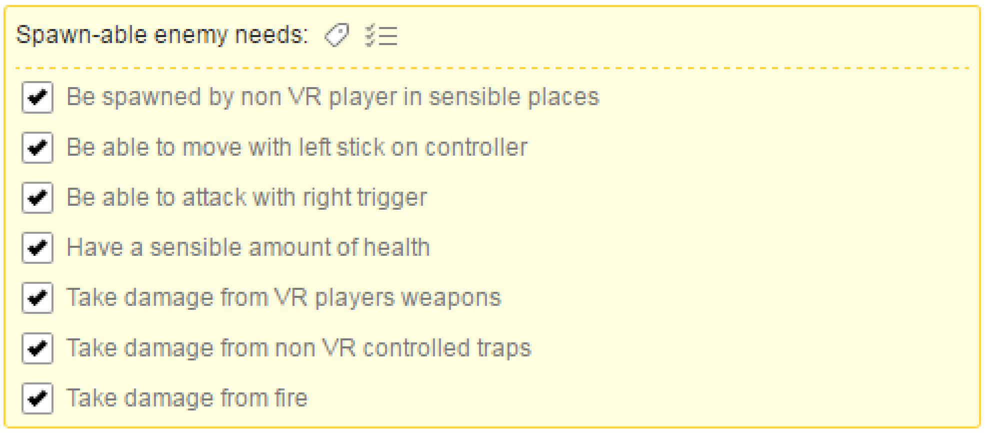
Enemy has health 2h / 2h

Non VR Player | Spawn-able Enemy

- ✓ Have health
- ✓ Method to take damage
- ✓ Enemy can die

Figure 3: Example of task on Kanban board.

The Kanban board is also used as a convenient place to store other pieces of information relevant to the project such as issues with the project, any ideas for extra features or usability that may be added in the future, and a list of possible assets that could be used in the project along with links to them. There is also a collection of user stories stored on the Kanban board for the main aspects of the project such as the dungeon as a whole, the pre-made rooms, and the weapons. The user stories are in the form checklists that can be checked off once each part of the user story is implemented.


Spawn-able enemy needs:

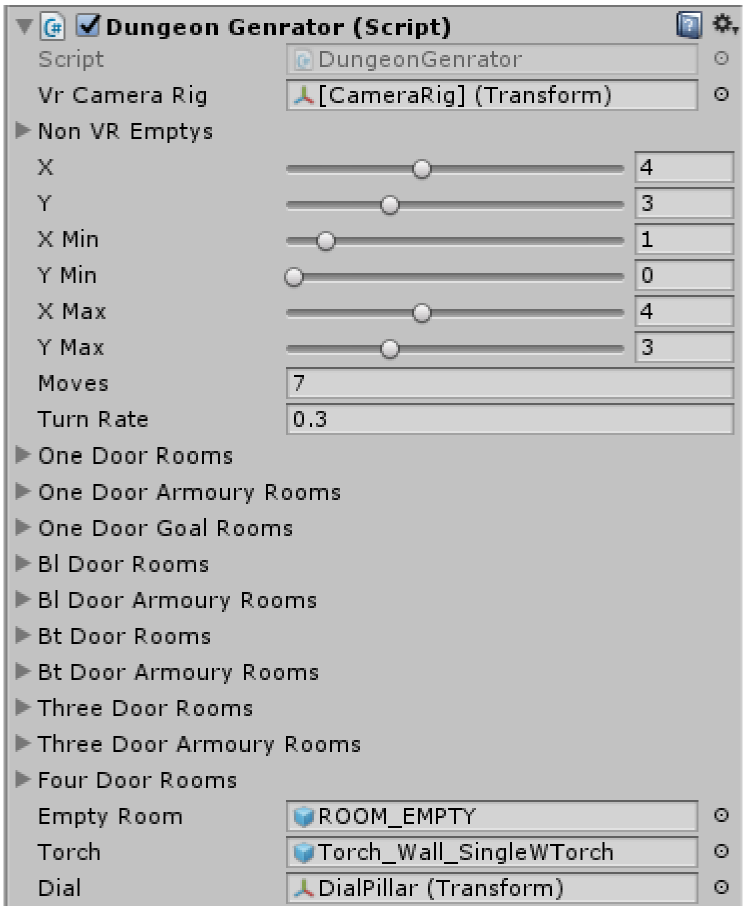
- ✓ Be spawned by non VR player in sensible places
- ✓ Be able to move with left stick on controller
- ✓ Be able to attack with right trigger
- ✓ Have a sensible amount of health
- ✓ Take damage from VR players weapons
- ✓ Take damage from non VR controlled traps
- ✓ Take damage from fire

Figure 4: Example of user story on Kanban board.

The project also uses version control in the form of Bitbucket and Sourcetree. This allows for the project to be accessed from any computer with an internet connection making working on the project on a few different machines very easy and efficient. The use of version control also means that as changes are made to the project these changes are stored, which in turn means if any changes need to be reversed for any reason, the project can quickly and effortlessly be restored to an earlier version.

Architecture

The dungeon generator has three main sections to it in the Unity editor; The first and third sections are the simplest so will be explained first. The first is the section where the VR camera rig is assigned and the empties for the non-VR players containing their cameras and controllers; This is so they can all be moved to the starting room once the dungeon has been generated. The third section is where all of the pre-made rooms are assigned, while all rooms types may not appear in every dungeon it is advisable that there is at least one version of each room type assignable. It also contains a variable for the premade torch that needs to be assigned and a variable for the dial that will be in the dungeon. The dial must start already in the scene due to a bug in Unity that stops hinge joints working correctly when instantiated. The second section is the section with all of the variables that control how the dungeon will be created, such as variables for the size of the dungeon and min/max values for which parts of the dungeon rooms can be. There is also variables for the turn rate and amount of moves the dungeon generator can make.


Dungeon Genrator (Script)

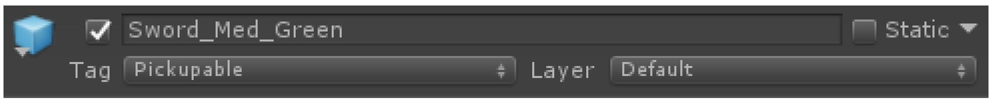
Script	DungeonGenrator
Vr Camera Rig	[CameraRig] (Transform)
▶ Non VR Emptys	
X	4
Y	3
X Min	1
Y Min	0
X Max	4
Y Max	3
Moves	7
Turn Rate	0.3
▶ One Door Rooms	
▶ One Door Armoury Rooms	
▶ One Door Goal Rooms	
▶ Bl Door Rooms	
▶ Bl Door Armoury Rooms	
▶ Bt Door Rooms	
▶ Bt Door Armoury Rooms	
▶ Three Door Rooms	
▶ Three Door Armoury Rooms	
▶ Four Door Rooms	
Empty Room	ROOM_EMPTY
Torch	Torch_Wall_SingleWTorch
Dial	DialPillar (Transform)

Figure 5: Dungeon generator setup in Unity 3D.

The pre-made rooms must all follow a certain layout for them to work with the dungeon generator and the layout is as follows. Firstly the first four children of the premade room object must be empties containing the entirety of a wall and have an underscore in their name if that wall contains a gate; The first child of this collection of wall pieces must also be the middle wall as this is used to spawn a torch on. Secondly, the room object must also have an empty call “EXTRA” that contains empties at least these four empties, “Spawnable Area”, “Dials”, “BarrelSpawnPoints”, and “Traps”; All four of these names must be correct for the dungeon generator to work correctly. The “Spawnable Area” empty should contain empties with the tag “SpawnableArea” and box colliders that designate where a non-VR player can spawn an enemy. The “Traps” empties are in setup so they have traps in the possible positions they can spawn, with all being inactive so that one or more can be turned on by the dungeon spawner as needed. The “Dials” empty is set up in a similar manner except it only contains the position and rotation of where the dial should be in a given room. The “BarrelSpawnPoints” only differs slightly to the “Traps” empty in that the empties in this empty contain a variety of different collections of barrels that are inactive ready to be made active when needed.

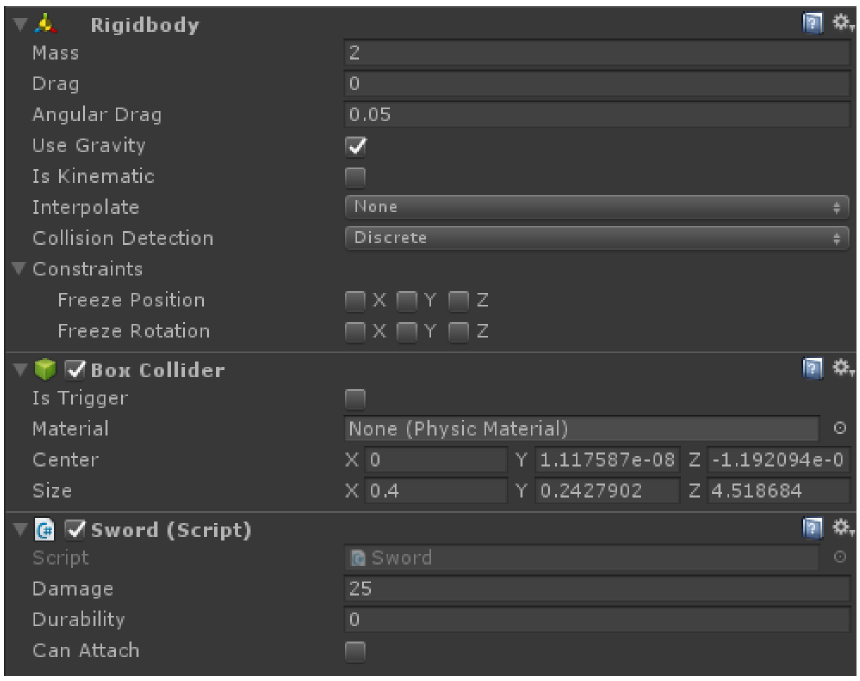
Objects that can be picked up by the VR player must all have a similar setup, this setup being the object must have the tag “Pickupable”, a rigidbody component and a collider, e.g. box collider. Some objects such as swords may need other scripts attached for extra functionality needed. This functionality may work on its own or have functionality connected to the VR player such as food giving health.

(a)



✓ Sword_Med_Green		Static	
Tag	Pickupable	Layer	Default

(b)



Rigidbody

Mass	2
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐ X ☐ Y ☐ Z
Freeze Rotation	☐ X ☐ Y ☐ Z

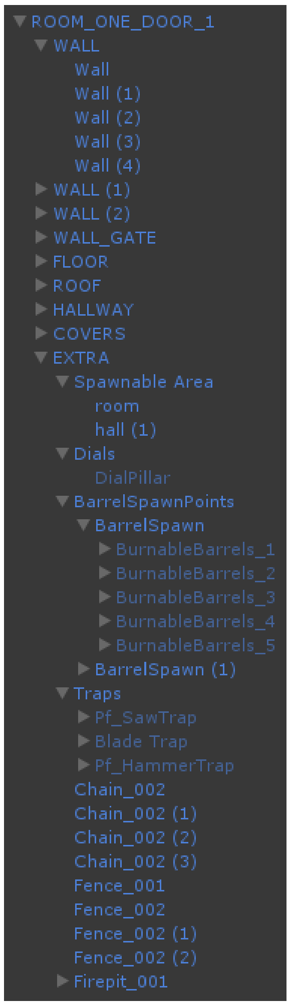
✓ Box Collider

Is Trigger	☐
Material	None (Physic Material)
Center	X 0 Y 1.117587e-08 Z -1.192094e-0
Size	X 0.4 Y 0.2427902 Z 4.518684

✓ Sword (Script)

Script	Sword
Damage	25
Durability	0
Can Attach	☐

(c)



```
▼ROOM_ONE_DOOR_1
  ▼WALL
      Wall
      Wall (1)
      Wall (2)
      Wall (3)
      Wall (4)
  ▶WALL (1)
  ▶WALL (2)
  ▶WALL_GATE
  ▶FLOOR
  ▶ROOF
  ▶HALLWAY
  ▶COVERS
  ▼EXTRA
    ▼Spawnable Area
        room
        hall (1)
    ▼Dials
        DialPillar
    ▼BarrelSpawnPoints
      ▼BarrelSpawn
        ▶BurnableBarrels_1
        ▶BurnableBarrels_2
        ▶BurnableBarrels_3
        ▶BurnableBarrels_4
        ▶BurnableBarrels_5
      ▶BarrelSpawn (1)
    ▼Traps
      ▶Pf_SawTrap
      ▶Blade Trap
      ▶Pf_HammerTrap
      Chain_002
      Chain_002 (1)
      Chain_002 (2)
      Chain_002 (3)
      Fence_001
      Fence_002
      Fence_002 (1)
      Fence_002 (2)
    ▶Firepit_001
```

Figure 6: (a) Example of pickup-able with correct tag. (b) Example of pickup-able object setup. (c) Example of premade room's hierarchy.

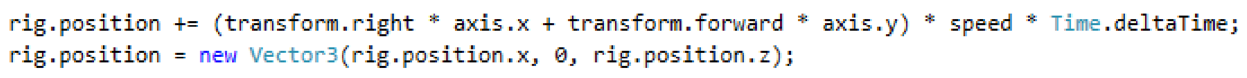
Stages

There is a total of 6 key stages in the PID, found in the appendix, however, for the course of the project, these have been treated as three main stages with a fourth final stage for the finishing touches.
For this section the stages are as follows:
Stage 1: Initial setup and basic control for VR and Non-VR
Stage 2: Pre-made room creation and VR/Non-VR interaction
Stage 3: Procedural generation
Stage 4: Final touches

Stage 1: Initial setup and basic control for VR and Non-VR

At the start of the project, the first task was to set up the project management tools, the first of which is the project's roadmap. To create this a Google sheet was used to show a list of tasks for each week of the project, the days there were to be completed on and whether or not they have been completed. With the road map setup, the project's Kanban board was created with a list of the tasks that need to be done, tasks that are in progress and tasks that have been completed, as well as a list of general notes. It was at this point that the initial repository was created on Bitbucket so that version control could be used for the project. Once all the project management tools were in place the final piece of initial setup could be completed, this being creating user stories for all the main parts of the game such as the dungeon itself, the VR player and the non-VR player. An example of a user story can be found in the project management section. The final part of the initial setup was to find all art assets needed for the project and use them to create a small test scene the features of the project can be implemented in.

With the initial setup of the project completed, the basic control inputs for the game could be implemented. This breaks down into two main parts, the basic controls for the VR player and the basic controls for the non-VR player. The basic controls for the VR player that were implemented were the ability for the VR player to move and the ability to pick up/drop objects. There are many different types of movement that could have been used but after the research mentioned earlier in the report it was decided the game would use a combination of the touchpad and the direction the controller is pointing. This effect is achieved relatively simply with the code in figure 7.

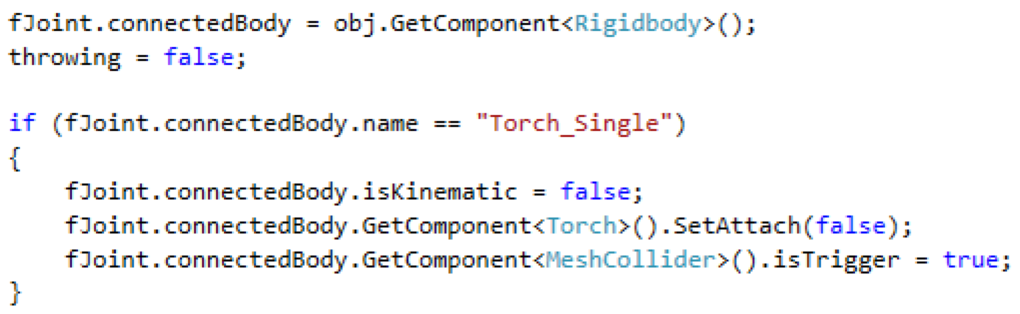


```
rig.position += (transform.right * axis.x + transform.forward * axis.y) * speed * Time.deltaTime;
rig.position = new Vector3(rig.position.x, 0, rig.position.z);
```

Figure 7: Code snippet for moving the VR player.

This code adds to the position, of the VR players play area, a vector that is calculated by adding the x value of the touched position multiplied by the right direction of the controller, to the y value of the touched position multiplied by the controllers forward direction. This value is then multiplied by a speed value and the time between frames so that movement is frame rate independent. The y value of the play area is then set to 0 to stop the VR player being able to fly upwards.

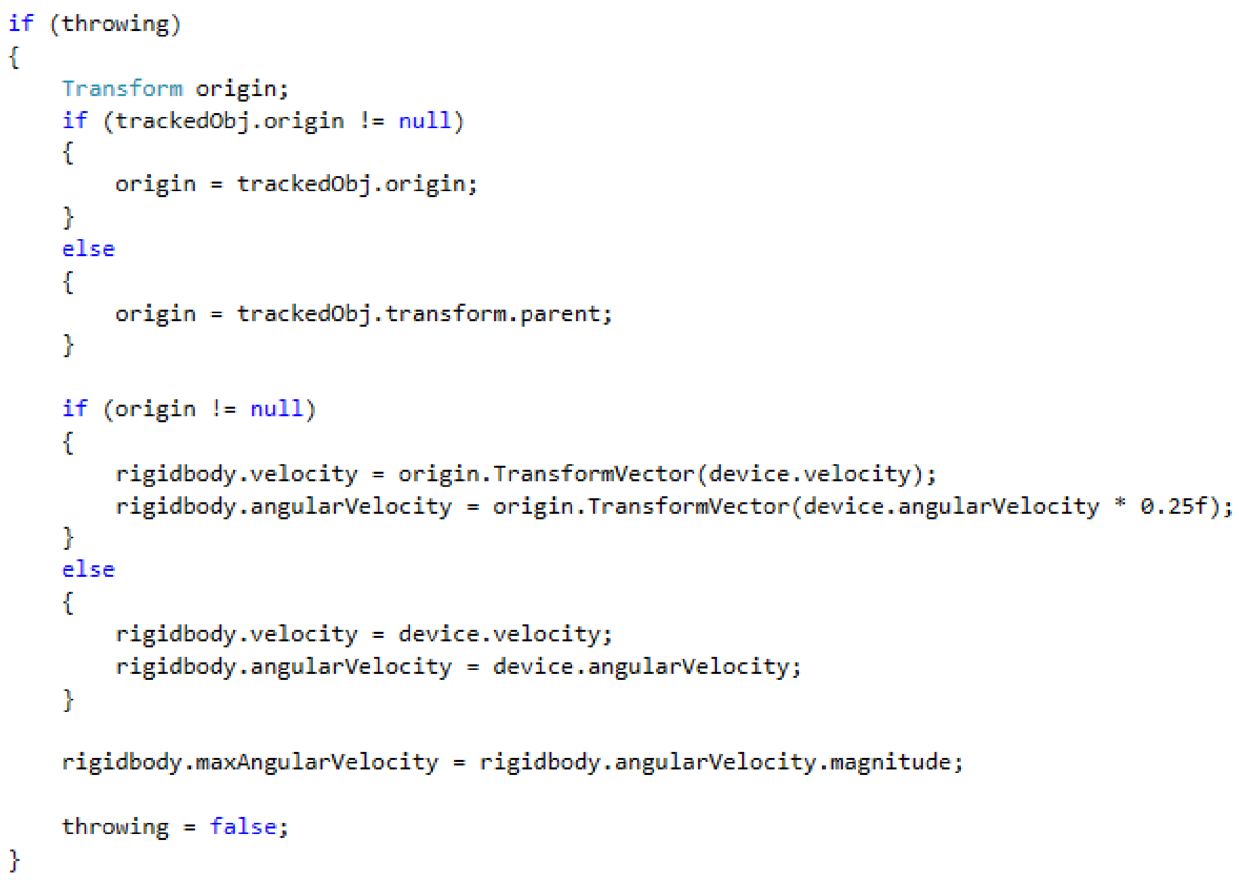
The VR players ability to pick up objects works by storing the last object, that has the “Pickupable” tag, that is within a small range of a controller and attaches the object's rigidbody to the fixed joint component on the controller when the user holds down the trigger. At this point, the name of the object can be checked and if other things are required to happen when the object is picked up it can be called. For example, if a torch is picked up there are extra things that must be altered for the torch to function properly such as the torches rigidbody being set to kinematic.



```
fJoint.connectedBody = obj.GetComponent<Rigidbody>();
throwing = false;

if (fJoint.connectedBody.name == "Torch_Single")
{
    fJoint.connectedBody.isKinematic = false;
    fJoint.connectedBody.GetComponent<Torch>().SetAttach(false);
    fJoint.connectedBody.GetComponent<MeshCollider>().isTrigger = true;
}
```

Figure 8: Code snippet for VR player picking up a torch.

When the VR player wants to drop the object they are currently holding they release the trigger on the controller holding the object. This will then cause the object to no longer be attached to the controllers fixed joint and a variable set for the last object held. At this point, a variable for if an object is being thrown is also set to true. This then causes the relevant forces to be applied to the object until the end of the frame where Unity’s inbuilt physics engine will continue to apply the correct forces to make the object fly through the air naturally.



```
if (throwing)
{
    Transform origin;
    if (trackedObj.origin != null)
    {
        origin = trackedObj.origin;
    }
    else
    {
        origin = trackedObj.transform.parent;
    }

    if (origin != null)
    {
        rigidbody.velocity = origin.TransformVector(device.velocity);
        rigidbody.angularVelocity = origin.TransformVector(device.angularVelocity * 0.25f);
    }
    else
    {
        rigidbody.velocity = device.velocity;
        rigidbody.angularVelocity = device.angularVelocity;
    }

    rigidbody.maxAngularVelocity = rigidbody.angularVelocity.magnitude;

    throwing = false;
}
```

Figure 9: Code snippet for throwing objects.

The basic controls that were implemented for the non-VR player were the ability to move and attack with a selected enemy, along with all appropriate animations, and the ability to select and use a trap. To move the selected enemy the same process is used as moving the VR players play area, except that the x and y positions of a Xbox 360 controller's left stick is used, the selected enemy is told to play it’s moving animation and the rotation of it is set to be the direction it is heading.

```
selectedEnemy.GetComponent<Animator>().SetBool("move", true);

selectedEnemy.transform.position += (transform.right * lx + transform.forward * ly) * 1.5f * Time.deltaTime;

float heading = Mathf.Atan2(lx, ly);

selectedEnemy.transform.rotation = Quaternion.Euler(0, heading * Mathf.Rad2Deg, 0);
```

Figure 10: Code snippet for non-VR enemy movement.

To attack with a selected enemy the selected enemy plays its attack animation and a bool, dictating if the selected enemies collider enters the VR player the VR player will take damage, is set to true for a very short time. This means that is the selected enemy hits the VR player while playing its attack animation the VR player will take damage.

To select a trap when the A button is pressed by a non-VR player, the game first checks if there is already a trap selected and if there is currently a trap selected, sets its reference to which player has selected it to be the current player and sets the outline of the trap to green. It then checks to see if the player has tried to select a trap or clicked off a trap to deselect it and sets the selected traps reference to who has it selected to null if the latter is true. It then assigns whatever is under the selecter be it another trap or nothing to the currently selected trap. If the player has selected a trap it then checks if the trap is usable and isn’t selected by the other player and sets the selected trap to null if either is true.

```
if (Input.GetAxis((int)player + "AButton") > 0)
{
    if (selectedTrap != null)
    {
        selectedTrap.GetComponent<Trap>().SetCurrentSelecter(selecter.GetComponent<Selecter>());

        LineRenderer[] linerenderers = selectedTrap.GetComponent<Trap>().GetLineRenderers();

        for (int i = 0; i < linerenderers.Length; i++)
        {
            linerenderers[i].startColor = Color.green;
            linerenderers[i].endColor = Color.green;
        }
    }

    if (selecter.GetComponent<Selecter>().GetSelect() == null && selectedTrap != null)
    {
        selectedTrap.GetComponent<Trap>().SetCurrentSelecter(null);
    }

    selectedTrap = selecter.GetComponent<Selecter>().GetSelect();

    if (selectedTrap != null)
    {
        Trap t = selectedTrap.GetComponent<Trap>();

        if (!t.GetCanUse())
        {
            selectedTrap = null;
        }

        if (t.GetSelecter() != null && t.GetSelecter().transform.parent.name != transform.parent.name)
        {
            selectedTrap = null;
        }
    }
}
```

Figure 11: Code snippet for non-VR player selecting a trap.

To use the currently selected trap the game checks that the player has a selected trap and then if that player has pressed the Right Trigger. It then calls the traps activate method and similarly to making an enemy attack allows the trap to hurt the VR player if they should enter the traps colliders. It also sets the players selected trap to null as the trap has to cool down before it can be used by a player again.

Stage 2: Pre-made room creation and VR/Non-VR interaction

The first half of this stage was creating all the necessary pre-made rooms needed for the procedural generation to work. As the dungeon generation is based off a square grid with rooms not being able to have access from any entrance other than left, right, up and down the rooms can be broken into five main types, rooms with one door, rooms with two doors next to each other, rooms with two doors opposite each other, rooms with three doors and rooms with four doors.

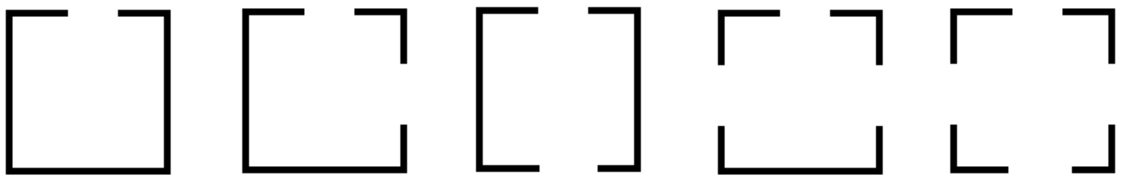


Figure 12: All five possible room variations needed for procedural generation.

Each of these five main types of rooms can potentially then be one of three subtypes, with these subtypes being a normal room, an armoury or a goal room. However, due to the way to generation works only rooms with one door can be a goal room and rooms with four doors cannot be an armoury. More on the exact structure need for a room can be found in the architecture section.

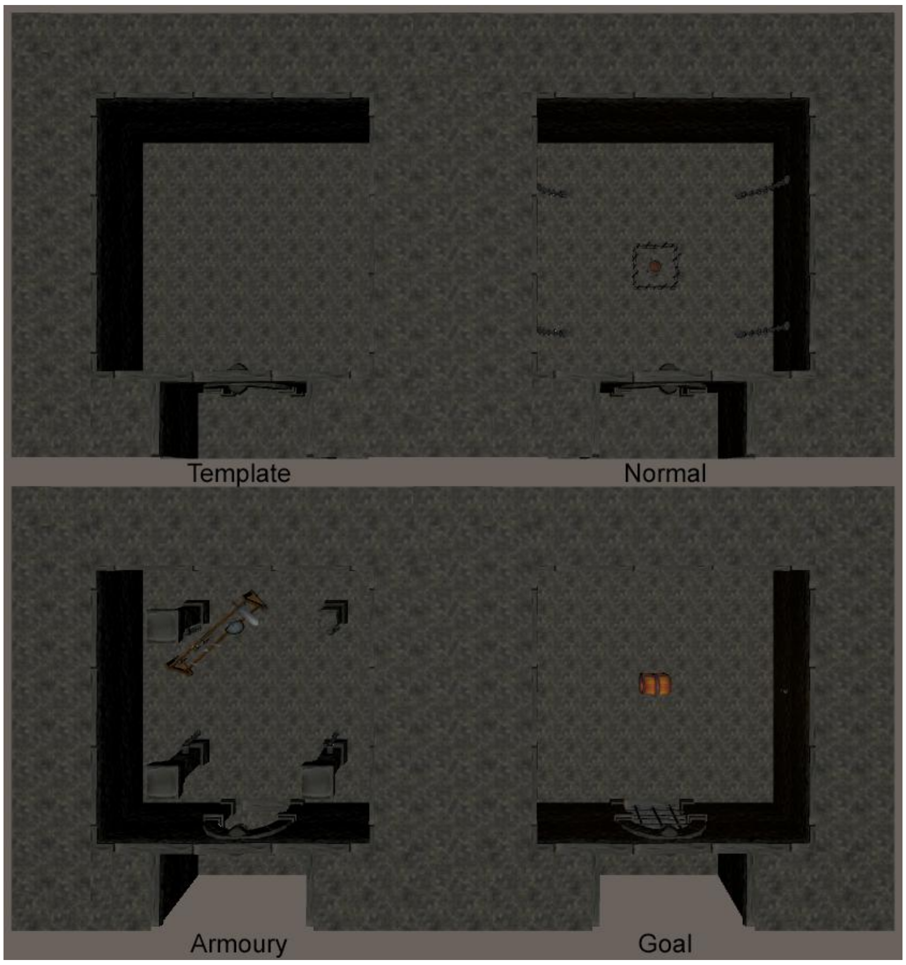


Figure 13: Template for one door room and examples of possible designs for a normal, armoury or goal room.

Once all of the pre-made rooms needed were created they could be used to create a test version of a dungeon to replace the earlier test scene and to be that could be used as an example of what the procedural generation can do.

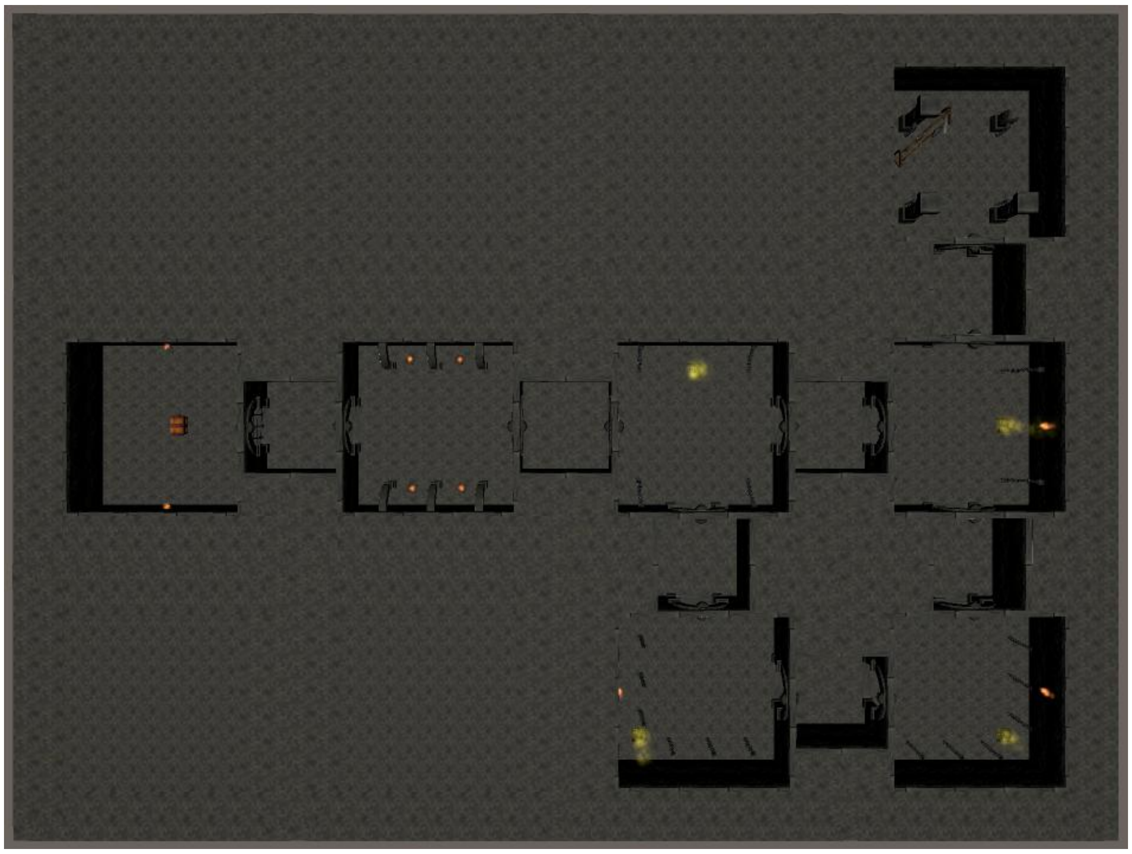
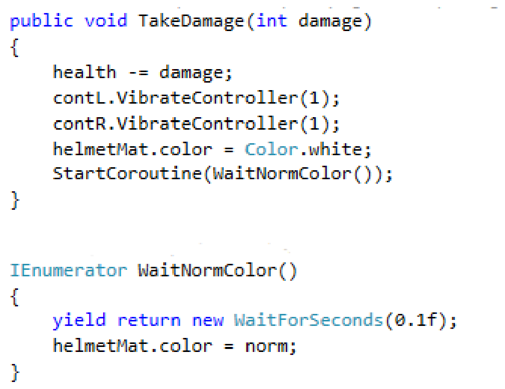


Figure 14: An example dungeon made to demonstrate the procedural generations potential.

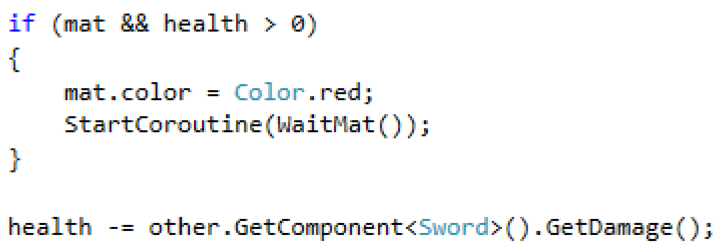
The second half of this stage was spent implementing how the main interactions between the VR player and the non-VR players as well as some other features they required. For the VR player, this was having health, the ability to take damage and increase health if food has been eaten. The VR players health was relatively simple to implement and just required adding an integer to a script, that deals with collision detection for the VR player, that stores a value for health between 0 - 100. This value has methods that increase health, that is used by objects like food, and a method that decreases health by a given amount. This method also vibrates the VR players controllers so that they know they have been hit. It also makes the VR players helmet flash white so that the non-VR players know they have hit the VR player.



```
public void TakeDamage(int damage)
{
    health -= damage;
    contL.VibrateController(1);
    contR.VibrateController(1);
    helmetMat.color = Color.white;
    StartCoroutine(WaitNormColor());
}


IEnumerator WaitNormColor()
{
    yield return new WaitForSeconds(0.1f);
    helmetMat.color = norm;
}
```

Figure 15: Code snippet for a function to damage the VR player and cause his helmet to flash.

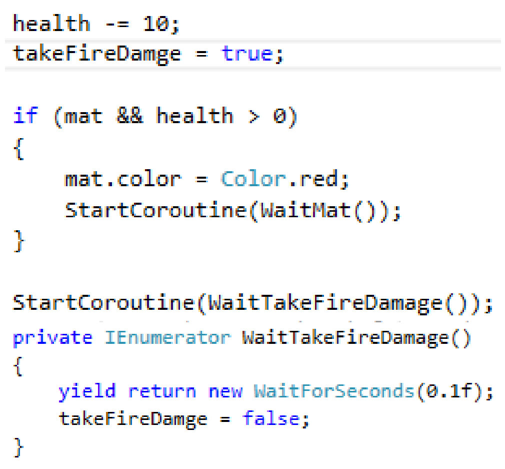
The spawn-able enemies needed a similar setup to the VR player so that they too can have health and take damage. However, there is no way for them to regain health so do not need a method like this and they can also take damage from fire so this is also accounted for. The method for enemies to take damage works by accessing the damage variable from the sword script on the sword they have been hit by and then deducts this value from the enemies health. It also makes the enemy flash red so the player can tell the hit has been successful.



```
if (mat && health > 0)
{
    mat.color = Color.red;
    StartCoroutine(WaitMat());
}

health -= other.GetComponent<Sword>().GetDamage();
```

Figure 16: Code snippet for damaging a spawn-able enemy.

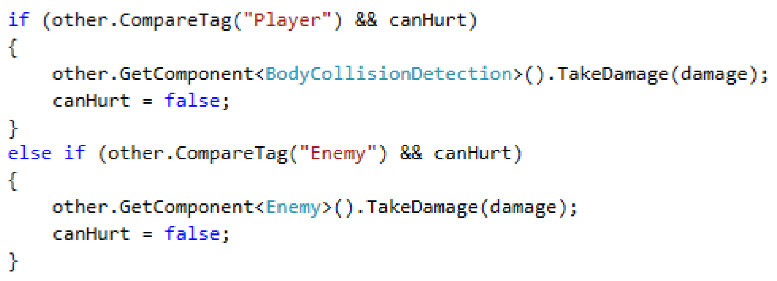
In order to take damage from fire the enemies need a method similar to the normal take damage method that occurs when a sword enters the enemies collider. This method checks if fire is currently in the enemies collider and tries to damage the enemy by a set amount if a boolean variable, called “takeFireDamage”, is false. If the fire does damage to the enemy it sets the boolean variable to be true but changes it back to false after a second has passed, meaning the enemy will take 10 damage every second it is in/on fire. The method, much like the normal take damage, also makes the enemy flash red so that players are alerted to it taking damage.



```
health -= 10;
takeFireDamge = true;

if (mat && health > 0)
{
    mat.color = Color.red;
    StartCoroutine(WaitMat());
}

StartCoroutine(WaitTakeFireDamage());
private IEnumerator WaitTakeFireDamage()
{
    yield return new WaitForSeconds(0.1f);
    takeFireDamge = false;
}
```

Figure 17: Code snippet for making a spawn-able take fire damage.

The traps that are used by the non-VR players have no need for health as they are inanimate objects, however, they do need to be able to do damage to both the spawn-able enemies and the VR player. To do this the traps have a simple script attached to the parts of the trap that can cause damage that when activated can cause damage to either one spawn-able enemy or the VR player using one of the methods mentioned above.



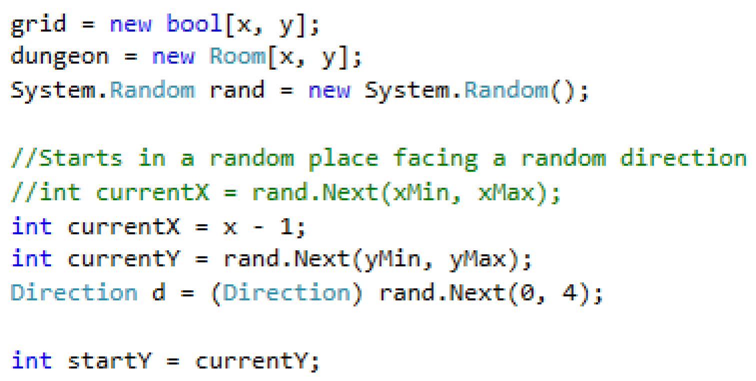
```
if (other.CompareTag("Player") && canHurt)
{
    other.GetComponent<BodyCollisionDetection>().TakeDamage(damage);
    canHurt = false;
}
else if (other.CompareTag("Enemy") && canHurt)
{
    other.GetComponent<Enemy>().TakeDamage(damage);
    canHurt = false;
}
```

Figure 18: Code snippet for making traps hurt spawn-able enemies and the VR player.

Stage 3: Procedural generation

The procedural dungeon generation stage can be split into four sections. The first being generating the dungeon in the form of a boolean grid. The second section turns the boolean grid into a grid of “Room” objects which contain the information about each room such as a number of doors and the type of room, e.g. a normal room with three doors. The third section takes the information about each room and actually creates the dungeon. The fourth and final section adds the extras to the dungeon such as torches and traps.

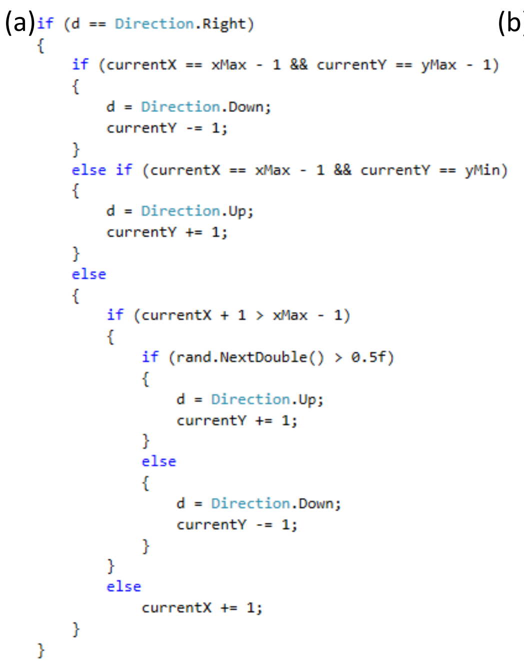
The first section works by first creating a grid of booleans and picking a random boolean to start at with an x coordinate of the furthest right column and y coordinates within the range of its minimum and maximums. It also chooses a random direction to start moving in, the direction being up, down, left or right.



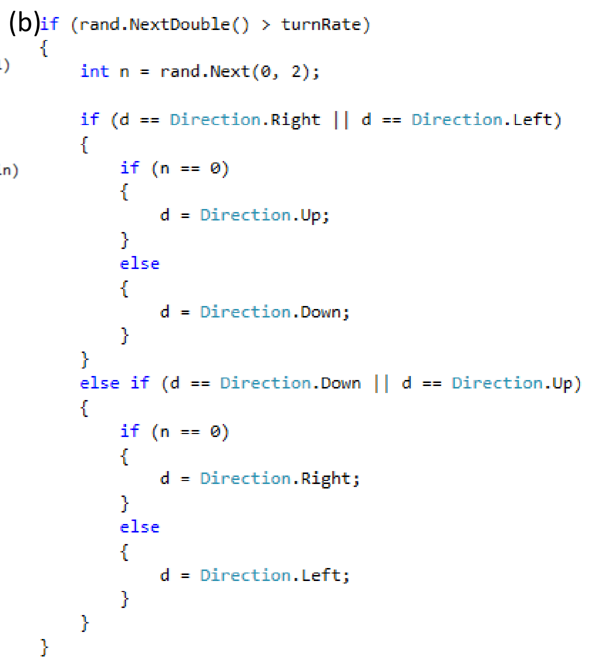
```
grid = new bool[x, y];
dungeon = new Room[x, y];
System.Random rand = new System.Random();

//Starts in a random place facing a random direction
//int currentX = rand.Next(xMin, xMax);
int currentX = x - 1;
int currentY = rand.Next(yMin, yMax);
Direction d = (Direction) rand.Next(0, 4);

int startY = currentY;
```

Figure 19: Code snippet for initialisation of variables for dungeon generation.

The dungeon generator then sets this starting boolean to be true and moves in the random direction it chose. If it is in a corner and tries to move out of the grid it will instead move in the only direction it that isn’t back on itself, for example, if it was in the top right corner and trying to moving right it would move down instead. If it is on the edge of the grid and could move two directions, not including back on itself, it will pick one of the two directions at random, for example, if it was on the far right edge of the grid and tried to move right it would instead randomly move up or down. If it can move in the direction it is going it will do so and then generate a random number and if this is bigger than the dungeon generators turn rate it will randomly turn. The dungeon generator then repeats this process of turning on the grids it’s, going the direction it’s facing and randomly turning for the amount of move specified.



(a)
```
if (d == Direction.Right)
{
    if (currentX == xMax - 1 && currentY == yMax - 1)
    {
        d = Direction.Down;
        currentY -= 1;
    }
    else if (currentX == xMax - 1 && currentY == yMin)
    {
        d = Direction.Up;
        currentY += 1;
    }
    else
    {
        if (currentX + 1 > xMax - 1)
        {
            if (rand.NextDouble() > 0.5f)
            {
                d = Direction.Up;
                currentY += 1;
            }
            else
            {
                d = Direction.Down;
                currentY -= 1;
            }
        }
        else
            currentX += 1;
    }
}
```

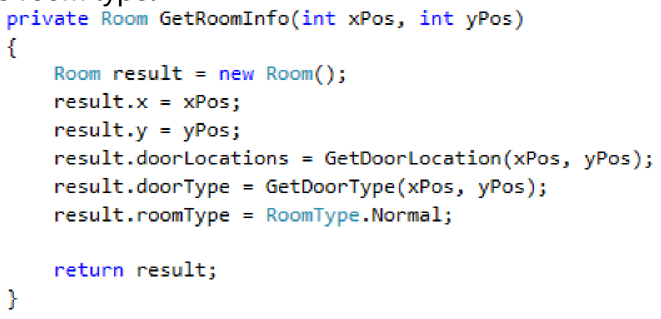


(b)
```
if (rand.NextDouble() > turnRate)
{
    int n = rand.Next(0, 2);

    if (d == Direction.Right || d == Direction.Left)
    {
        if (n == 0)
        {
            d = Direction.Up;
        }
        else
        {
            d = Direction.Down;
        }
    }
    else if (d == Direction.Down || d == Direction.Up)
    {
        if (n == 0)
        {
            d = Direction.Right;
        }
        else
        {
            d = Direction.Left;
        }
    }
}
```

Figure 20: (a) Code snippet for dungeon generator taking a step right. (b) Code snippet for dungeon generator randomly turning.

Once it has done this it adds a room to in the first column of the dungeon, that will become the goal room. This works as the x minimum is set to be the second column so the generator cannot place rooms in the first column so there will only be one room in the first column. To place a room in the first column the dungeon generator first checks if there is a room in the second column. If there is it will place the goal room in line with this room so the dungeon is definitely complete-able. However, when there are no rooms in the second column it checks the third column for rooms and will place the goal room in line with one of the rooms. It then turns on the rooms between the goal room and the room in line with it so that the dungeon is complete.

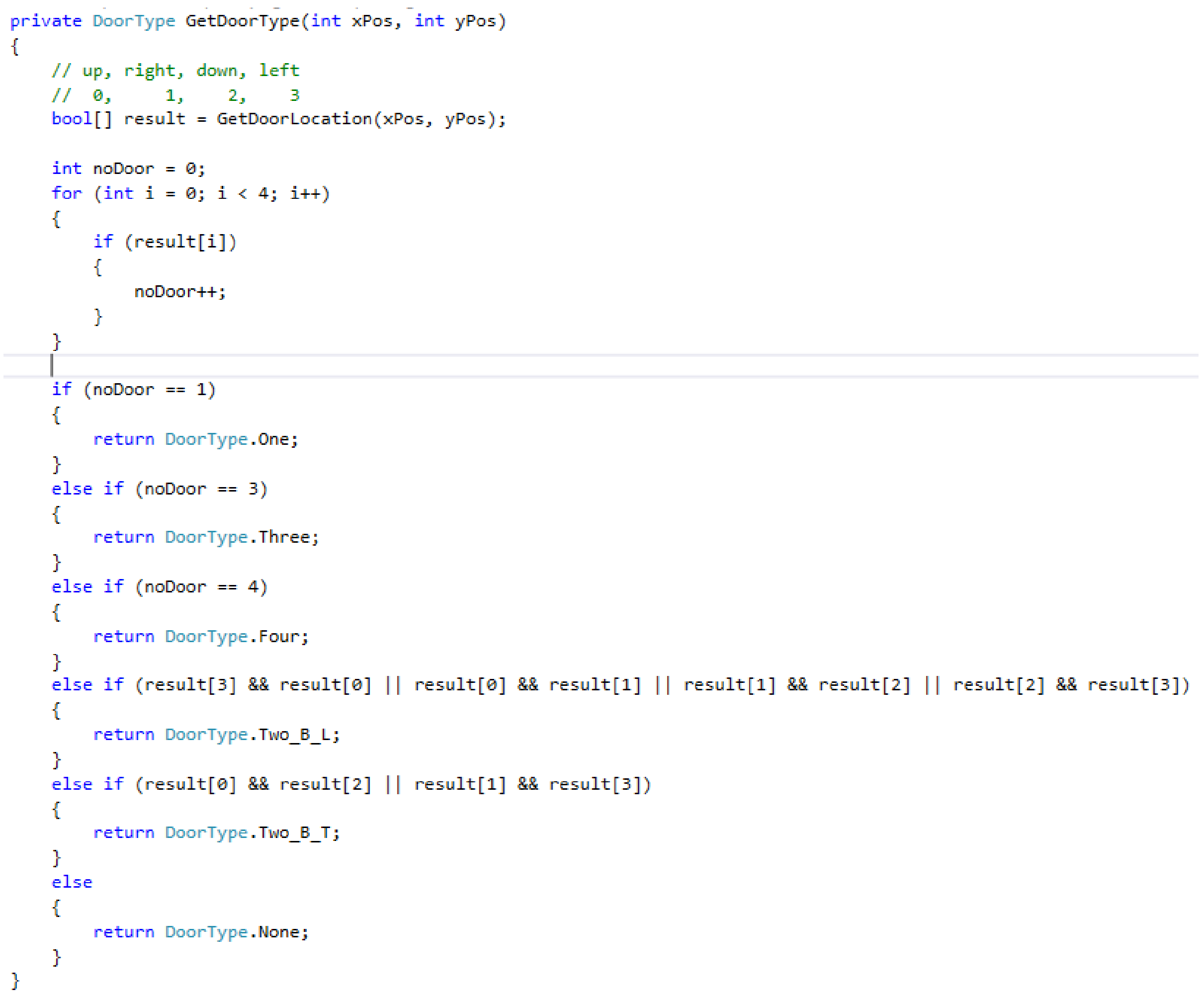
The second section starts by initiating a list that will contain all of the rooms in the dungeon and will be used later to add an armoury into the dungeon randomly. It then loops through the grid of booleans created in the last section and checks if the boolean is true or false, if it is false it creates an empty room type and assigns it to the corresponding position it the grid of room objects. If it is true it then checks to see is it is currently looking at a room in the first column as this would make it the goal room and as so it generates all the necessary room information and assigns it the goal room type; It is then assigned it to the corresponding position in the grid of room objects. If it is not a room in the first column it instead generates the necessary room information, assigns it to the corresponding position it the grid of room objects and adds it to the list of active rooms that can be turned into an armoury. To create and generate the room and necessary information the method “GetRoomInfo” is used. This method creates a new room object, assigns its coordinates of the room, the door locations, the door type, and the room type.



```
private Room GetRoomInfo(int xPos, int yPos)
{
    Room result = new Room();
    result.x = xPos;
    result.y = yPos;
    result.doorLocations = GetDoorLocation(xPos, yPos);
    result.doorType = GetDoorType(xPos, yPos);
    result.roomType = RoomType.Normal;

    return result;
}
```

Figure 21: Code snippet for generator a information about a room at a given position.

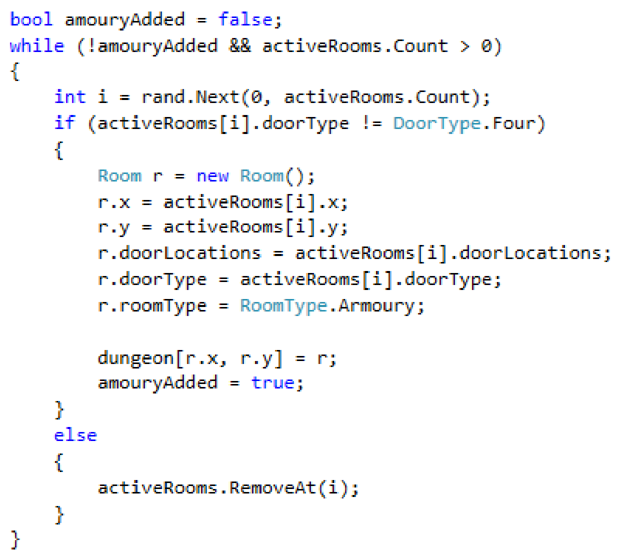
To get the locations of the doors the generator uses a method called “GetDoorLocation” simply checks the booleans above, below, to the left, and to the right of a given position. It then returns an array of booleans that indicates the positions of the doors, with the first boolean being whether there is a door to the top of this room, the second boolean being whether there is a door to the right of this room, the third boolean being whether there is a door to the bottom of this room, and the fourth boolean being whether there is a door to the left of this room. To find the type of door a room has it uses the “GetDoorLocation” method to first get the locations of the doors. It then counts how many doors the room has and uses this to assign the door type. If however to the room has two doors it can be either two doors next to each other or two doors opposite each other. As it has the locations of the doors it can simple check which doors are active and assign the appropriate door type.



```
private DoorType GetDoorType(int xPos, int yPos)
{
    // up, right, down, left
    //  0,     1,    2,    3
    bool[] result = GetDoorLocation(xPos, yPos);

    int noDoor = 0;
    for (int i = 0; i < 4; i++)
    {
        if (result[i])
        {
            noDoor++;
        }
    }

    if (noDoor == 1)
    {
        return DoorType.One;
    }
    else if (noDoor == 3)
    {
        return DoorType.Three;
    }
    else if (noDoor == 4)
    {
        return DoorType.Four;
    }
    else if (result[3] && result[0] || result[0] && result[1] || result[1] && result[2] || result[2] && result[3])
    {
        return DoorType.Two_B_L;
    }
    else if (result[0] && result[2] || result[1] && result[3])
    {
        return DoorType.Two_B_T;
    }
    else
    {
        return DoorType.None;
    }
}
```

Figure 22: Code snippet for calculating a given rooms door type.

The final step of the second section is to randomly assign one of the rooms as an armoury. As the starting room should not be an armoury as this would be unfair the starting room is removed from the list of rooms that can be an armoury. To do this the generator loops until it has added an armoury or there is no rooms left and randomly picks a room from the list and removes it if it has four doors as these can also not be armouries. If the room does not have four doors it creates a copy of the room except its type is set to armoury and replaces the room with the new armoury created.



```
bool amouryAdded = false;
while (!amouryAdded && activeRooms.Count > 0)
{
    int i = rand.Next(0, activeRooms.Count);
    if (activeRooms[i].doorType != DoorType.Four)
    {
        Room r = new Room();
        r.x = activeRooms[i].x;
        r.y = activeRooms[i].y;
        r.doorLocations = activeRooms[i].doorLocations;
        r.doorType = activeRooms[i].doorType;
        r.roomType = RoomType.Armoury;

        dungeon[r.x, r.y] = r;
        amouryAdded = true;
    }
    else
    {
        activeRooms.RemoveAt(i);
    }
}
```

Figure 23: Code snippet for adding an armoury to the dungeon.

Section three starts by instantiating a list for all of the actual game objects for the rooms in the dungeon, separate game objects for the start room and the goal room and a bool for if a room should be added to the list of rooms game objects. Once these variables have been initiated the generator can now start to spawn in the appropriate rooms and rotate them to the correct orientation. To do this it simply iterates over all of the rooms the grid of room objects and checks the door type. It checks room type and spawns in the appropriate premade room. If the room is the goal room it also assigns the room to the separate goal room game object and sets the boolean for if the room should be added to be false as things should not spawn in the goal room later, such as barrels. Now that the room has been spawned the locations of the doors are checked and the room rotated to the correct orientation.

```
if (dungeon[i, j].doorType == DoorType.One)
{
    if (dungeon[i, j].roomType == RoomType.Normal)
    {
        currentRoom = (GameObject)Instantiate(oneDoorRooms[0], gameObject.transform);
    }
    else if(dungeon[i, j].roomType == RoomType.Armoury)
    {
        currentRoom = (GameObject)Instantiate(oneDoorArmouryRooms[0], gameObject.transform);
    }
    else if (dungeon[i, j].roomType == RoomType.Goal)
    {
        currentRoom = (GameObject)Instantiate(oneDoorGoalRooms[0], gameObject.transform);
        addRoom = false;
        goalRoom = currentRoom;
    }
    else
    {
        currentRoom = (GameObject)Instantiate(emptyRoom, gameObject.transform);
    }

    if (dungeon[i, j].doorLocations[3])
    {
        currentRoom.transform.Rotate(new Vector3(0, 90, 0));
    }
    else if (dungeon[i, j].doorLocations[0])
    {
        currentRoom.transform.Rotate(new Vector3(0, 180, 0));
    }
    else if (dungeon[i, j].doorLocations[1])
    {
        currentRoom.transform.Rotate(new Vector3(0, 270, 0));
    }
}
```

Figure 24: Code snippet for creating and orienting actual rooms.

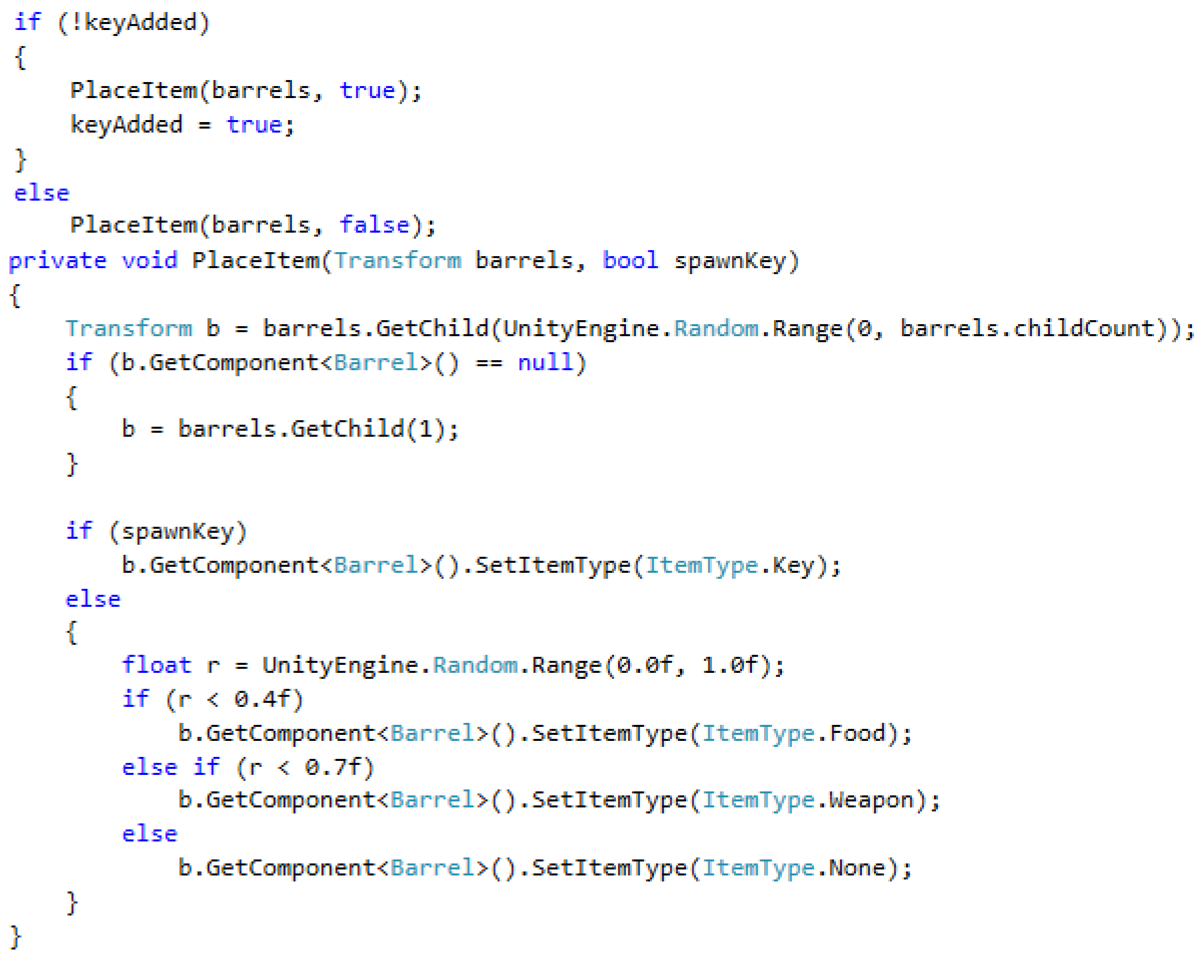
Now that the correct room has been spawned and is in the correct rotation it can be moved into the correct position and added to the list of room game objects if it is not the goal room or the starting room.

The fourth and final section of the procedural generation adds torches, barrels, traps and a dial. Adding traps is achieved by the generator by first creating a duplicate list of the room gameobjects making sure unwanted rooms are not present. It then finds the appropriate empty game object containing all possible spawns for the object, e.g. if spawning traps it finds the rooms “Traps” gameobject. Next, it adds all of the objects spawns to a list and then simply spawns the appropriate amount.

```
List<GameObject> trapRooms = new List<GameObject>();
for (int i = 0; i < dungeonRooms.Count; i++)
{
    if (dungeonRooms[i].name.Contains("DOOR"))
    {
        trapRooms.Add(dungeonRooms[i]);
    }
}

List<GameObject> trapSpawns = new List<GameObject>();
for (int i = 0; i < trapRooms.Count; i++)
{
    Transform extra = trapRooms[i].transform.FindChild("EXTRA");
    Transform trapSpawnPoints = extra.FindChild("Traps");
    for (int j = 0; j < trapSpawnPoints.childCount; j++)
    {
        GameObject trap = trapSpawnPoints.GetChild(j).gameObject;
        trapSpawns.Add(trap);
    }
}

int trapsToAdd = barrelSpawns.Count / 1;
int trapsAdded = 0;
while (trapsAdded < trapsToAdd && trapSpawns.Count > 0)
{
    int i = rand.Next(0, trapSpawns.Count);

    trapSpawns[i].SetActive(true);

    trapSpawns.RemoveAt(i);
    trapsAdded++;
}
```

Figure 25: Code snippet for adding traps to the dungeon.

Spawning barrels is done in the same way as traps except there is an extra step. This extra step is to ensure there is only one key in a barrel and that there is a random chance of either food, a weapon or nothing being in other barrels. Items are placed in barrels using a method “PlaceItem” that will place a random item in a random barrel from a set of barrels if spawn key is set to false but will only spawn a key in a random barrel if spawn key is true.



```
 if (!keyAdded)
 {
     PlaceItem(barrels, true);
     keyAdded = true;
 }
 else
     PlaceItem(barrels, false);
private void PlaceItem(Transform barrels, bool spawnKey)
{
    Transform b = barrels.GetChild(UnityEngine.Random.Range(0, barrels.childCount));
    if (b.GetComponent<Barrel>() == null)
    {
        b = barrels.GetChild(1);
    }

    if (spawnKey)
        b.GetComponent<Barrel>().SetItemType(ItemType.Key);
    else
    {
        float r = UnityEngine.Random.Range(0.0f, 1.0f);
        if (r < 0.4f)
            b.GetComponent<Barrel>().SetItemType(ItemType.Food);
        else if (r < 0.7f)
            b.GetComponent<Barrel>().SetItemType(ItemType.Weapon);
        else
            b.GetComponent<Barrel>().SetItemType(ItemType.None);
    }
}
```

Figure 26: Code snippet for placing items in barrels.

Torches are spawned on walls in a similar manner to the above. The generator first gets all of the wall in a room that do not have doorways in them. It then gets all of the centre wall sections, as this is the only place a torch should be spawned, and a torch is spawned in and positioned correctly. This is then repeated for the amount of torches need in the dungeon.

To spawn a dial the generator performs the same steps as spawn a trap except once it has the empty containing the positions and rotations a dial can be in, it moves the dial gameobject, passed to the dungeon generator as a variable, into one of the possible positions.

Stage 4: Final touches

The final touches that were added to the project primarily consisted of completing the game loop by adding a menu scene at the beginning and a win/lose scene.

The menu scene consists of two halves, one for the VR player and one for the non-VR players. The half for the VR player is quite simple and is made up of parts implemented earlier in the project and acts as a small tutorial that covers the main interactions the VR play needs to know in order to be able to progress through the dungeon and find the treasure.

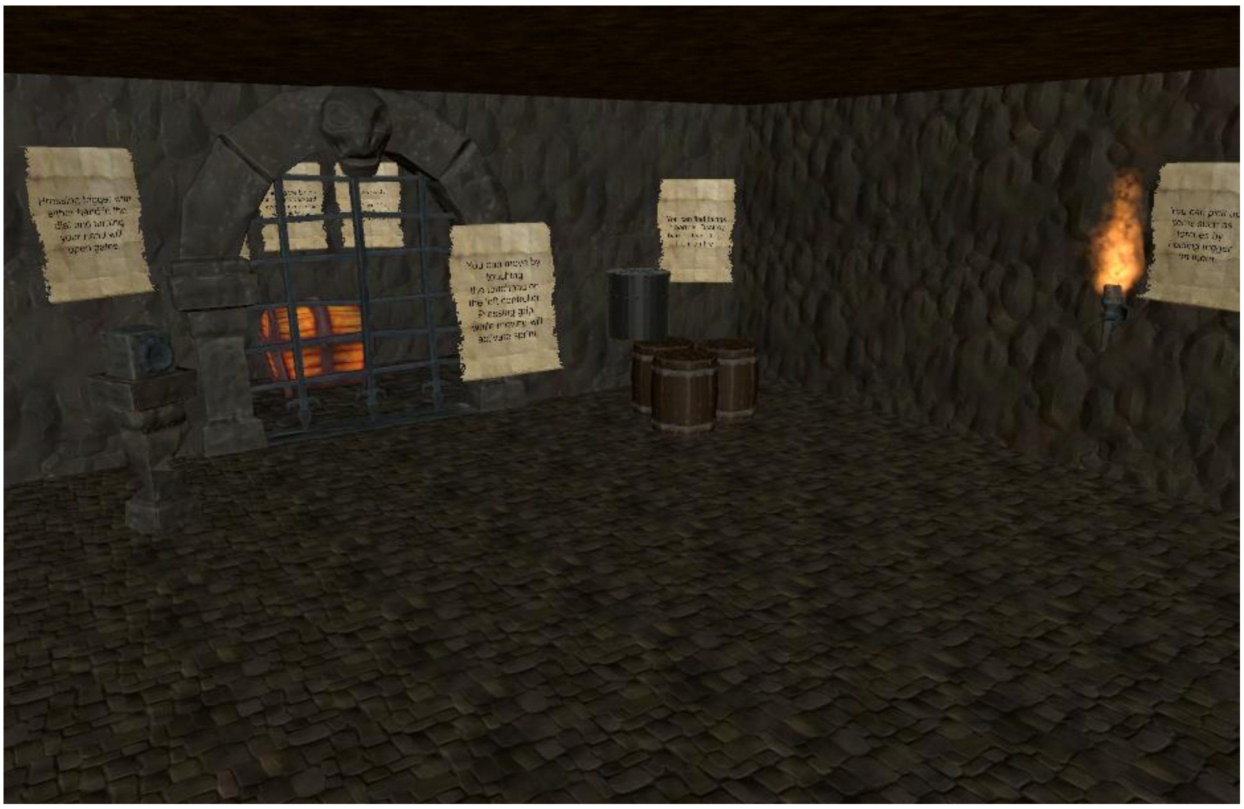
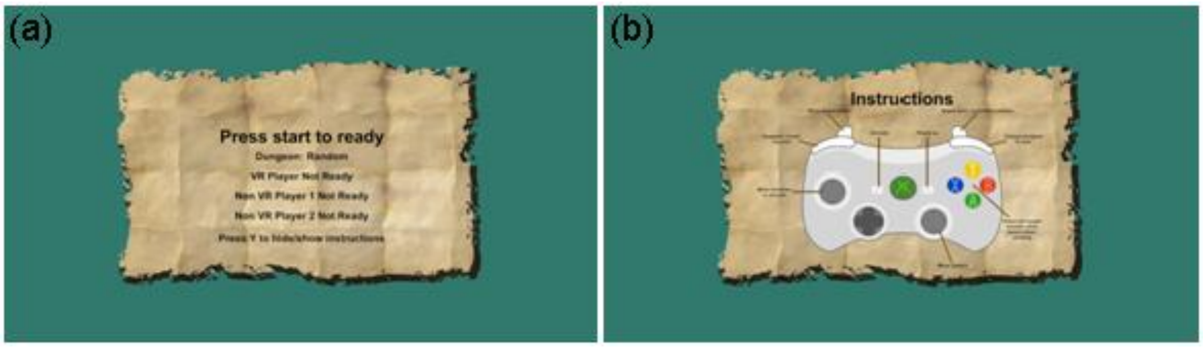

Pressing trigger with either hand in the dial and turning your hand will open gates.
You can move by touching the touchpad on the left controller. Pressing grip while moving will activate sprint.

Figure 27: Screenshot of VR players menu room.

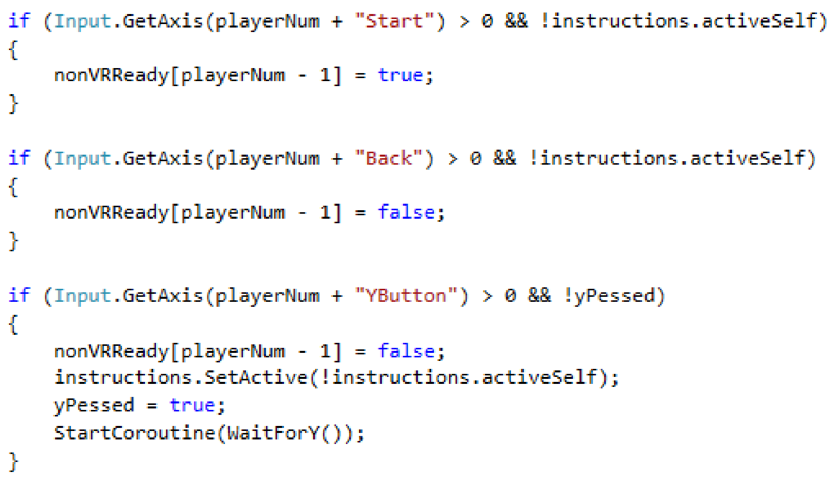
The other half of the menu scene for the non-VR players is relatively simple and just provides an interface for the two non-VR players to ready up, select the dungeon to play, and view their controls for the game.



Press start to ready
Dungeon: Random
VR Player Not Ready
Non VR Player 1 Not Ready
Non VR Player 2 Not Ready
Press Y to hide/show instructions

Figure 28: (a) Screenshot of ready up menu for non-VR players. (b) Screenshot of instructions menu for non-VR players.

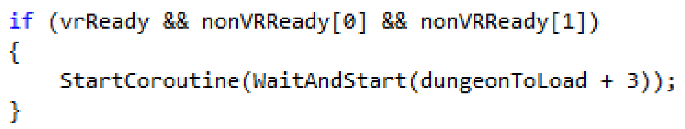
To achieve this there is a manager object that simply reads the input from the controller and if the instructions are not visible sets the players ready up bool to be true or false depending on if the player press “Start” or “Back”. It also detects when a player presses the “Y” button and shows/hides the instructions when this happens. It also only allows for the instructions to be shown/hidden every few seconds. When either player presses the right bumper the manager changes an int that represents the level to load, it will also loop this int so that it doesn’t try to load a level that doesn’t exist.



```
if (Input.GetAxis(playerNum + "Start") > 0 && !instructions.activeSelf)
{
    nonVRReady[playerNum - 1] = true;
}

if (Input.GetAxis(playerNum + "Back") > 0 && !instructions.activeSelf)
{
    nonVRReady[playerNum - 1] = false;
}

if (Input.GetAxis(playerNum + "YButton") > 0 && !yPessed)
{
    nonVRReady[playerNum - 1] = false;
    instructions.SetActive(!instructions.activeSelf);
    yPessed = true;
    StartCoroutine(WaitForY());
}
```

Figure 29: Code snippet for non-VR player input for menu scenes.

To load the dungeon and start the game the manager simply checks once when all the player's ready bools are true and loads the dungeon that was selected.



```
if (vrReady && nonVRReady[0] && nonVRReady[1])
{
    StartCoroutine(WaitAndStart(dungeonToLoad + 3));
}
```

Figure 30: Code snippet for loading the correct dungeon once all players are ready.

The win and lose scenes are both very similar to each other and like the menu also have a VR half and a non-VR half. The VR half is a simple room that either congratulates the VR player on winning or offers them commiserations for losing. The VR player has to then wait for the non-VR players to decided on reloading a dungeon or the menu scene.

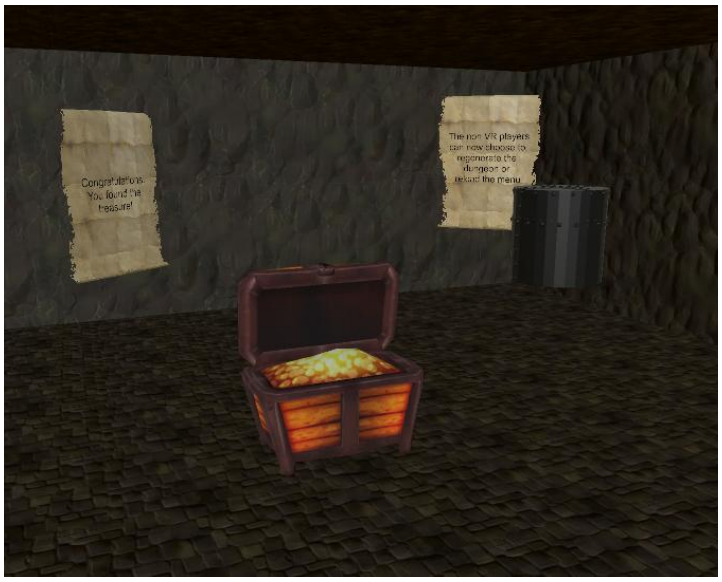

Congratulations. You found the treasure!
The non VR players can now choose to regenerate the dungeon or reload the menu

Figure 31: Screenshot of VR players win scene.

The non-VR half of the win and lose scenes do a similar job as the VR half as it offers either congratulates the non-VR players on winning or offers them commiserations for losing. It is however at this point the non-VR players can choose the reload straight back into a dungeon of their choice, selected with the right bumper, by pressing the “Start” button, or reload the menu scene so a new user can learn to play and then continue onto an actual dungeon. To do this a manager script uses similar code to the menu scene to load the selected dungeon.

Functionality Testing

Throughout the project as and when new features were implemented the author tested them by making sure they performed as they should and did not break the game in any way. VR related features were also tested to make sure they were intuitive for a user as this is a key part of what makes a VR game immersive.

Once the core gameplay for the VR player and non-VR player had been implemented user testing was performed to help ensure that the gameplay was intuitive and fun. It was also an opportunity to discover extra features users expected to be a part of the gameplay.

The format of this user testing was as follows:
1. Two people test the game, with one using VR and the other using a controller
2. Once users have played for 5 - 10 minutes each user fills out a feedback form for the input they used. (VR or Non-VR)
3. Users then play the game using the type of input they have not used. (VR now uses Non-VR and vice versa)
4. Once users have played for 5 - 10 minutes each user fills out feedback form for the input they used (Opposite form to one previously filled out)
5. Have to new users repeat this process

The results from this user testing show that users liked the look and feel of the game from both the VR and non-VR perspectives and that the game's control scheme for VR worked as expected. However, it did show that while users found the VR controls were intuitive some interactions that were expected were not present such as being able to move hanging chains with both hands. It also showed that while users liked the current selection of monsters users would like more such as a skeleton. The results of this testing can be found in the appendices.

Feature	Implemented?	Standard	Comments
Procedurally generated dungeons	Yes	MAP	Improvements could be made, such as not all rooms being connected
Local multiplayer	Yes	MAP	Works with 3 people, 1 in VR and 2 non-VR
Pre-made dungeons	Yes	MAP	Two pre-made dungeons
VR player can move around the dungeon	Yes	MVP	Works well but collision detection could be improved
Weapons for VR player	Yes	MAP	Can use can use five different weapons
Non-VR player can view dungeon from top down	Yes	MVP	Works well but can move up/down/left/right forever
Non-VR player can spawn bad guys	Yes	MAP	Can spawn in four types of bad guys
Non-VR player control traps	Yes	MVP	Can control three different traps

Table 3: List of features and if there if they were implemented.

End project report

The project is a local multiplayer procedurally generated roguelike dungeon crawler in virtual reality. This means the dungeon will be randomly generated at run time and as one player explores the dungeon in VR using an HTC Vive other players will interact with the player in VR, using Xbox 360 controllers, by spawning in and controlling monsters and by taking control of traps.

Have the project's objectives been met?

1. To have procedurally generated dungeons
   This objective was met, however they are currently improvements that could be made to the dungeon such as not always having every room connected to every room around it.

2. To have a local multiplayer VR experience
   This object was met. Two players can use controllers to interact with the dungeon the player in VR is exploring all from the same machine.

3. To have a playable demo meeting at least the MVP
   To meet the requirements of the MVP the project has at least one pre-made dungeon that shows off the potential of the procedural generation. The player in VR can explore the dungeon and find the treasure with at least one weapon. At least one other player can use a gamepad to spawn in and control at least two different types of enemy and control two different types of trap.

Project post-mortem

The project's objectives were sufficient and reasonable but really should have been broken down into smaller more concise objectives that made it clearer what was trying to be achieved over the course of the project. For example, the objective “To have procedurally generated dungeons” should have been made more specific and contained sub-objective that detailed what exactly the procedural generation should have such as “The procedural generation should add a torch to half of the rooms in the dungeon, not including the starting and goal rooms”.

The project’s development methodology was chosen as the author has participated in several successful projects that have used this methodology in the past. This methodology coupled with the appropriate tools helped to manage the project well and achieve the objectives wanted. It also helped keep the project in scope as it made it easy to see where time was needed to be spent and if there was extra time to implement other features and interactions.

If the project could be repeated more time would be put into planning out exactly what features are needed for the project in a clear and concise manner to ensure that the project was of a high standard and an achievable amount of work in the allotted time for the project. There would also be more time put into planning testing sessions and making sure there was allotted time for weekly/biweekly user testing as although the project had some user testing more would have given a lot of helpful feedback on how the project can be improved.

Conclusions

To conclude the author feels that the project went quite well and was very successful. It has helped show how much work a person can get done in a relatively short amount of time when a project has some semi-decent project management in place. It also served as a good way to help the author gauge how much time it will take to implement certain features of a project which will be an invaluable skill after university.

The project met all the goals for it to be considered the MVP and even has features bringing it closer to the MAP. The goals for these, however, should have been slight more in depth though as there is little to no consideration to sound design for the project and as such while the game is functional and enjoyable to play could have been better with improved sounds. While only a few of the art assets were created for the project, all of the art assets gathered plus the few made all fit together quite well and help give a whole project a nice look and feel. Most of the scripts created for the project are of a high standard and follow good programming practices, however, some are somewhat lacking as lots of the implementations for VR related features have yet to been perfected as it is still quite a new technology and there is not much information on the best practices for VR feature implementation.

In summary, the project was a success, achieved more goals than required to be functional and proved to be a valuable learning experience.

Word Count

10,718

References

[1] L. Johnson, G. N. Yannakakis, J. Togelius. (2010). "*Cellular automata for real-time generation of infinite cave levels". * Available: https://pdfs.semanticscholar.org/5f05/6b9bb84015cdd650f043e07f6e7d7d193ae6.pdf. Last accessed 15th May 2017.

[2] V. Valtchanov, J. A. Brown. (2012). "*Evolving dungeon crawler levels with relative placement".* Available: http://dl.acm.org/citation.cfm?id=2347587. Last accessed 15th May 2017.

[3] C. Khundam. (2015). *"First person movement control with palm normal and hand gesture interaction in virtual reality".* Available: http://ieeexplore.ieee.org/document/7219818/. Last accessed 16th May 2017.

[4] S. Mirhosseini, I. Gutenko, S. Ojal . (2017). *"Automatic speed and direction control along constrained navigation paths".* Available: http://ieeexplore.ieee.org/document/7892228/. Last accessed 16th May 2017.

[5] HONETi. (2014). *"Dungeon Modular System".* Available: https://www.assetstore.unity3d.com/en/#!/content/17933. Last accessed 9th May 2017.

[6] PI Entertainment Limited. (2016). "*Level 1 Monster Pack".* Available: https://www.assetstore.unity3d.com/en/#!/content/77703. Last accessed 9th May 2017.

[7] Sugar Asset. (2015). *"Dungeon Traps".* Available: https://www.assetstore.unity3d.com/en/#!/content/50655. Last accessed 10th May 2017.

[8] Lucie Lescuyer. (2014). *"FREE Cartoon Weapon Pack".* Available: https://www.assetstore.unity3d.com/en/#!/content/23956. Last accessed 11th May 2017.

Bibliography

Unity - Game Engine. (2017). *"Unity - Game Engine".* Available: https://unity3d.com/. Last accessed 18th May 2017.

Unreal - Game Engine. (2017). *"Unreal - Game Engine".* Available: https://www.unrealengine.com/what-is-unreal-engine-4. Last accessed 18th May 2017.

CryEngine - Game Engine. (2017). *"CryEngine - Game Engine".* Available: https://www.cryengine.com/. Last accessed 18th May 2017.

Visual Studio - An IDE. (2017). *"Visual Studio - An IDE".* Available: https://www.visualstudio.com/. Last accessed 18th May 2017.

Blender - 3D Modelling and Animation. (2017). *"Blender - 3D Modelling and Animation".* Available: https://www.visualstudio.com/. Last accessed 18th May 2017.

Kanbanflow - Project Management Tool. (2017). *"Kanbanflow - Project Management Tool".* Available: https://kanbanflow.com/. Last accessed 18th May 2017.

PEGI. (2017). *"PEGI".* Available: http://www.pegi.info/en/index/. Last accessed 18th May 2017.

Appendix

Appendix Table of Contents

Project Management Links

Bitbucket

https://bitbucket.org/sperriton-branch/dungeonvr

This repo is private but can be accessed by the ISS-Plymouth account. If you cannot access the ISS-Plymouth account or the repo but require access please email me so I can add you to the repo.

Email

Sam.PB@hotmail.co.uk

Sam.Perriton-Branch@students.plymouth.ac.uk

Kanbanflow

https://kanbanflow.com/board/ac2c36985658407b924827f0aeed65bd

Kanbanflow boards cannot be made public so if access as with the repo please email me to request access.

Project Roadmap

https://docs.google.com/spreadsheets/d/1jL7LK7mTP66GgkQ3J-pN1lUYFKWkNv5K6OvHWFwZ12Q/edit?usp=sharing

PID

Introduction

The project is call Dungeon Crawler VR, it will be a multiplayer roguelike dungeon crawler in virtual reality. This means the dungeon the player will be exploring in an attempt to find treasure will be procedurally generated at run time. It will also be permadeath so once the player is killed that's it, game over. The game will be multiplayer, one player will control the explorer adventuring through the dungeon killing monsters and looking for treasure with a HTC Vive and the other player/players will use a gamepad (Xbox 360 controller) to control the dungeon itself. This means while one player is exploring the dungeon the other will be controlling the monsters/traps in the dungeon to try and stop the player controlling the explorer from reaching his goal.

Background/Motivation

This game is inspired by Crawl[1], a 2D top down dungeon crawler in which up to 4 players take on a dungeon. The twist is that only one person plays as the hero while the others control the monsters and try to stop him from reaching his goal.

There are several reason I have been motivated to make this game. One main reason for this is that as VR devices are still relatively new there is not an abundance of games out there for them; This also means that there is even less multiplayer VR games with an even smaller number of these being local multiplayer. As I plan to start an indie game studio this gives me a good opportunity to test this test this gap in the market as a viable game genre for me to make games for.

This project also provides me with a great platform to improve and hone several game development skills such as modeling, animation and texturing, as well as developing games for VR.

Project objectives

1. To have procedurally generated dungeons
2. To have local multiplayer VR experience
3. To have a playable VR demo meeting at least the MVP

Initial scope

- Procedurally generated dungeon
  The game will need to procedurally generate a new dungeon at runtime to provide the game with lots of levels increasing it replayability. Princess.Loot.Pixel.Again[2] is a good example of how to handle procedural dungeon generation and is the approach that will be used in this project. There will be a variety of pre-designed rooms, with a few random features, being randomly selected and connected together to create random dungeon that will be fun and random but the developer will have plenty of control as to how each room is balanced.

- Single player experience
  The player will need to be able explore a dungeon, battle enemies and find treasure, in first person VR. The original Tomb Raider is a classic example of a game that does exploration very well with its use of level design and puzzles to keep the game interesting. [3]

- Multiplayer experience
  Other players will need to be able to control monsters in the dungeon with the aim to

attack the hero and stop them from finding the treasure. Crawl is an excellent example of how this can done with players able to control traps in the dungeon, such as an arrow trap with which they can fire arrow, and can only spawn a given number of monsters per room with some rooms unable to have any monsters spawned in them. [1]

- Minimum viable product
  For the project to be the MVP the game will need to have at least one pre-made dungeon that can show off the potential of procedurally generated dungeons. The player in VR should be able to explore the dungeon in first person, with at least one weapon, to find the treasure. At least one other player should be able to use a controller to view the dungeon for the top down to control at least two different types of bad guy and 2 different types of traps.

- Minimum awesome product
  For the project to be the MAP it should be the MVP but with two premade dungeons and have working procedural generations that provides fair (they are completable) and fun dungeons. The player should be able to use at least 4 weapons. At least 2 other people should be able to use a controller each to view the dungeon from the top down, on the same screen, to control at least three different types of bad guys and 4 different types of traps.

Resources and dependencies

This project will rely on resources such as Blender to create models and animations with Photoshop to create textures. The Unity asset store and TurboSquid will also be used for models and animations where needed. Sounds effects will be made either by using foley sound with royalty free sites like Soundbible providing sound effects that cannot be created. The game will also need a HTC Vive and Xbox 360 for development with both of these being supplied by the university.

Method of approach

An agile software development approach with this project will be taken, specifically there will be weekly scrums done to keep the project on track and on schedule with the projects roadmap. As the author of the project is the only contributor to this project, he will be taking on the roles of the project manager, artist and lead programmer at the weekly scrums where tasks will be put kanbanflow along with an allocated amount of time.

Initial project plan			
Stage	**Output**	**Expected Start Date**	**Expected Completion Dates**
1	Create project roadmap for project. Create user stories for dungeon, pre-built rooms, traps, weapons, VR player, non VR player and spawn-able enemies.	6th February	9th February
2	Acquire basic assets and create a test scene. Add ability for VR player to pick up/drop weapons with triggers and move with left touch pad.	10th February	17th February
3	Add ability for non VR players to be able to control enemy. Add ability for non VR players to be able to	18th February	25th February

	control traps.		
4	Create the five basic types of pre-built rooms needed. Create small example dungeon.	26th February	3rd March
5	Add ability for non VR player to spawn as enemy. Add health to both the VR player and enemies. Add the ability for the VR player to be hurt by traps and enemies. Add ability for enemies to be hurt by traps and the VR player.	4th March	10th March
6	Procedurally generate dungeon by selecting squares on grid. Then spawning and orienting one of the pre-built rooms. Pick one to spawn VR player in and one for the treasure.	11th March	24th March
7	Create more versions of each of the room types	25th March	2nd April
8	Easter vacation	3rd April	23rd April
9	Complete final report	24th April	5th May

Control plan

The following control techniques will be used:

1. Weekly scrums
2. A project roadmap
3. Highlight reports as dictated by the PRCO304 module
4. Review meetings with project supervisor as dictated by the PRCO304 module.
5. Risk management plan

Communication plan

I will have review meetings with my project supervisor inline with the Control plan with ad-hoc communication as it is needed.

Risk management plan	
Risk	**Management strategy**
Schedule overrun	The plan takes into account that there may be some schedule overrun throughout the project and will make sure there is extra time that can either be used for catch up or to add extra features.
Difficulty to implement procedural generation due to potential complexity	If there should be any difficulties implementing the procedural dungeon generation there is always the option to make a few levels by hand that show off the potential of the other gameplay features.
Technology failure	There are two main pieces of technology that could fail my home computer itself orthe HTC Vive provided by the university. If my

	home computer should fail for any reason there will be little to no affect on the project’s progress as I will be able to continue working on it using the facilities provided by the university. If the university’s HTC Vive brakes this will significant issues with the project but as a fallback if this unlikely event should happen I will have to remove the VR aspect of this game and make each player control his character with a gamepad.
Loss of work	Regular backups of the project will be made and this coupled with the use of version control such as bitbucket means that if there is any data loss it will cause minimal issues as there will be a recent backup.
Injury while in VR headset (e.g. walking into things)	There will be several safety precautions in place to avoid any accidents occurring such as removing any thing in the area being used, mating being down so anyone in the headset will be able to feel if they stray from the clear area and using chaperone to give a visual indication of the clear area. Frequent breaks should also be taken to minimise any feelings of dizziness or sickness felt from being in the headset for extended periods of time.

Quality plan	
Quality check	**Strategy**
Requirements	A Project Initiation Document, PID, will be made that will outline the project's requirements and these will be checked to ensure that they SMART and a reasonable amount of work.
Design validation via unit testing	Throughout the project there will be regular unit testing performed to ensure that all of the features implement work.
Design validation via user testing	During the project various features of the game will be tested by actual users and feedback recorded so that the current quality of the game can be keep a high standard.

Legal, social, ethical and/or professional issues

All testing done with real users will comply with Plymouth University's ethics policy. The project will also follow “Real Virtuality: A Code of Ethical Conduct. Recommendations for Good Scientific Practice and the Consumers of VR-Technology” by Michael Madary and Thomas K. Metzinger.

References

[1] http://www.powerhoof.com/crawl/
[2] http://princesslootpixelagain.wikia.com/wiki/Princess.Loot.Pixel.Again_Wikia
[3] http://tombraider.wikia.com/wiki/Tomb_Raider_(1996_Game)

Highlight Reports

Highlight Report 1

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 08/02/2017
Review of work undertaken: At the beginning of this week I started by editing my initial PID as per my supervisors request as it was not detailed enough. Once I had done this I then completed all the work that was planned for the week. The project roadmap that was completed as per the plan for this week can be found here. The users stories that were also completed for this week as per the plan can be found on my kanbanflow board here.
Plan of work for the next week: Next week I plan to acquire all the basic assets I will need and use these to put together a simple test scene in which I can implement other features. I also plan to implement the main controls for the VR player; Specifically these will be the VR players ability to move with the left controller's touch pad and pick up/drop weapons using the controller's triggers.
Date(s) of supervisory meeting(s) since last Highlight: 06/02/2017
Notes from supervisory meeting(s) held since last Highlight Update PID using comments on SPMS
Stage review:

Highlight Report 2

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 15/02/2017
Review of work undertaken: At the beginning of the week I gathered possible assets and then decided on which ones I would be using in my project, these can be found listed on my kanbanflow board. Once I had these assets I put together a small test scene that can be used to develop other features of the project. I also implemented the ability for the VR player to pick up/drop objects and the ability to move. My supervisors recommended I write a short paper comparing and contrasting types of player movement in VR, as I such I made a small mindmap and start to gather resources for this.
Plan of work for the next week: Next week I plan to implement the non VR player enemy control along with the non VR players ability to control traps. I also intend to write the bulk of the short report my supervisors suggested.
Date(s) of supervisory meeting(s) since last Highlight: 13/02/2017
Notes from supervisory meeting(s) held since last Highlight Add descriptions to tasks on knabanflow with more information on how I achieved each task and any other relevant research I had to do. Primary actions done (VR player can interact with objects), consider how the objects interact with each other. Add links to kanban to useful pages (roadmap, etc). Can use kanbanflow to help think of report topics. Write short paper comparing and contrasting control types in VR games.
Stage review: All the previous stages task were completed on time and in the allotted time. Some tasks were completed quicker than plan so I used extra time to refine the features implemented and the test scene.

Highlight Report 3

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 22/02/2017
Review of work undertaken: This week I implemented basic controller support along with all the needed animations for the four enemy types. This means they player is able to control any of the enemies with a controller, left stick to move and right trigger to attack. I also implemented the ability to take control of a trap with the A button and then activate it with right trigger. After speaking to my supervisor I decided that once a trap is used the player controlling it should lose control until trap's cooldown is over
Plan of work for the next week: Next week I plan to create a basic version of all of the necessary rooms, rough design can be found here. Once I have made all five of these room types I will be able to construct a small dungeon to use as an example of what the procedural generation will be able to do. I have also decided that once I have made an example version of a dungeon I will do some user testing to discover people's views on the current iteration of the project.
Date(s) of supervisory meeting(s) since last Highlight: 20/02/2017
Notes from supervisory meeting(s) held since last Highlight Mindmap object interactions. Sort out objects interacting with objects. Map few rooms to work out how objects could be used in future rooms. (Burn all barrels maybe need one for next room?) Map out potential combos of intractables. Different things in barrels. Deselect traps while cooling down.
Stage review: Nearly all of this stages goals have been completed on time. All of the basic goals have been completed however only a simple version of the edge glow effect I wanted to implement has been implemented; This means over the coming weeks I will need to improve this.

Highlight Report 4

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 02/03/2017
Review of work undertaken: This week I created the basic versions of the rooms I need for my procedural generation, see picture here. Once I had made these basic room types I then used them to create slightly different versions of each. These variations were an armoury and normal version except for the room with one door, which also need a goal version, and the room with four doors, which will not have an armoury version. Unfortunately due falling ill for a few days I did not have enough time to proceed with the user testing I had originally planned.
Plan of work for the next week: Next week I plan to add most of the interactions between the player using VR and the players using controllers, such as the player in VR and the players using being able to hurt one and other. I also plan to add the ability for the players not in VR to be able to spawn into the dungeon as one of the four types of enemy.
Date(s) of supervisory meeting(s) since last Highlight: 20/02/2017
Notes from supervisory meeting(s) held since last Highlight
Stage review: All of the goals for this stage have been completed. Some of these goals were completed late due to illness at the beginning of the week. I have not done the user testing I had planned to as I choose to improve the edge glow shader that I started to implement last week.

Highlight Report 5

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 08/03/2017
Review of work undertaken: This week I implemented the main interactions between the spawnable enemies, VR player, and traps. These features include things like the VR player being able to kill the spawnable enemies and vice versa and traps hurting both the VR player and spwnable enemies.
Plan of work for the next week: Next week I plan to start implementing the procedural generation that will create the dungeons. After speaking to my supervisor I now also plan to try and write a good portion of the intro for my report as well as trying to map out the possible user/object interactions for each object.
Date(s) of supervisory meeting(s) since last Highlight: 07/03/2017
Notes from supervisory meeting(s) held since last Highlight Look into sounds sounds Lookup academic references from conferences Trial an error weapon property's and write up For report show method of working, user story, give features, details lead to unit test, then task completed if it passes unit tests Map potential item interactions Say why you're using Unity over say unreal Think of possible story
Stage review: All tasks have been completed on time for this stage. There are however new tasks added to the next stage so I will have to update my road map accordingly.

Highlight Report 6

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 15/03/2017
Review of work undertaken: This week I implemented the basics of the procedural generation as well the ability for the players not in VR to spawn in enemies. I did a small amount of testing this week and gained some valuable feedback. The followed this format: One user would play using the VR headset while the other user plays using an xbox 360 controller Once one of the user has won they then complete a feedback form about how they found using their input (xbox 360 controller or HTC Vive) They then play using the other input type Once again after someone has won they fill out a feedback form on their respective input (xbox 360 controller or HTC Vive) The results from this feedback show that users liked the look and feel of the project The results also showed that while users were happy with the control schemes the movement was too slow It was also revealed that users would like more interactable objects in the game such as food I also spent time mapping out the possible interactions between objects. As well as the above I wrote a good section of my report.
Plan of work for the next week: Next week I plan to finish the implementation of procedural dungeon generation. I also plan to on the interaction between objects will continue writing my report. Hopefully I will be able to do some more user testing.
Date(s) of supervisory meeting(s) since last Highlight:
Notes from supervisory meeting(s) held since last Highlight
Stage review: End of stage next week.

Highlight Report 7

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 23/03/2017
Review of work undertaken: Improved procedural generation of dungeons. It now adds a goal room and makes sure there is always a torch in the first room; The first room is now also never an armoury. The dungeon generator also ensures there is a room in the far right column of the dungeon. This coupled with the goal room always being on the far left of the dungeon means the dungeon will always at least four rooms in the dungeon. There is also one torch spawned in a room for every two rooms in the dungeon. After last weeks feedback I also implemented the ability for items to be spawned in barrels, this will be how the VR player will find the key need to open the treasure chest to win the game.
Plan of work for the next week: Next week I plan to create more of the pre-made rooms to make the dungeon feel more unique each time it is generated. I will also have to continue improving the generation as I did not get as much done this week as planned.
Date(s) of supervisory meeting(s) since last Highlight:
Notes from supervisory meeting(s) held since last Highlight
Stage review: This stage was meant to be completed by today, however as I had to spend most of my time this week working on a STEM application I have not completed it. I will however aim to have it finished by the end of next week.

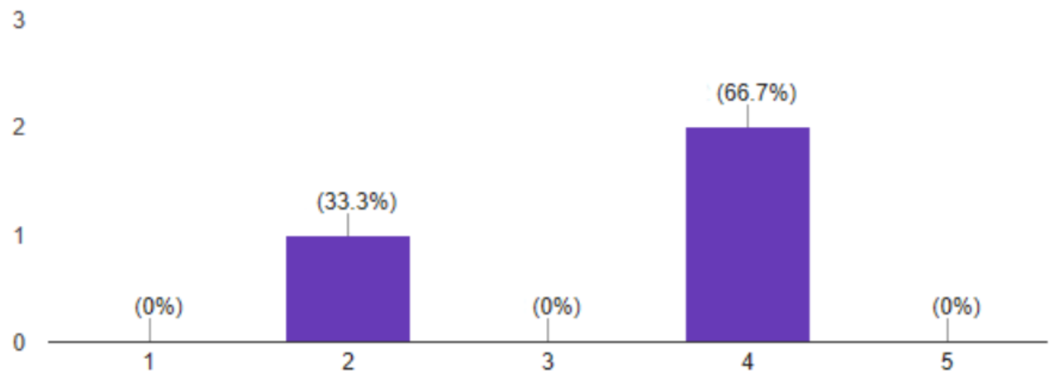
Highlight Report 8

PRCO304: Highlight Report
Name: Sam Perriton-Branch
Date: 30/03/2017
Review of work undertaken: This week I pushed back the work I originally intending on doing, in favor of implementing several things in VR such as adding UI for player health, stamina and if they have a key, the ability to sprint, the ability to eat food and some smaller bugs. I also improved the dungeon generation by adding a key that will spawn in a barrel in one of the many sets of barrels that now spawn in the dungeon. These collections of barrels that are placed into the dungeon will also randomly contain weapons or food, that can be eaten to regain health. The VR player can now use this key the open the treasure chest and win the game. The dungeon generator now also spawns in a lever in a random room that is used to open the gate into the treasure room.
Plan of work for the next week: Next week I plan to do the work I had initially planned for this week as well as continuing the report. I will also add some extra features such as weapon durability, a lobby screen for non VR players, a room for the VR player to ready up and a win/lose screen so that the non VR players can chose to restart a new game or not.
Date(s) of supervisory meeting(s) since last Highlight:
Notes from supervisory meeting(s) held since last Highlight
Stage review: Last weeks stage was not completed, however I did manage to complete last week's stage and am currently up to date with this week’s task.

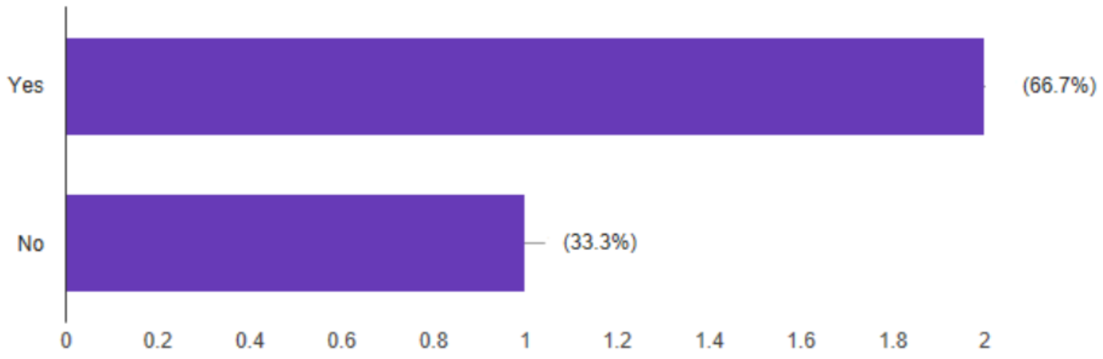
Test Results

VR Player

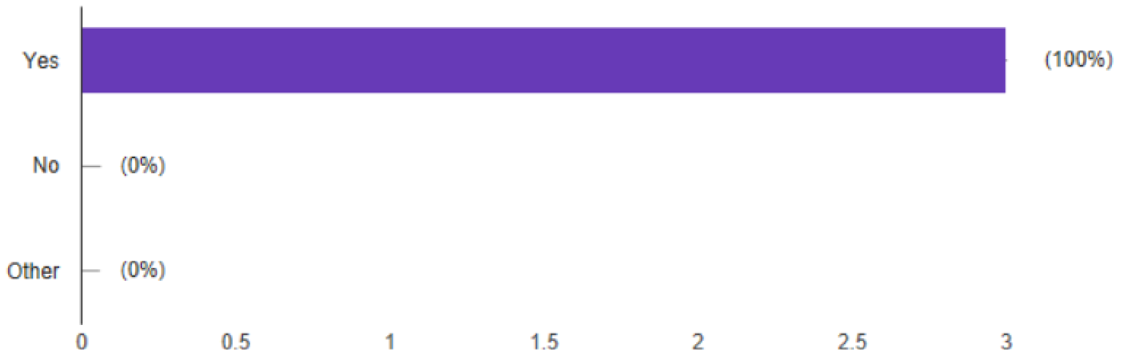
How was the movement?



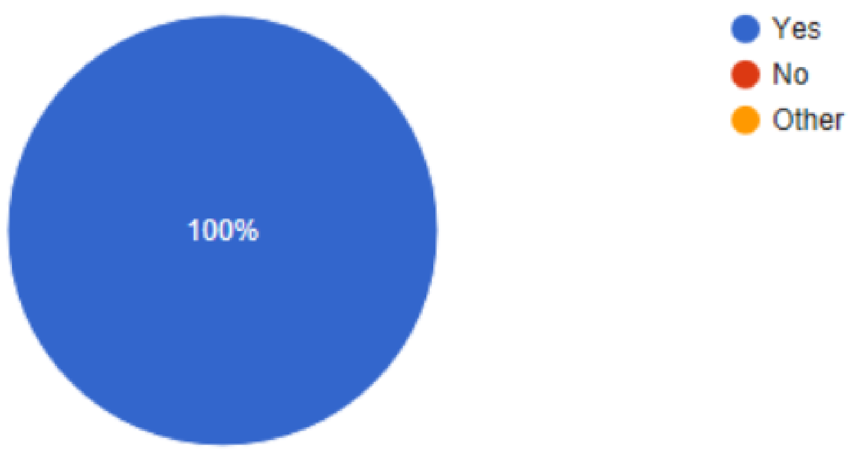
Could you interact with items as expected?



Did you like the look and feel?

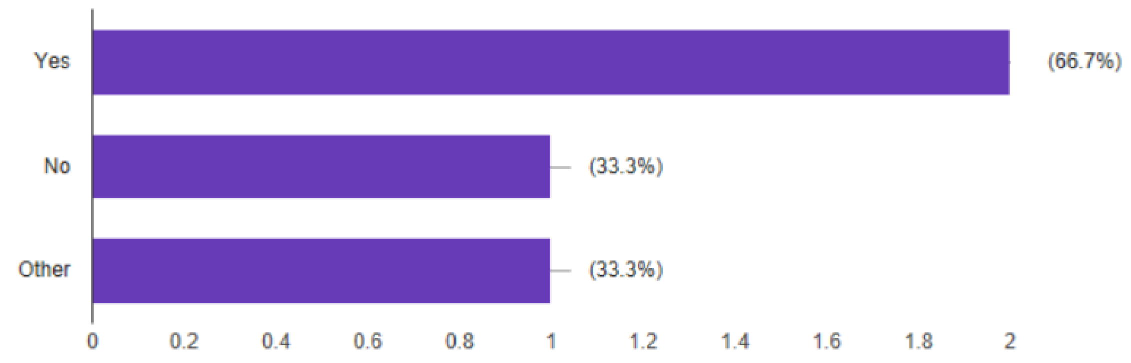


Would you like food in the game?

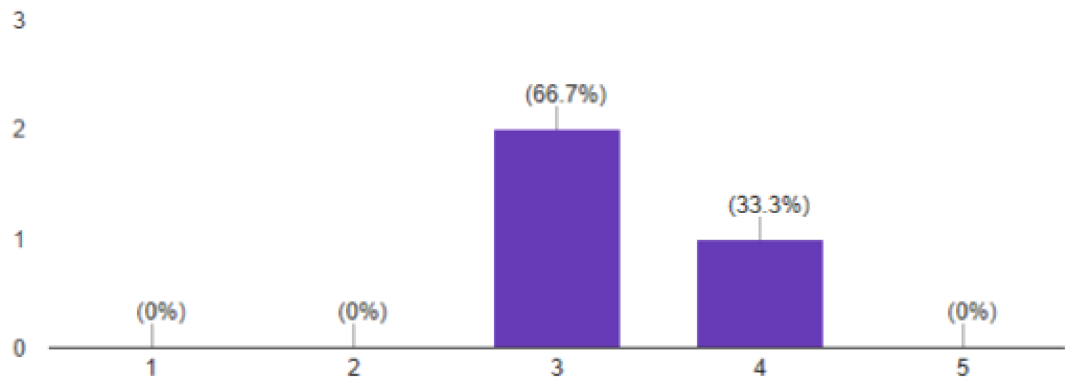


Non VR Player

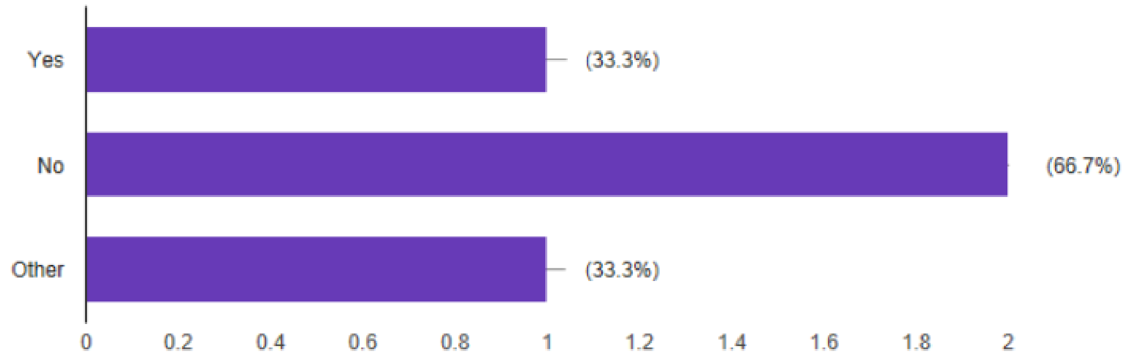
Did you like the look and feel?



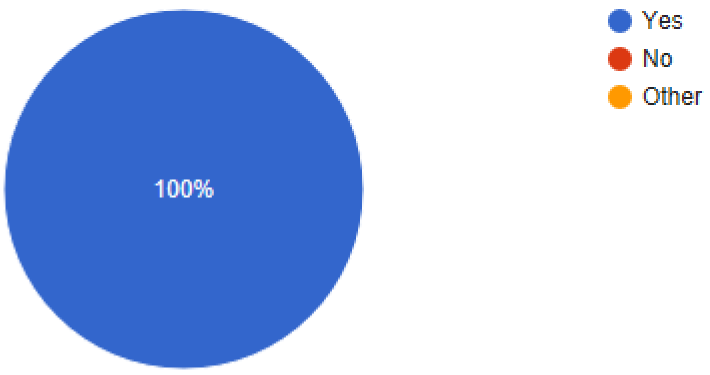
How was the monsters movement?



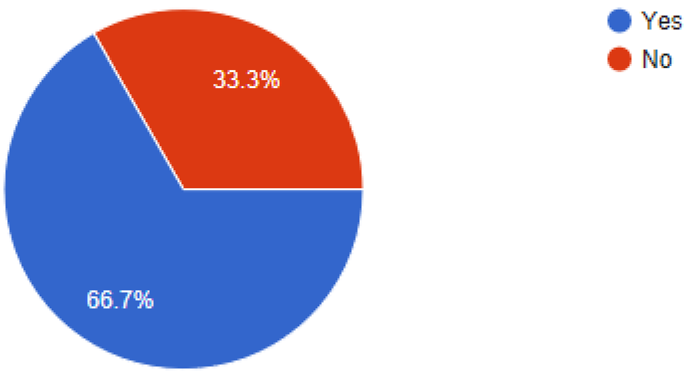
Was the control scheme intuitive?



Did you like the types of monsters?



Would you like more monster types?


Yes
No