George Hanks P17173176

Procedural Level Generation and VR - Literature Review

Procedural generation is techniques within computer science used to create data algorithmically, it is commonly used to make textures and 3D models. Procedural generation has a long history with computer games, a popular example of its use is the game Minecraft (Mojang 2009), in which a world much larger that the earth is generated and stored in a file that is only several megabytes.

(Greuter et al., 2003) introduces a method for creating potentially infinite cities in real time, the goal of this project was to be able to generate visually interesting and complex environments, and the process should be self-contained. They use seeded random number generation to decide a building's parameters, the floor plans of these buildings are created in an iterative process from randomly generated polygons. This method removes lots of control from the designer and can lead to technically different but similar feeling outcomes, this feeling is emphasised by the fact that they only generate one type of building. They conclude that such a programme is ideal for large expansive environments, including the entertainment industry, but quality can still be strengthened.

(Greuter et al., 2003) has developed a method for developing large virtual worlds using procedural generation. They emphasize the need for these worlds to be generated at real time so that they can be used for video games and the entertainment industry. They use the example of large high-rise city to demonstrate the framework they have developed. They use an iterative process to design the floor plans of the buildings and then extrude these floor plans to create the facades of a building. They make use of view frustum culling to increase performance and store the building data within a cache. They conclude by saying that the worlds they can generate can be extremely large and do not make use of external data stored on the disc, and that the worlds they create are complex and diverse.

(Tutenel et al., 2009) presents a method of generating interiors for buildings within large urban environments. They state this would improve engagement in the games and contribute to opportunities for gameplay. They highlight the need for these interiors to be almost indistinguishable from hand-made counterparts, and the designs must be varied and look lived in. To do that, they use a rule-based design solution method where the software can identify all potential furniture positions within the space based on a set of guidelines, e.g. "The couch must be facing the television and within 5 metres." They conclude by saying that the need for methods like this is increasing with the size of game worlds and expecting artists and designers to make all the content would be unrealistic. This method allows the authoring of material on a large scale, but also leaves the control in the hands of the developers who can identify the rules and items in a space.

(Lopes et al., 2010) proposes a solution for generating interior floor plans, they justify the need for such a system as current solutions limit the power the designers and artist have over the outcome, and the floor plans that are generated generally do not adhere to consistency constraints. They make use of a grid-based layout to represent the rooms of a building, like architectural drawings, these rooms can then expand and shrink based on a set of rules and constraints. They summarise by saying that there proposed method delivers valid and plausible floor plans, that the designer has control over, and that the technique could easily be implemented into a larger system for generating urban environments.

(Smelik et al., 2010) presents a method of incorporating procedural generation techniques with designer authored content. They call this method declarative modelling and they use it to generate 3D worlds. The process starts with the designer describing a 2D worlds using sketches, these sketches could be road layouts, rivers, or elevation maps. A 3D world is then generation suing these maps, this world can then be edited afterwards by the designer to refine it. They emphasize this approach as it fixes some of the problems with other procedural generation techniques, and it can be a much faster approach compared to hand made alternatives. They summarise that getting procedural generation into the mainstream is still a hurdle that needs to be addressed however this approach is a step in the right direction.

(Cardamone et al., 2011) delivers an approach for generating racing tracks with player feedback and evolutionary methods. They use search-based procedural generation to create their racetracks, allowing them to generate racetracks based on a set of parameters with certain features and characteristics. They demonstrate that such approaches will increase the speed that developers can generate and validate content, creating a large amount of diverse content that remains enjoyable and increasing the game's replay value. They summarise by saying that on this system they have received positive feedback from users, but it still needs improvements.

(Togelius et al., 2011) presents their findings in the applications of evolutionary and metaheuristic search algorithms for procedural content generation. They justify the need for a survey as the demand from players increases along with cost for artists and designers, producing content using algorithms can alleviate the need for expensive artists, allow for much larger game levels, and allow funds to be focused elsewhere. They investigate multiple techniques such as parametric vs seed-based generation and weigh up the pros and cons of such techniques. They conclude by stating that there exist many suitable techniques for procedural content generation, for certain applications, but there are many areas of the field that still need to be explored.

(Parberry et al., 2014) proposes a method for generating realistic terrain for real time applications like games. The methods they use are a mixture of Perlin noise, a very fast noise generator, and real-world elevation data to build the height maps for the terrain. They justify the need for such techniques as they can produce more realistic terrain over other techniques, it can be run at real time, it is more varied and interesting, and it can be controllable by the designers and artists. They conclude by saying that their method of terrain generation can save processing time of Perlin noise methods, it allows intuitive designer control, and it can produce interesting and varied features, however for this method to be successful they need to extract interesting real-world terrain data.

(Parberry et al., 2014) present findings in the common uses of procedural generation techniques within the games industry. They justify their need for such a survey by saying that despite the abundance of procedural content generation techniques there are not commonplace with commercial products. They describe multiple benefits to such techniques such as reduced workload on designers and artists, large game levels and it can meet the increased demand of the consumer. However, they also describe the negatives to such a design philosophy such as less control in the hands of the designers, and it can be hard to match the quality of handmade content. They conclude by saying that existing procedural generation implementations are usually limited to certain aspects of games, but in order to reach consumer demand, further research needs to be done to expand procedural content generation applications.

(Van Der Linden et al., 2013) presents multiple techniques for generating environments. They illustrate how to use cellular automata and generative grammar to construct indoor and outdoor levels that can be scaled to create large and diverse experiences. Such environments include landscape and scenery such as trees, rocks and infrastructure. They emphasise that such strategies can take power from the developers, but it allows for fast content generation, which can save a developer time and money. They also conducted a survey of the methods they have been investigating and how they can be applied to certain aspects of the game, highlighting the popular methods and usage gaps. They conclude that such strategies are ideal for environments in the style of dungeons, and the next task would be to incorporate them into player-oriented objectives.

Procedural generation techniques can be used to create almost anything for video games, (Greuter et al., 2003), (Tutenel et al., 2009), and (Lopes et al., 2010) use it for architectural purposes, in order to create a more realistic and performant environment. Furthermore, it can be used to work in conjunction with authors and artist, (Smelik et al., 2010) and (Cardamone et al., 2011), present two different methods for achieving this.

Bibliography

- Mojang 2009, Minecraft, Windows, Mojang, Sweden.
- Greuter, S., Parker, J., Stewart, N. and Leach, G., 2003, February. Real-time procedural generation of pseudo infinite'cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (pp. 87-ff). ACM.
- Greuter, S., Parker, J., Stewart, N. and Leach, G., 2003, August. Undiscovered worlds–towards a framework for real-time procedural world generation. In *Fifth International Digital Arts and Culture Conference, Melbourne, Australia* (Vol. 5, p. 5).
- Tutenel, T., Bidarra, R., Smelik, R.M. and De Kraker, K.J., 2009, June. Rule-based layout solving and its application to procedural interior generation. In *CASA Workshop on 3D Advanced Media In Gaming And Simulation*.
- Lopes, R., Tutenel, T., Smelik, R.M., De Kraker, K.J. and Bidarra, R., 2010, November. A constrained growth method for procedural floor plan generation. In *Proc. 11th Int. Conf. Intell. Games Simul* (pp. 13-20). (Greuter etal., 2003)
- Smelik, R., Tutenel, T., de Kraker, K.J. and Bidarra, R., 2010, June. Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (p. 2). ACM.
- Cardamone, L., Loiacono, D. and Lanzi, P.L., 2011, July. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (pp. 395-402). ACM.
- Togelius, J., Yannakakis, G.N., Stanley, K.O. and Browne, C., 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), pp.172-186.
- Parberry, I., 2014. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques*, *3*(1).
- Hendrikx, M., Meijer, S., Van Der Velden, J. and Iosup, A., 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, *9*(1), p.1.
- Van Der Linden, R., Lopes, R. and Bidarra, R., 2013. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(1), pp.78-89.

George Hanks P17173176

Procedural Level Generation and VR - Functional Requirements

Scenario

This software is going to be a prototype / tech demo of procedural level generation and virtual reality, it is not a fully featured or polished game. The game is set in a dungeon / crypt with a high fantasy theme, populated by different enemies. The player is a mage that is equipped with a set of spells to eliminate the occupants, combinations of these spells should hopefully create interesting synergies and emergent gameplay. There is no specific objective of the game except to explore the dungeon and survive. Procedural generation is going to be used to build the levels, with the goal of including more variety in the experiences.

Target Market / Use Cases

The software's main audience / users will be people or organizations who want to test the technology used in the demo / prototype. Game developers are always looking for unique selling points for their games, this demo would be a great way to let people experience such features. A secondary audience would be using the game for entertainment purposes, although this is the secondary goal, it is the most likely scenario as there are lots more gamers than game developers. A third but unlikely scenario would investors who want to take the prototype and then turn it into a fully featured game.

Given its long history, procedural generation is not a common tool used by game studios in games, this software could be used to demonstrate the power of such techniques and to facilitate their use in the field of game development.

Virtual reality is a common emerging technology that has yet to be fully embraced by the bigger video game market, potential users of this technology may test Virtual reality headsets with this demo and then make informed decisions about whether to invest in a virtual reality project.

As a means for entertainment this demo would provide a limited amount of playability to most users, as it will not come with all the quality of life features that a published game would, causing frustration. Furthermore, most users would likely find the game uninteresting or boring as there is no progression in the game or further end goal, it may also be unintuitive and hard to work out how to play without someone helping them.

Objectives / Rules

There is not specific goal with the demo, it is a kind of sandbox where the player can decide what they want to do, but the underlying goals are to explore the dungeon and survive. The game will end when either the player kills all the enemies within the level, or the player dies, the players dies when their health reaches zero. The player will be limited at how fat they can cast spells this is to balance the game. The enemies will also be limited in the move speed and attack rate, this will also help with balancing.

Functionality

The primary functionality of the software is to generate and populate a level, a level will consist of several rooms connected by hallways/corridors. The rooms will be filled with enemies and props for the player to interact with. A room must have an entrance so that I can be connected to rest of the level, and sometimes it may have one or more exits which will connect to more rooms. When the player starts the game, they will be able to customise the number of rooms they want in the dungeon, this will determine the size/ length of the level.

The player is equipped with spells in order to kill enemies, these spells must feel impactful to use, this can be achieved with sounds and particle effects. Spells can either be offensive, defensive, or utility, the player can decide how they want to use them. Offensive and defensive spells must be able to apply effects to enemies or

the player such as removing health or effecting move speed. Utility spells will offer unique functionality that doesn't always apply to combat. They only required spells in order to play a level will be one offensive spell and a teleport spell. They teleport spells will allow the player to move great distances and move around the level.

Enemies must offer a challenge to the player, they must be able to hinder the progress of the player by causing damage or making it difficult to fight other enemies. They way that the enemy causes damage to the player is not pre-determined but it will be a mixture of ranged projectiles or melee attacks. Enemies also add to the theme of the game, so their appearance need to be in line with a high fantasy setting.

User Experience

When the user is playing the game, they will interface with it using a virtual reality headset and controllers. They will be able to look around the environment by turning their head which will translate into the game. They can also move a small amount in game by walking around in there play space, however this is limited to the size of the play space. The player will be holding a controller in each hand, which represents the hand within the game, they will use these controllers to aim and fire spells, to navigate the level, and to interact with menus.

There will be two state within the game, the main menu and the dungeon level. The main menu will allow the player to transition to the dungeon level state via a UI interaction, it will also inform the player how to play the game via the same menu with text and illustration. The Dungeon Level is the primary state of the game, it is where the user will spend most of their time. There will be a limited user interface to allow the players to select spells, this will be in the form of a 3D menu attached to the controller, there will also be a pause menu so the player can go back to the main menu state. The amount of UI is going to be limited to increase immersion.

George Hanks P17173176

Procedural Level Generation and VR - Testing Plan

Introduction

When developing, testing software and games is an important part of the process, its primary goal being to uncover defects before reaching the end user. Defects in a game may include accidental functionality, freezing, stuttering, and crashing, all of which may spoil a player's experience and may harm a product's reputation, therefore catching them during development is very important.

Methodology

During the bulk of development there will be lots of ad hoc testing, it is common for this to be unplanned and not recorded, this is because as a developer/ programmer it is easier to solve problems on the fly. This kind of testing is common when implementing a feature for the first time, as it is not necessary to test partially complete systems or functions as the code base could change dramatically by the end.

When it comes to planned testing it will be based on the functional requirements of the project, however as the goals of the project can change this is not always the case, and new tests will need to be planned to accommodate these changes. At the start of the project the descriptions of these tests will be broad as the specifics of the how the code function will not have been ironed out. Towards the end of the project it will be possible to design more specific tests as the goals of the project won't change much, and specifics about the inner working of the code will be known.

Black Box Testing

Black box texting is a very simple way to testy if a system works, however it does not concern itself with how the systems works, all it cares about is the input and output. During this testing valid and invalid inputs will be used to create positive and negative test cases. A set of expected outputs are then created by the designer of the test to compare with the outputs. It is then very easy for someone with no knowledge of the system to test and record the outputs and compare them with the expected results.

Functionality Testing

Functionality testing is done to check if the game is performing against the specification of the project, it is a very easy way to tell if a system works as what should happen is predefined. It will investigate errors that will affect user experience such as audio, visuals, and gameplay

Play Testing

Play testing is done by having exposure to the game, its test abstract features of the game such as fun, balance, and difficulty. This testing will be mostly opinion based and anecdotal, but it is still valuable in order to build a successful project.

Regression Testing

Regression Testing is done after a code fix, or upgrade to a system or function, this is done to check that the fix has not altered the existing code. These tests will be reruns of previous test to confirm the change has the desired effect and not negatively affected the project. Depending on how often testing is done regression testing may not be necessary as the error will be picked up in the next set of tests.

Procedural Level Generation and VR – Test Cases

HTC Vive Input Test Cases

Test Name	Expected Outcome
Head Rotation	• When the player rotates their head, the camera rotates as well • The rotation is in the correct direction • The rotation is the same distance
Head Position	• When the player moves their head, the camera moves as well • The movement is in the same direction • The movement is the same distance
Controller Left / Right	• The Left and Right Controller Should Work independently from one another
Controller Position	• When the player moves the controller, the hands/controllers in the game should move • The movement is in the same direction • The movement is the same distance
Controller Rotation	• When the player rotates the controllers, the hand/ controllers in the game should rotate • The rotation is in the same direction • The rotation is the same distance
Controller Button Input	• When the player presses an input on the controller an event should trigger • A button press should trigger a single event • Each button has its own event

Spell Test Cases

Test Name	Expected Outcome
Spell Equip	• The unequip function for the old spell is executed • The old spell is unequipped from the correct hand • The old hand particle effect is removed from the correct hand • The new spell is equipped to the correct hand • The equip function runs on the new spell • The new hand particle effect is added to the correct hand
Spell Unequip	• The unequip function for the spell is executed • The spell is removed from the correct hand • The particle effect is removed from the correct hand
Spell on press event	• When the player starts pressing the input the on-press function is call a single time
Spell on hold event	• When the player holds the input the on-hold function is called every frame
Spell on release event	• When the player stops pressing the input the on-release function is called a single time

Enemy Test Cases

Test Name	Expected Outcome
Collison	• The collider for the enemy should collide with other objects in the world • The collider should be in the correct position
Character Model	• The model for the enemy should be visible • The model should be in the correct position • The model should be facing the right direction
Movement	• The object for the enemy should move • The movement should be in the correct direction

	• The movement should be correct speed
Turning	• The object for the enemy should rotate • The rotation should be in the correct direction • The rotation should be the correct speed
Target Acquisition	• If a potential target is within the range of the enemy, and the enemy has line of sight it should make it its target. • If the target goes out of line of sight the enemy, it should try to get to the last known position of the target • If the target goes outside the range of the enemy, it should try to get to the last know position of the target • If the enemy can't find the target after an amount of time the enemy should return to its spawn position.
Attack	• If the enemy has a target it should try to attack • The enemy can only attack if the target is within range • The enemy can only attack if it has been long enough since its last attack • If the target is out of range the enemy should try to reduce the distance
Health is Zero	• If the enemy's health is zero it should run the on-death function • The enemy should no longer be able to attack • The enemy should no longer be able to move • After the on-death function the enemy should not be visible • The enemy should no longer exist as an object in the game

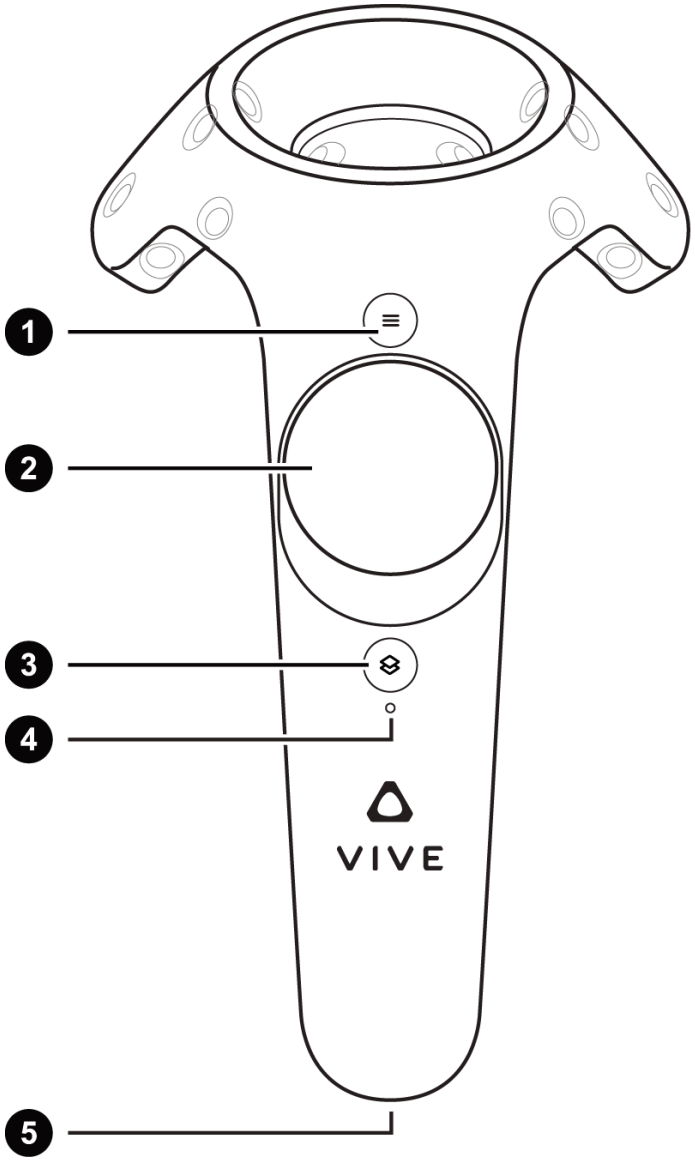
Dungeon Generation Test Cases

Test Name	Expected Outcome
Dungeon Debug	• A wireframe square should be draw at each occupied tile/ cell of the dungeon. • The size of the square represents the real size of a tile/ cell • The colour of the square should be different for each cell type
Adding a Room	• The room should be added to the dungeon data • The information is transferred correctly • The room cannot overlap another room or corridor • The room should be aligned to corridor/ hallway it was added from • The entrance door is marked as an entrance
Adding a Corridor/ Hallway	• The Corridor should start from the door of an existing room • The length of the corridor should be within the limits set by the dungeon settings • The corridor cannot overlap any other rooms or corridors • The Information about the corridor is added to the dungeon data.
Spawning Room Prefabs	• The 3D models for each room is added to the level • The correct set of models are added • The models are alleged to the tile/ cells of the dungeon • The models are in the correct position • The models are facing the correct direction
Spawning Corridor/ Hallway Prefabs	• The model for the corridor is added to the level • The correct number of corridor section are added to match the length • The sections are aligned to the tiles/cells of the dungeon • The sections are in the correct position • The sections are facing the correct direction.

Vive Controller Interface

Lookup

1. Pause Game
2. Spell Select
3. N/A
4. N/A
5. N/A
6. Hand Position
7. Cast / Activate Spell
8. N/A



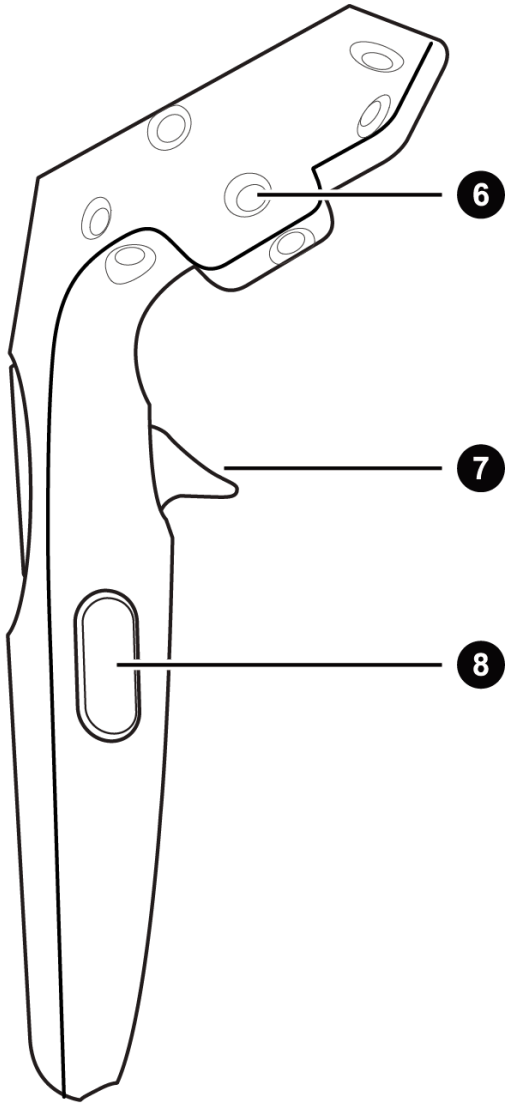
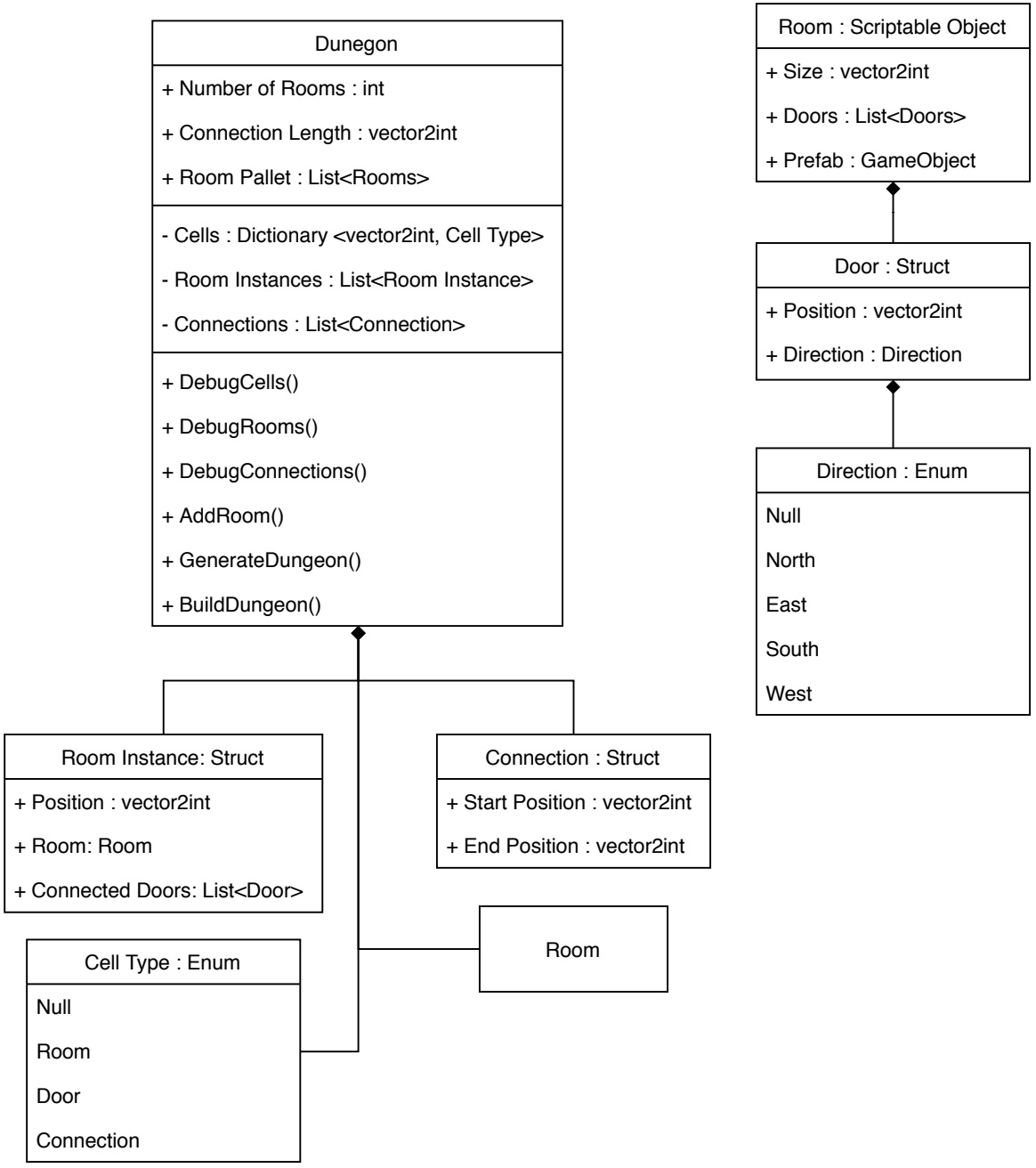
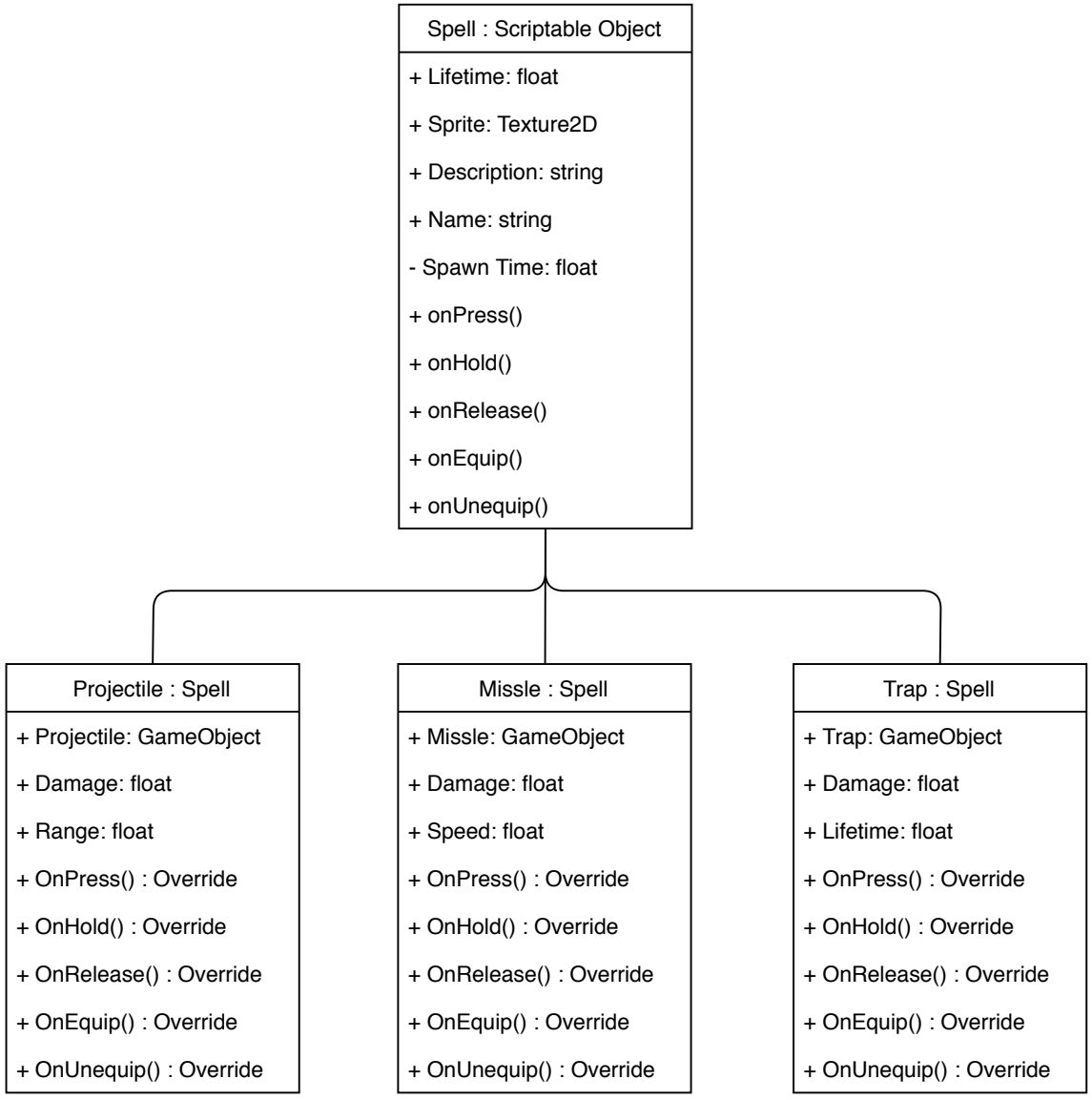


Image Source:
https://www.vive.com/eu/support/vive/category_howto/about-the-controllers.html

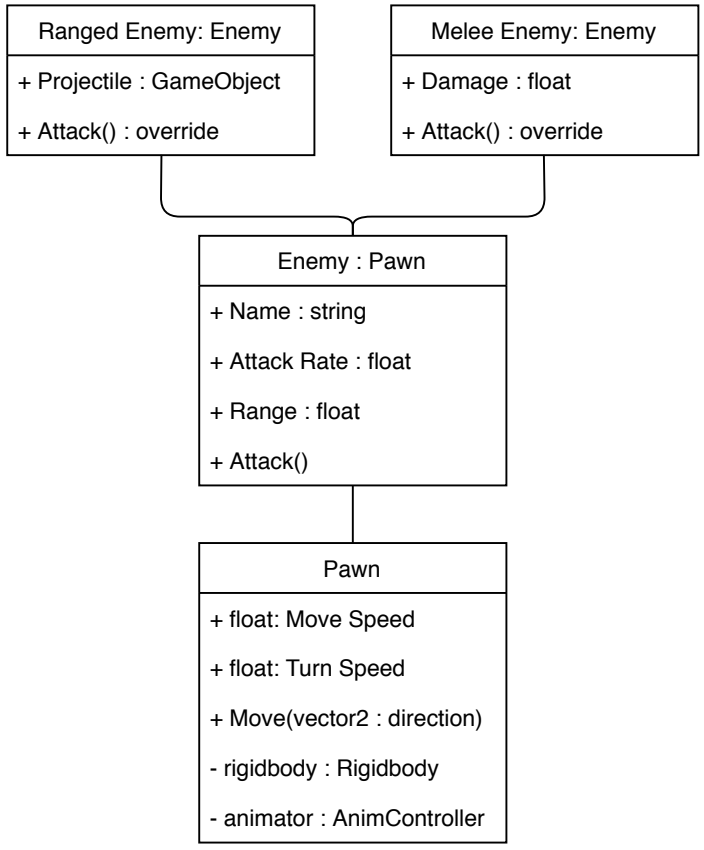
Dungeon Class UML



Spell Class UML



Enemy Class UML



George Hanks P17173176

Procedural Level Generation and VR - Implementation Report

The project so far includes player movement, the player can look and move around the scene they are in. The scene is a very basic test level, with a static enemy and a few props to interact with. The props are physics objects that can be hit, later these could be turned into more interesting gameplay elements. There is basic lighting in the scene, a single directional light.

The player can cast two spells, a teleport spell which works by aiming and pressing the trigger and a target will appear, when the player lets go of the trigger they will move instantly to that location. The player can only teleport to specific surfaces, to prevent them from glitching through walls or the floor. The teleport spell has no cooldown time.

The second spell is a basic projectile that can be used to attack the enemy it has a basic particle effect, the projectile uses a rigid body and a collider to simulate physics. There is also a basic explosion particle effect for when the projectile hits a surface or enemy. The projectile is fired when the player releases the trigger. There is also a small cooldown time for the spell.

There is a basic user interface, you can pause the game, and read the controls, the player can also press the touchpad to show the spell selection menu however this is on visual currently not functional.