Game Engines Report

by
Alexander Kirk
and Emil Buhl

IT University of Copenhagen
MGAE, F2015
Dan Lessin
December 18, 2015

Contents

1 Introduction

This paper is a technical account, describing the design and development of the game 'Redshift', using the game engine 'Unity'. The paper is split into six sections, the first of which is this introduction. In the following section, we will present the initial concept idea and the vision behind the game, followed by a section dedicated to the chronological account of the development process. The fourth section is concerned with discussing the implementation choices and alternative solutions, evaluating the final implementation. The fifth section concludes upon the work, and leaves the final section dedicated to bibliography and appendix.

1.1 Scope and Context

The development of 'Redshift' was the subject of two exam projects at the IT-University of Copenhagen, as part of the courses 'Game Design' and 'Game Engines' completed during the fall semester of 2015.

The scope of these two projects are very different, as Game Design is focused on the design of mechanics and aesthetics of games, whereas Game Engines is focused on working with game engines specifically, i.e. the implementation of features in game engines, and the related software architecture.

This paper is part of the Game Engines course, thus the subjects pertaining to the game's design and its mechanics is not within the scope of this paper. However, the focus will be on the implementation of said mechanics.

The projects done for Game Design are usually technically simple, and often done with 2D-graphics. Merging the Game Design and Game Engines projects allowed for more technically challenging features and mechanics.

The team on the Game Design project included two additional members. These members had only very limited technical experience and was assigned the roles of designers. Their tasks were mainly asset creation and level design. They did not work on the scripts, and did only limited work in the Unity editor. In the final product only one of the levels and the rotating cube special effect at the end of each level was made by one of the designers.

2 Project Description

The goal of this project was to create a puzzle game that incorporates a fourth spatial dimension, using the Unity 3D-engine.

The fourth dimension is achieved by building several 3D-environments which inhabit the same 3D-space, yet is separated on the fourth axis (W). The W-axis can be traversed in incremental steps, where each step moves the player from one 3D-environment to another.

2.1 Design and Features

The design and desired features of the game can be divided into two groups; 4D-mechanics and 3D-mechanics. This distinction separates the design into features that are unique to the game, and those which still function with the absence of a fourth spatial dimension.

A complete UML diagram of the final system architecture can be found in appendix, figure 2.

2.1.1 3D-Mechanics

These are the desired features which are not dependent on the existence of a fourth spatial dimension. The following list aims to define and clarify each feature and their dependence on other parts of the design.

- 3D-FPS movement:
  A player character controlled from a first person perspective using the keyboard and mouse, allowing the player to traverse, perceive, and interact with the game world. Unity offers several built-in prefabs consisting of game objects, cameras, and scripts which fulfils this role.

- Picking up and moving objects:
  The player must be able to pick up certain game objects. When an object is picked up, it must be held in front of the player character until dropped, allowing the player to move it from one place in the game world to another. Unity Lessons[1] offered scripts for a basic implementation that allows for a single object to be picked up and moved. However it has to be expanded, allowing for more complex objects to be used, eg. several objects linked as one. It also had to be tweaked since the original scripts made a carried object kinematic, allowing it go through walls.

- Moving objects:
  Certain objects like doors must be able to move or rotate. Using a single generic script allows for both movement, rotation, and scaling of objects.

- Interactable objects:
  The player must be able to interact with certain game objects, such as buttons which can be used with various effects, e.g. opening a door or calling an elevator. The interactable objects must also have the ability to be locked, requiring a certain key to become unlocked. The interactable objects must be implemented such that they can be used in combination with other parts of the design. Implementing both a generic script and an interface allows for easy interaction between objects

and offers a template such that special cases are easily implemented with the rest of the design.

- Display/Monitor:
  In-game displays, like a computer monitor or TV-screen, is a requirement for more advanced puzzles relying on multiple buttons and combinations, e.g. the player has to operate a machine. This can be achieved in the Unity editor alone, using built in cameras and render textures.

- UI Mini-map:
  A mini-map is required in order to help the player navigate the game world, offering a top down view of the environment. Using cameras and render textures as with the in-game displays, this is achieved inside the Unity editor, suspending a downwards facing camera over the player and linking it to a render texture in the UI.

2.1.2 4D-Mechanics

- Collisions:
  Objects in the same 3D-space, but on different points along the fourth axis, must not collide, meaning that even though two objects are intersecting in the 3D-environment they should only collide if they also intersect on the fourth dimension. Using Unity's 3D-collision detection and expanding it through the layers system (making collision layers), an incremental 4th axis can be achieved. This requires objects to change layers according to their W position for the collisions to work correctly. Using scripts to add a W coordinate and handle the layering creates a fairly simple implementation. However, many objects require this mechanism, thus it is important for it to be optimized in terms of processing time, memory usage, and accessibility in level design. Therefore, a script which can handle multiple nested game objects was also required (see appendix, figure 3 for appropriate UML).

- Moving:
  In relation to collisions and interaction, the player controller must also change layer according to its W-position, while granting the player an ability to change their W-position. This requires that the mechanism used for detecting interactable objects is filtered according to W-position.

- Rendering/Colouring:
  Objects must be rendered differently dependant on their W-position, or the player is able to see everything at all times, with no obvious way to distinguish what walls the player will collide with. Changing colours and fading objects, making them transparent or completely invisible, are different ways of helping the player perceive the game's 4D nature. Using scripts to detect the change of the players W-position, and changing the rendering of game objects accordingly, facilitates the world changing around the player when they traverse the fourth axis.

3 Week Structure

In this section, we will describe the time table and weekly progress throughout the project. We will go over problems and some design decisions, especially those that had consequences for the overarching work flow.

3.1 Time Table

This time table reflects who took responsibility of what tasks. The table itself was developed without any assignment of responsibilities before beginning the project.

Week number	Task	Owner
31-10-2015	4D-Movement	Both
	Colour switching	Alex
	Pickupable Objects	Emil
7-11-2015	Composite objects, tuning of carrying implementation	Emil
	Interactive objects	Alex
14-11-2015	AI, Hazards and In-game displays	Both
21-11-2015	Doors and locks, Refactoring of code	Alex
	Elevator, level design	Emil
28-11-2015	4D-lighting, sound effects	Alex
05-12-2015	Polish	Both

Table 1: Work schedule separated into weeks

3.2 Breakdown

3.2.1 Week one

Initially, we needed to decide upon what approach to take in regards to the core mechanics of our game. Using the standard asset 'FPSController' shipping with Unity, we only needed a way to explore the fourth dimension. We figured that this movement would take on either of two possibilities; integral or fluid.

To better explore these possibilities, Emil started working with the integral approach, while Alex explored the fluid approach.

Our efforts revealed that a fluid approach would be much more faithful to the mathematical concept of dimensional movement, but would be exceedingly complicated to implement properly in the Unity engine, since the collision detection is handled by Unity behind the scenes. Fluid 4D-movement would require the collision detection system to be extended with a fourth coordinate, which was simply not feasible.

On the other hand, integral movement - being the more 'unrealistic' approach - would be much easier to implement even without compromising the core idea of the game. Additionally, it complimented the collision detection system of Unity through the use of collision layers, which with little interpretation could be transformed into a working 4D-collision engine.

With the movement system in place, Emil implemented the first version of pickupable objects, inspired by unitylessons.com[1]. Meanwhile, Alex implemented the first version of the colour switching algorithms.

3.2.2 Week two

During the second week, we started working with especially composite objects; objects that stretched across multiple collision layers (had a w-width larger than 1). These objects proved to become a problem, given that they seemingly did not follow the laws of physics, detaching from each other seemingly at random.

Investigating, we concluded that this was a problem founded in our PickupObject script[citation needed], where we forced the displacement of carried objects instead of applying forces to it, hence bypassing the inherent physics inside Unity. This problem also meant that objects could displace through walls. Emil began solving these issues, while Alex started working with interactive elements, such as buttons.

Multiple problems were fixed during the week, with both issues related to PickupObject being solved. Button functionality was also added to the code. Regrettably a new problem appeared, where carried objects started floating around the player erratically, increasing in speed as it went. This problem was solved later in the process.

In the end, composite objects achieved a working state. While they did carry some complexity during implementation, their behaviour was reasonable.

3.2.3 Week three

With most of our core functionality implemented, we made an attempt to implement a simple AI and in-game monitors.

The AI was thought to be used as an opponent, but was never fully implemented to fulfil this position. Likewise, the in-game monitors were thought to relay information that was not obviously available to the player.

Due to misunderstandings, we thought the week to be imminently before feature freeze and focused our efforts implementing whatever features we found was still lacking. This reasoning was of course flawed, since we had no need for neither an AI or in-game monitors, and these never appeared in the final product.

The implementation of these minor, irrelevant features took time away from refactoring of our currently implemented code, which could have spared us many troubles later in the development.

3.2.4 Week four

Having gathered all of the features we deemed necessary, we started this week by distributing work between us. Alex became responsible for refactoring all of the currently available code, additionally creating the first implementation of the doors and locks that would later be used. Emil switched focus, and started development of content and designing levels, and was responsible for implementing the first elevator used in the game.

This week was undoubtedly the worst in work flow, and arguably catastrophic for the end result.

Given that the code implementation up until this point had been very specific as to its use, most of the refactoring was concerned with making the code more generic and

easily accessible for the designers. Yet unintentionally, the refactoring created a multitude of new problems, breaking all of the previous functionality and thereby rendering most of the code useless. While it did fulfill its purpose regarding core functionality, many tangential features were utterly useless after the refactoring, e.g. the elevator and socketing system were broken, and buttons were volatile.

Upon realizing the situation, we decided to start fixing a multitude of problems, eventually resulting in working, albeit unstable, code. The fixes included correctly rendering trees and bushes, reimplementing pickupables and inventory items, fixing interaction between multiple 4D-objects, adding additional algorithms for animated doors and similar objects, and fixing the floating of carried objects.

An overview of the changes can be found in appendix, figure 1. All files contained within the folder 'Scripts_v02' were created from this week forward, and all scripts contained in the 'Scripts' folder were frozen.

3.2.5 Week five

While our time table suggested that we were to add 4D-lighting and sound effects, most of the fifth week went by with content creation and bug fixing.

Being behind mostly on content, we opted to attempt to create as much as possible within this week. Sadly, our code was less generic than expected, and the designers on our team had a hard time applying functionality to their scenes, which in turn made content generation very slow.

When week five concluded, we had a sketch of what we envisioned, but had not been able to apply most of our planned features. The sketch was a level consisting of three layers, where the objective was to carry a teddy bear through the level.

This level was made to be a beta-presentation of the game for 'Game Design', and hence was tied heavily by time.

3.2.6 Week six

During the beta-presentation, we were panned by our lack of game puzzles and features. For this reason, we decided to drastically change approach.

Instead of a "One-large-level" approach, we switched towards multiple scenes that were strung together loosely, more akin to what one would experience in Portal[2] (see appendix, figure 4).

This meant that we scrapped most of our previous levels in an attempt to recreate a better representation of the game, resulting in what the final product looks like.

The five levels present in the final game was designed and created during this week, four of which Alexander or Emil created - Alexander one, Emil three. A joint effort was also put in for some needed last minute bug fixing related to the door animations and interactable buttons.

Play tests showed a need for a mini-map, which Emil created in the Unity editor.

4 Discussion

4.1 Alternate solutions

Throughout our project, multiple choices were made to further development. In this section, we will discuss alternatives to our chosen solutions.

4.1.1 4D-transformations and -movement

Instead of moving between points on the fourth dimension, we could instead have used a more fluid movement system, which we considered during the first week of development. This approach lends much more freedom to the player, and creates a true 4D environment, which in design terms better communicate our chosen message.

Fluid movement would mean less replication of environments, but more complex collision calculations. The trade-off here would be hard to measure concretely, since a fluid collision detection system was never implemented properly, but it would apply pressure to the algorithms instead of the object count. Additionally, only one complex collision detection algorithm would be necessary, instead of the multiple needed as part of composite objects used in our final solution.

This movement could include correct mathematical rotation around all geometric planes, with resulting complicated, yet interesting, puzzle and game play opportunities. One could argue these transformations would be unnecessary from a design perspective, given their overtly complex and incomprehensible nature.

While a technically interesting challenge, the design itself would be a struggle of understanding and applying mathematical concepts to properly fit a game play in which it would be highly unnecessary. The approach we used lends itself especially well to game play due to its relative ease of understanding and appliance, two qualities which the fluid solution does not possess.

4.1.2 Transparency of 4D objects

We decided to use transparency as a means to visualize the fourth dimension. We did consider colour, and used some elements of that consideration in our final design.

The transparency is handled by employing functions within the standard shader Unity uses, in which options are changed when the player moves along the fourth dimension. This enables objects to become opaque and fully transparent, depending on relative distance to the player character (still on the fourth dimension).

Our implementation does need some optimization. Currently, the materials have their colour changed such that the transparency applies, but this does not exclude even fully transparent objects from being drawn during the rendering passes. Unity also needs every object to be non-static in order to change the rendering options correctly, which tolls the graphics with a large amount of unnecessary obfuscated polygons.

Due to the way Unity handles rendering passes, the problems could be mended by changing the properties of materials that appear in one space. Additionally, a custom shader or likewise implementation could be used to skip the rendering passes of the completely invisible meshes, drastically reducing the amount of rendered polygons within a

scene.

5 Conclusion

The final product differs from our vision in that we have multiple small levels and rather simple puzzles instead of one large level featured complicated puzzles. This deviation is largely constituted to an unrealistic scope and setbacks during development. The technical capabilities of the game supports most of the envisioned features, however time limited the level design and the content creation and in the end many ideas - like elevators, machines and displays - were not implemented even though the functionality is present.

Much is still left to be desired especially from a Game Design perspective, but from a technical point of view the game almost fulfils our goals, as our work with the Unity engine has concluded in a 4d-puzzle game albeit we are not demonstrating the full potential of our features.

6 Bibliography

References

[1] unitylessons.com. 2014. *Unity3D Tutorial - Pickup and Carry Objects.* [Online]. [Accessed 16 December 2015]. Available from: `https://youtu.be/runW-mf1UH0`

[2] *Portal* (standard edition). 2007. PC / Mac, Playstation® 3, Xbox 360 [Game]. Valve: Bellevue, Washington.

[3] Klow. 2009. *Testchmb03.jpg.* [Online]. [Accessed 17 December 2015]. Available from: `http://half-life.wikia.com/wiki/File:Testchmb03.jpg`

7 Appendix

7.1 Project files

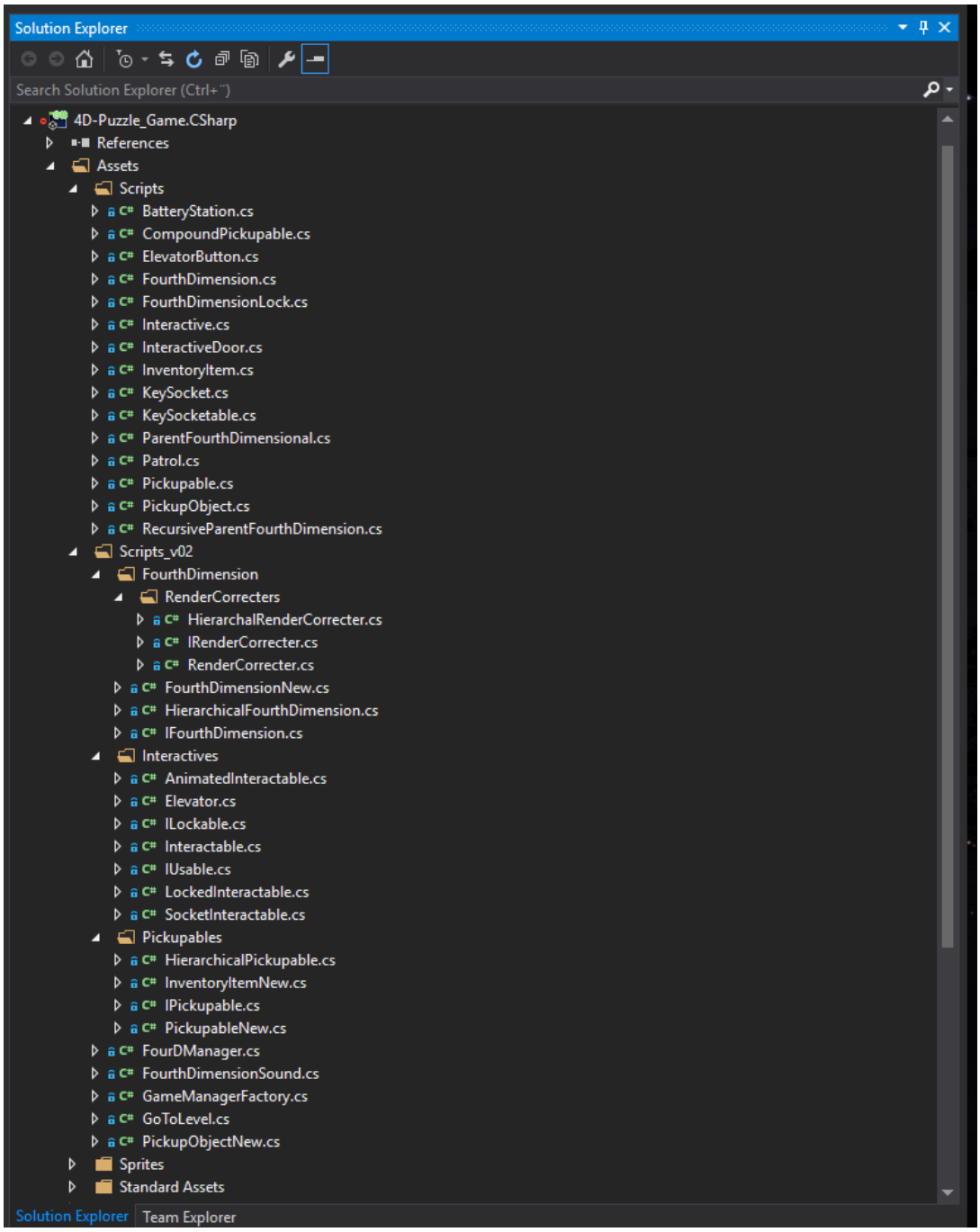

Figure 1: All project files created throughout the course.

7.2 Figures

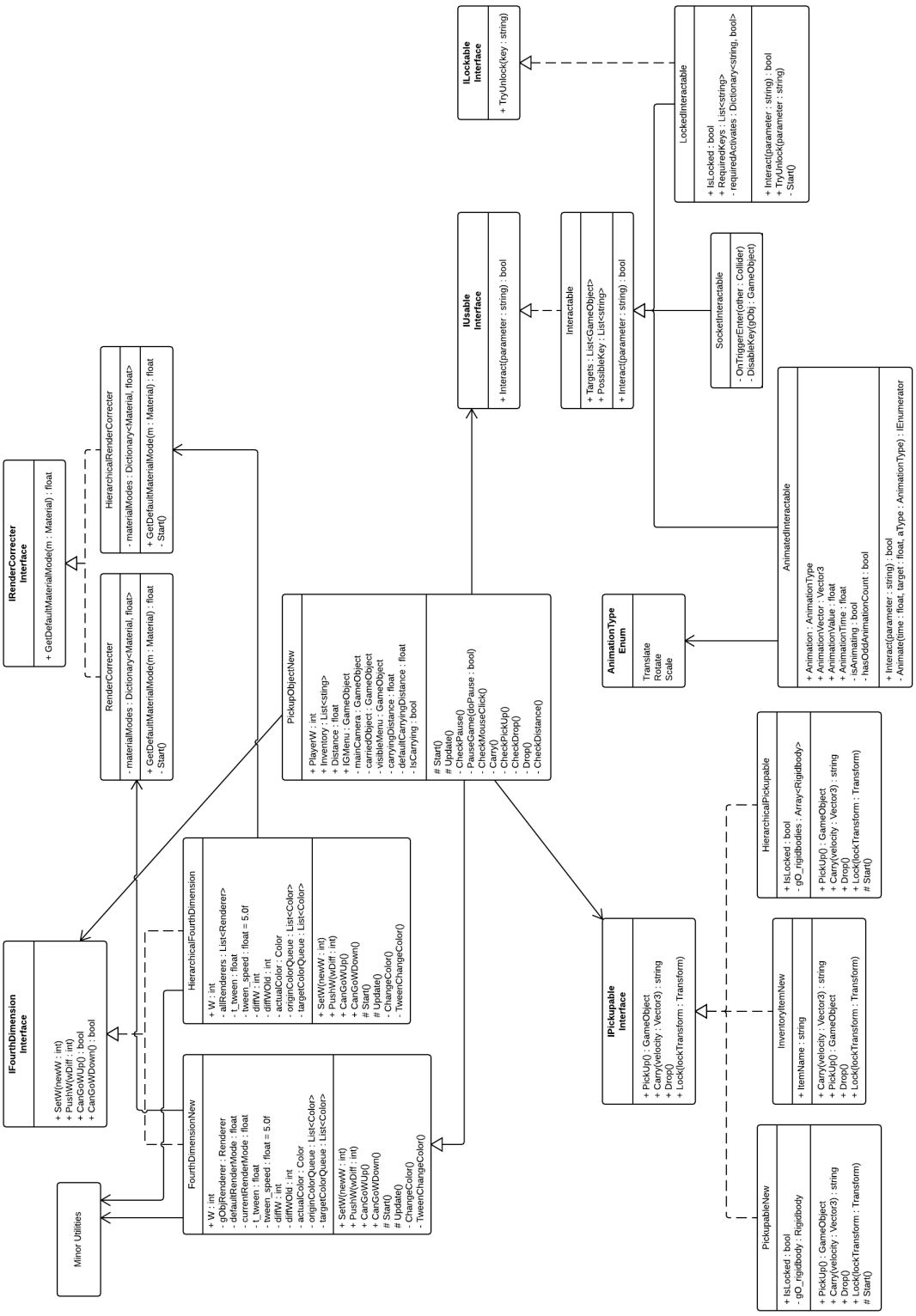

Figure 2: The entirety of our system architecture, expressed in UML

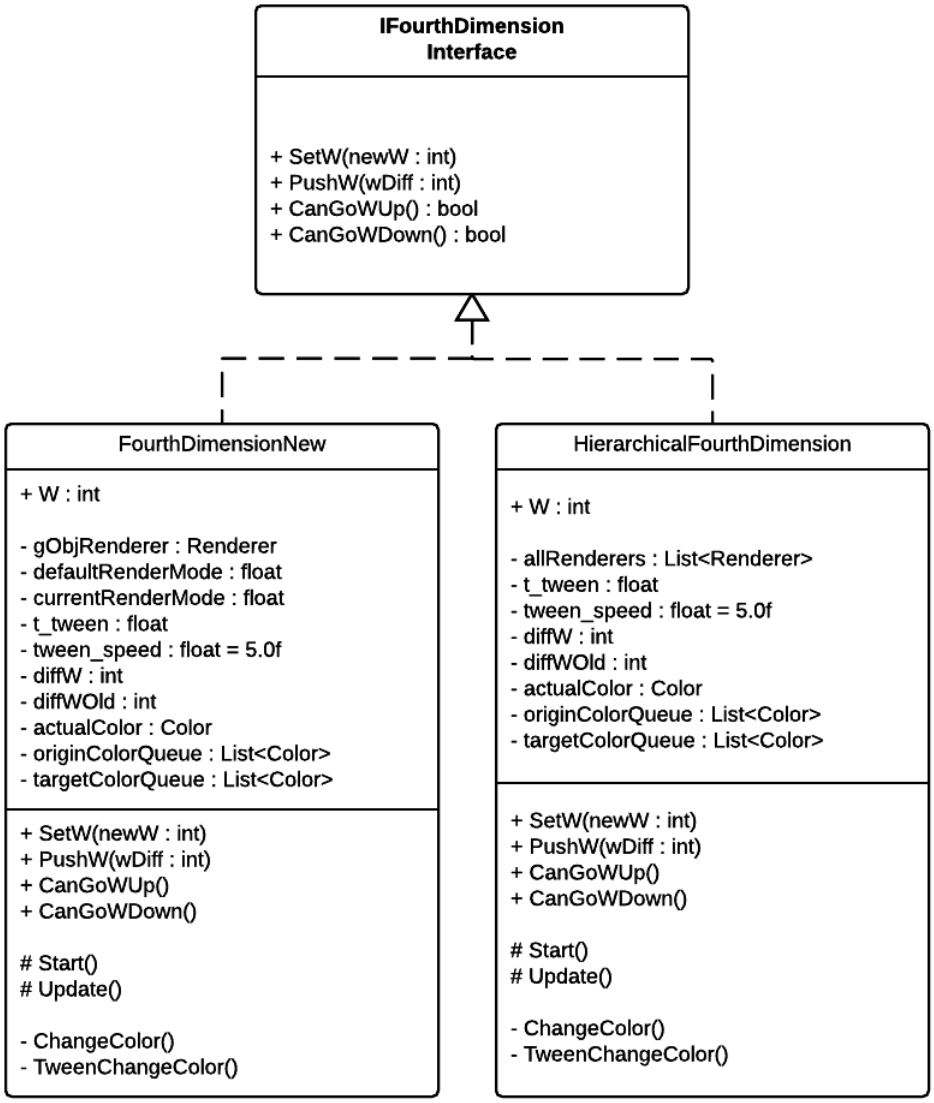

Figure 3: The FourthDimension architecture, expressed in UML

Figure 4: A test chamber, as seen in Portal.[3]