School of Electronics and  
Electrical Engineering and  
Computer Science  
ECS Building 1  
1 Elmwood Avenue  
Belfast  
BT9 6AY

**Masters of Engineering in Computer Games Development**

**Software Development Report**

CSC4006

**Visual interfaces effect on player performance in virtual reality**

Author Timothy O’Neill [40062164]

Project supervisor Dr Darryl Stewart

Programme MEng Computer Games Development

Date 06 May 2016

Contents

[1. Introduction 3](#_Toc449730278)

[2. System Specification 3](#_Toc449730279)

[3. Hardware Requirements 6](#_Toc449730280)

[4. Design 6](#_Toc449730281)

[4.1 Interface Design [3] 7](#_Toc449730282)

[4.1.1 Interfaces wireframes and general design 7](#_Toc449730283)

[4.2 Player Movement 8](#_Toc449730284)

[4.2.1 Rail Movement *[5]* 9](#_Toc449730285)

[4.2.2 Separated Head Movement 9](#_Toc449730286)

[4.2.3 Connected Head Movement 9](#_Toc449730287)

[4.3 Metrics Tracking 9](#_Toc449730288)

[5. Implementation and testing 10](#_Toc449730289)

[5.1 Software development process 10](#_Toc449730290)

[5.2 Coding Standards 10](#_Toc449730291)

[5.3 Engine Choice 10](#_Toc449730292)

[5.3.1 Unity 5 11](#_Toc449730293)

[5.3.2 Bespoke 11](#_Toc449730294)

[5.3.3 Unreal Engine 4 11](#_Toc449730295)

[5.4 Oculus Rift and Unreal Engine 11](#_Toc449730296)

[5.5 Developed Classes 12](#_Toc449730297)

[5.6 Blueprints 14](#_Toc449730298)

[5.6.1 Blueprints for Widgets 14](#_Toc449730299)

[5.6.2 Blueprints for HUD’s 15](#_Toc449730300)

[5.6.3 Diegetic Interface 15](#_Toc449730301)

[5.6.4 Extensibility of the system 16](#_Toc449730302)

[6. Testing 17](#_Toc449730303)

[6.1 Automated tests 17](#_Toc449730304)

[6.2 Experiment test runs 17](#_Toc449730305)

[6.3 Gameplay test plan 18](#_Toc449730306)

[7. Conclusions 19](#_Toc449730307)

[7.1 Future work 20](#_Toc449730308)

[References 20](#_Toc449730309)

1. Introduction

To explore fully the impact of different visual interfaces on a player’s performance in virtual reality a software artefact must be created, it must be able to provide the necessary results to help us reach a conclusion on the research topic proposed.

The artefact to be produced must allow a user to perform a predefined task using both a virtual reality headset and without the headset, the experiment should take the form of a game to keep the user engaged and interested in the task to be performed, the game must be exactly the same on both mediums thus the only varying factor is whether they are using the headset or not, each of the following interface groups should be implemented and be identical in both mediums, spatial, non-diegetic, diegetic and meta, a fifth interface should take the form of making the user play the game with no interface at all, the system should be recording various metrics in the background for analyse so that the research will be able to draw a strong conclusion.

The details of how this artefact will be designed, implemented and tested is outlined in the following document.

1. System Specification

The system to be developed should take the form of a simple game revolving around the user moving between locations guided by waypoints while dodging incoming projectiles, the user will be presented with five different interfaces to aid them in their task and metrics about their performance should be recorded for the researchers benefit.

The System will be used by two stakeholder groups myself as the researcher and the subject of the experiment the user, as such I have drawn up a set of functional requirements presented as use cases for both groups as outlined below in Table 1,2, use cases are simple concise statements that help us manage at high level of what will need to be completed in the project and to what standard, they are the most preferable method due to their simplicity, while not shown here each user story had is dependencies assessed and were put into the order they needed to be completed in on the project board, how the acceptance tests are signed off is covered in the testing section.

|  |  |  |
| --- | --- | --- |
| **User requirements** | | **Acceptance test** |
| 1. Player movement | As a player I want to be able to move around the game world in order to avoid the projectiles and complete my objective. | The Player must be able to move around the game world in an acceptable manner. |
| 2. Player camera | As a player I want to be able to view the game world using the Virtual reality headset or a traditional monitor. | The player must be able to see the game world with VR or without VR. |
| 3. Player asset | As a player I want a visual representation of my character to view while playing the game. | The player model appears correctly. |
| 4. Projectile movement | As a player I want the projectiles to move to various locations to make me have to makes choices on how to avoid them as they approach. | The projectiles correctly move towards their designated targets. |
| 5. Projectile collision | As a player I want the projectiles to collide with me and the game world so to create a game of avoiding being hit by the asteroids. | The projectiles correctly collide with entities in the game world. |
| 6. Projectile spawner | As a player I want multiple asteroids to come from various directions to create a more challenging experience. | Multiple asteroids appear from different locations. |
| 7. Projectile asset | As a player I want the projectiles to have an asset to create an immersive and real experience and to help me easily identify them. | The asteroid model appears correctly. |
| 8. Diegetic interface | As a player I want to be provided with a diegetic interface so I can use it to help me perform my task. | The diegetic interface appears correctly is easily understood and correctly shows entities. |
| 9. Non diegetic interface | As a player I want to be provided with a non-diegetic interface so I can use it to help me perform my task. | The non-diegetic interface appears correctly is easily understood and correctly show entities. |
| 10. Meta interface | As a player I want to be provided with a meta interface so I can use it to help me perform my task. | The meta interface appears correctly is easily understood and correctly show entities. |
| 11. Spatial interface | As a player I want to be provided with a spatial interface so I can use it to help me perform my task. | The spatial interface appears correctly is easily understood and correctly show entities. |
| 12. Interface assets | As a player I want the interfaces to be presented well so they convey the information clearly and in an understandable form. | The assets for the various interfaces look professional and correct. |

Table . User requirements and Acceptance testing.

|  |  |  |
| --- | --- | --- |
| **Developer Requirements** | | **Acceptance test** |
| 13. View switcher | As developer I want to have a means to switch between the normal view and the Virtual reality view at runtime. | You can switch to the VR headset easily at runtime without any restarting of the game. |
| 14. Menu system | As a developer I want a menu system to be able to switch between the various game modes, tutorial and experiment configurations at runtime. | You can easily define menus and they can be navigated and are easy to understand and use. |
| 15. Game timer | As a developer I want to be able to control the length of time a player has to complete a task and also how much time has passed since the scenario began. | The timers correctly countdown and report the time that has elapsed. |
| 16. Game mode | As a developer I want to be able to easily define the rules of the player’s task so there are easily followed and extendible in the system. | The rules are easily implemented and changed, the game follows the rules as outlined. |
| 17. Demo mode | As a developer I want the player to be allowed a practice run of the game mode to enable them to familiarise themselves with the game itself. | The rules are easily implemented and changed, the game follows the rules as outlined. |
| 18. Metric tracking | As a developer I want to be able to track metrics in order to make judgements about user performance. | Metrics can be recorded in an easy manner with very little function calls and written in a sensible manner to file. |
| 19. Interface manager | As a developer I want an easy way to switch between the various interfaces to make the program easy to manage and maintain. | Allows for easy switching between the various interfaces, and manages the clean-up of the interfaces. |
| 20. Tutorial | As a developer I want the player to easily understand the task they need to fulfil and the entities of the game therefore a tutorial is required to achieve this. | The tutorial is easily understood by the player and conveys all the necessary information. |
| 21. Projectile types | As a developer I want the projectiles to vary in order to create a more interesting experience for the player and enable more diverse tracking of metrics. | Different projectile types correctly spawn and have varied behaviour such as different targets. |
| 22. Game arena | As a developer I want the game to take place in a controlled arena that allows me to restrict the movement of the player a certain amount to make sure the experiment is consistent and challenging. | The arena is correctly defined and stops the player from leaving it. |
| 23. No interface | As a developer I want to provide the player no interface to perform the task in order to draw a richer conclusion on the subject matter. | The game functions as expected and can be played with no interface. |

Table . Developer requirements and Acceptance testing.

1. Hardware Requirements

The virtual reality headset chosen to be used for conducting the experiment has been the Oculus rift due to its availability and how well it is supported by multiple development environments, specifically we will be using an Oculus Rift Development kit two the specification and hardware requirements of which can be seen in Table 3.

|  |  |
| --- | --- |
| **Oculus Rift Dk2 Specification[1]** | |
| Resolution | 960x1080 per eye |
| Refresh rate | 75Hz, 72Hz, 60Hz |
| **Minimum System Requirements[2]** | |
| Operating system | Windows 7,8 or 8.1, 10 64bit |
| Processor | 2.5+ Ghz Processor |
| Ram | 4GB |
| GPU | AMD Radeon HD 6950 or above  Or  NVIDIA GeForce GTX 560 or above |

Table . Hardware constraints of the project.

As we can see the DK2 runs at a maximum of 75Hz therefore we need to produce a performant piece of software that will maintain a constant Frame Rate of 75 frames per second to be able to keep a consistent experience for the test subject.

1. Design

In traditional software when producing a system there would be a plethora of system designs to draw from, however in this artefact we are producing the gameplay for the experiment and in its nature this is unique, therefore the following UML has been created by my own assumptions of what will be needed in the artefact to perform the experiment, Fg1 shows the diagram.



Figure 1. The basic outline of the gameplay classes needed for the game.

* 1. Interface Design [3]

For the experiment the player will use the five interfaces outlined above excluding “No Interface” each of these must be designed in such a way that they make sense both on an Oculus and a standard monitor, the following section highlights the design choices for these interfaces.

4.1.1 Interfaces wireframes and general design

The interfaces should follow a consistent theme, the projectiles will be identified by red colours and the players objective by green, any interface that isn’t based in the game world should also show a dot to indicate the player shown in yellow, the interfaces should never be intrusive as to reduce the viewable game world of the player, in addition the interfaces should always be relative to the players position updating with their movement and rotation to allow the player to be aware of their relative surroundings.

For the game we have had to decide on four interfaces from the different classifications of diegetic, non-diegetic, meta and spatial while there were more considerations than I can go into detail here through personal knowledge and refinement four games were selected for direct inspiration when it came to the interfaces Fg 2 shows these games.

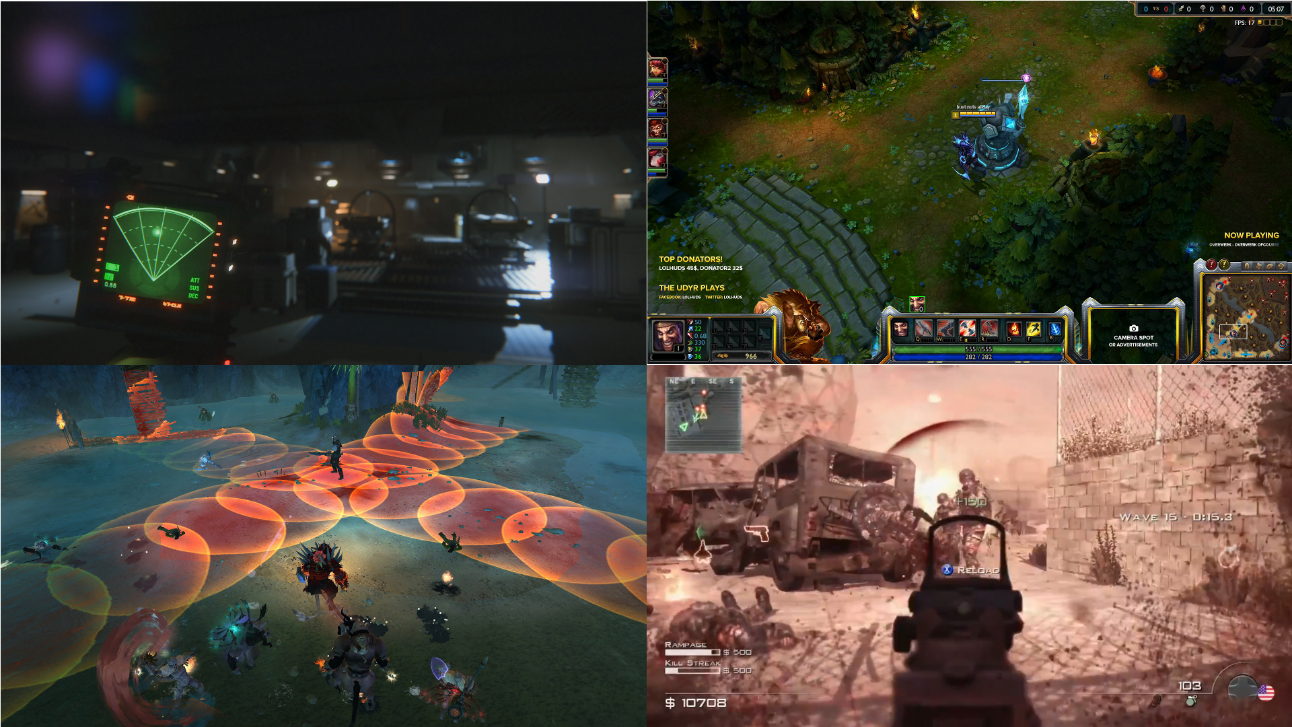


Figure . Top Left, diegetic Alien Isolation[4], Top right, non diegetic league of legends minimap  
Bottom left, spatial wild star telegraphs, Bottom right, meta call of duty enemy direction.

From these selections wireframes were created to be used as reference when creating the interfaces, Fg 3 shows the generated wireframes, both of these combined will be useful references when developing the interfaces with a clear goal and outcome in mind.

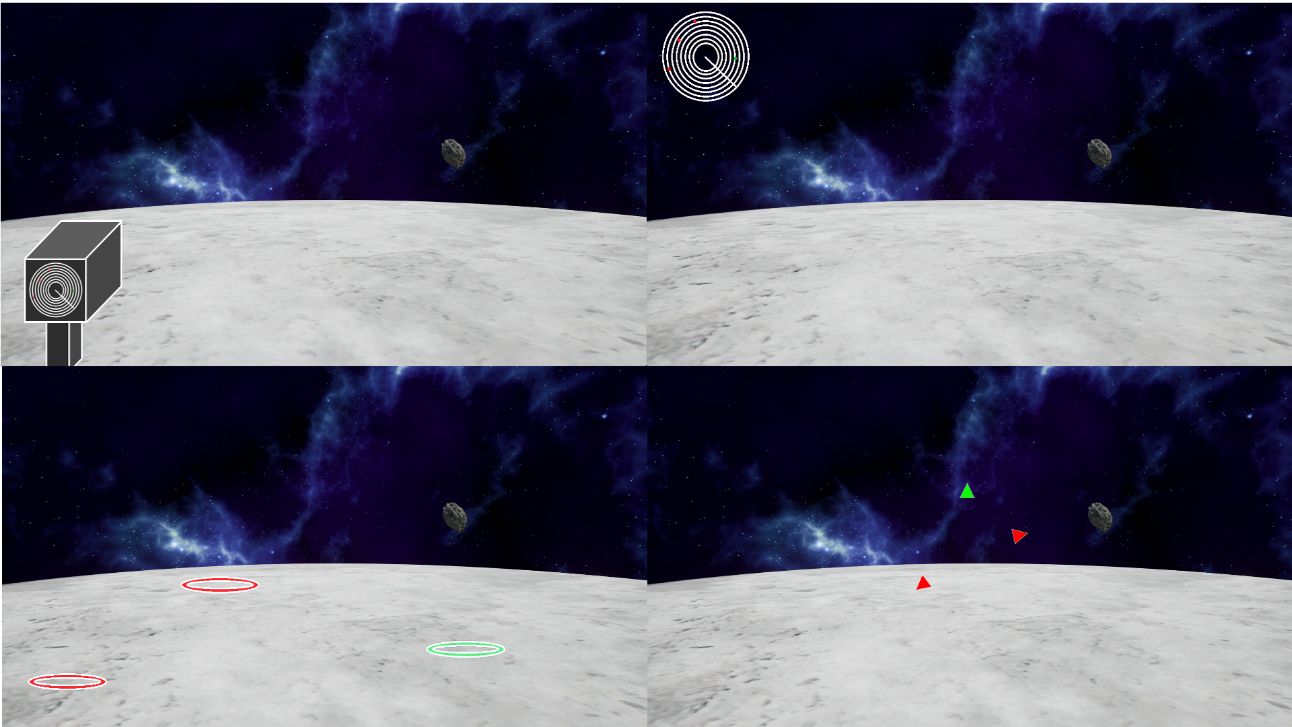


Figure . Top Left diegetic, Top Right non-diegetic  
bottom left spatial, bottom right meta.

4.2 Player Movement

Due to the infancy of virtual reality many core gaming systems are still in the teasing stage none more so that player movement, many games currently ported to VR have all very different control schemes for their players, for the design of player movement I will present previous existing games that have different control schemes all using an Oculus rift and a Xbox360 controller.

4.2.1 Rail Movement *[5]*

Many old arcade shooters used a system where the player was moved between locations and presented with enemies to shoot, this has made a resurgence in VR, Bullet train [5] a VR exclusive built in unreal engine uses this system, the player can move to fixed teleport locations and engage with enemies in their current view, this system while having some initial learning of where to teleport provides a clean immersive experience allowing for the user to perform the current task without getting taken out of the experience.

4.2.2 Separated Head Movement

This is the mostly commonly used control system for ported games, TF2 has many control systems available for VR however using its standard controls of player movement and then allowing the Oculus user to look in any direction leads to a certain familiarity for hard-core players however the mixture of the two for non-gamers can be a steep learning curve meaning that it is harder for them to just pick up and play with the Oculus.

4.2.3 Connected Head Movement

This is the simplest form of player movement allowing for the player to move around in the direction they are looking, the oculus controls the rotation of the player in the game world and such any forward movement will be in that given direction, this may be negatively received by hard-core participants but ultimately it is the most intuitive for non-gamers and hard-core alike, this allows any user to jump in and instantly use the Oculus.

4.3 Metrics Tracking

The metric tracking class will be implemented using a singleton pattern, this has several advantages to it, the metrics tracking will only have ever one instance since we are writing files to disk this will help make sure there’s no clashes on file writes or accesses, it also makes it globally available to other classes allowing them to write metrics specific to them with minimal coding required.

Due to the nature of the experiment each interface that the user is performing with will have metrics tracked against it, the number of metrics being tracked will be relatively small therefore it is reasonable to store them in memory until that particular interface is finished and then write the results to disk, this will save constant disk read writes and be a small performance optimization for our piece of software.

1. Implementation and testing

The following section outlines the final implementation of the artefact and the testing performed during the development of the project.

5.1 Software development process

Agile Kanban was chosen as the methodology to manage the project, Kanban is perfectly suited for this style of project as does not set dates to deliverables and instead only signs them off when they are finished, this suits a university style project where different priorities can lead to development not being consistent, all of the Kanban process was tracked using Trello and open source notification board which was perfect for the needs of this project Fg 4 shows the Trello board [8].

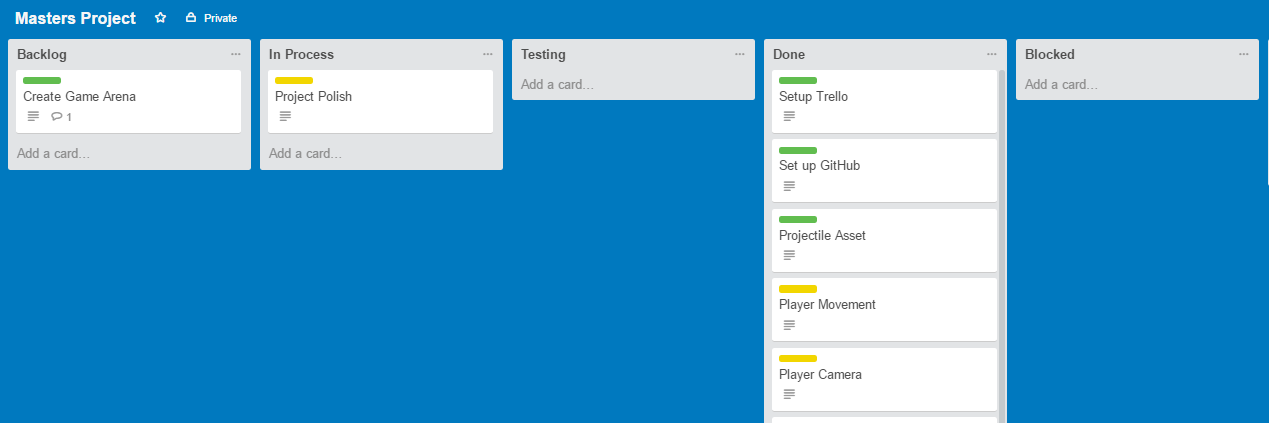


Figure . Trello board of the project.

5.2 Coding Standards

Every project needs a set of coding standards in order to maintain readability Google’s coding standards [10] were closely followed, also in addition some other practices picked up from my professional career were added such as always giving a default value to basic types in headers and using spaces instead of tabs, leads to having a clean and consistent code base, this allows for easy reading for other users and future work.

5.3 Engine Choice

For undertaking any games development project choosing a game engine foundation is fundamental to producing a solid artefact, while there is many options available I made the decision ultimately between three.

5.3.1 Unity 5

Unity comes with Oculus integration built in it also allows for fast game development due to it handling a lot of the low level systems leaving the high level details of the game to the user, unity gameplay functionality is primarily written in C# while I have previous experience in the language it would not be my preference. Unity is very well documented and therefore problems faced during development could easily be addressed, also the initial learning curve of picking up the engine would be simple.

5.3.2 Bespoke

Producing my own framework to work in is an option, using the Oculus Sdk I could easily interface with its functionality, it would also give me the freedom to choose the programming language I would wish to use, however producing the gameplay on top of a game engines core functionality would require an excessive amount of time and add unnecessary complexity, to an already large project.

5.3.3 Unreal Engine 4

Unreal engine 4 similar to unity comes with Oculus Integration built in, it too offers tools in order to handle rapid development of games, the primary language with which to produce gameplay is a mixture of two things, C++ programming and a Unreal visual scripting language called blueprints, C++ is my preferred language, blueprints are very useful to accomplish simple tasks such as menu navigation, they also could simplify creation of the game interfaces allowing for easy control of the visual presentation, one downside of unreal engine is that documentation is not widely available and can be outdated, also the C++ programming has very few tutorials due to the infancy of this most recent distribution of the platform, this would make the initial learning and development in Unreal potentially tricky.

With all the above considered the artefact was chosen to be produced in Unreal Engine 4 using a mixture of blueprints and C++, due to the language it uses and the integration it already has with the Oculus along with the suite of tools it provides to produce games.

5.4 Oculus Rift and Unreal Engine

The Unreal engine out of the box has a plugin to easily enable usage of the game engine with the Oculus rift, the unreal engine team has implemented the various technical pieces of the Oculus SDK into their game engine and thus makes the usage in our project much simpler, any created game in unreal engine can be launched into VR mode allowing us to easily test and iterate on the project.

Key functions are accessed using C++ through the HMDDevice reference, a list of the available functions can be found here [6], this was also a motive behind choosing unreal as it allowed for rapid development in what was a rather short period for development.

5.5 Developed Classes

The following UML Diagrams outline the main C++ and Blueprint classes and their composition. 

Figure . Player classes.



Figure . Interface classes



Figure . Game mode classes



Figure . Projectile classes



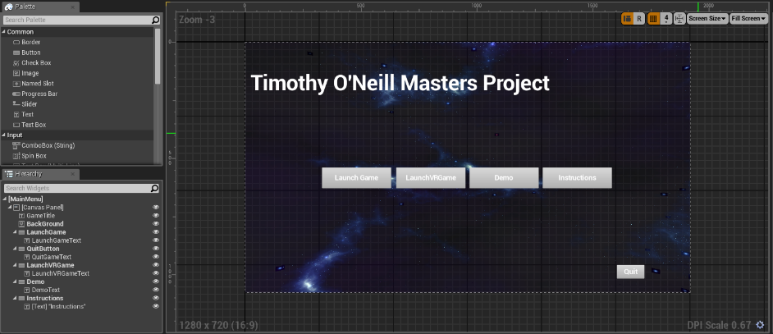
Figure . General classes.



Figure . Blueprint Classes.

5.6 Blueprints

Blueprints were used in areas where visuals where primarily required compared to coding, another use of the blueprints is for example shown with the asteroids, because they are being spawned at runtime it made it difficult to set their model using the normal Ue4 tools therefore by defining it in a blueprint the C++ code can pick this up at runtime making for a much tidier reusable code base, other classes also use this technique for their various meshes and visual assets.

5.6.1 Blueprints for Widgets

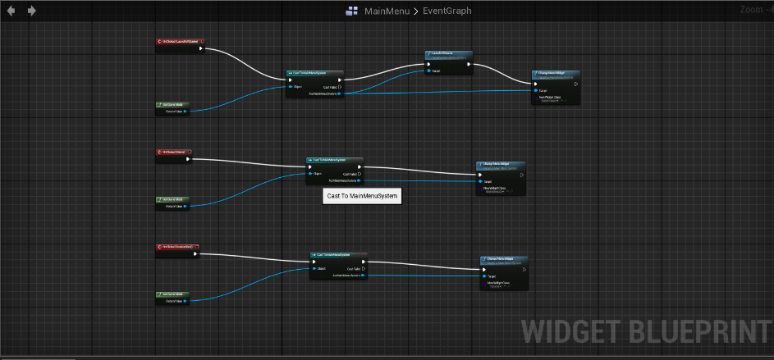
As mentioned before one of the main uses of blueprints was for visual elements, the various menu systems of the program where visually created using the unreal tools, then a light amount of scripting to move between menus was done in the background, an example of this is shown in Fg 11,12. This shows the overall main menu blueprint and the individual segments of visual scripting, as you can see the various buttons and components of the interface can be easily laid out and designed to suit the interface and their event functionality defined in the visual scripting of blueprints, this removes any complexity of using C++. The performance over head of using blueprints [8] is negligible for small processes such as a button event and as such is perfectly suited for this kind of functionality.

Figure . Main menu visual.

Figure 12. Main menu code.

5.6.2 Blueprints for HUD’s

Two out of the four HUD’s are made purely by Blueprints, the rationale behind this choice is similar to menu widgets discussed in section 5.3.3, we can easily define how we want the interface to look in the design section of the blueprint, and then we can define its behaviour in the graph section. This allows for rapid prototyping and easy to make interfaces, unreal allows for the ability to easily switch between HUD’s at runtime by calling.

APlayerController::ClientSetHUD(TSubclassOf<class AHUD> NewHUDClass);

Both the Non Diegetic and Meta interfaces are created using blueprints.

5.6.3 Diegetic Interface

The diegetic interface is probably the most complex of all the interface implementations, it comprises of several components.

* C++ class controlling when it renders and how it attaches to the player model.
* Widget class that controls how the interface appears visually.
* Actor class for the hand held model.
* Actor class to place the widget in making it a 3D widget it so it can be placed anywhere in the world.

Fg 13. Shows the various components of the interface.



Figure 13. Diegetic interface, top left shows debug view,  
 top right model, bottom left 3D interface, bottom right final product.

The interface is dynamically spawned in at runtime and attached to a socket on the players hand, because of this it takes on all location and rotation movement of the player meaning it appears consistently in the players hand, one of the key points of the diegetic interface is the ability for the user to look away from it, this is achieved by allowing the user to look up and down at the interface in the Oculus and default monitor, by using Unreal’s 3d interfaces I was able to simply combining the different components together to make a seamless interface.

5.6.4 Extensibility of the system

Adding additional interfaces to the system is a trivial matter as we have seen in section 5.3.2 blueprints offer excellent use when creating the visual representation of an interface, once the new interface is created it’s only a matter of creating a new object of the interface type in the interface manager and adding it to the ENUM list of the interface manager, then we can make requests for the interface to switched to where ever needed in the code base, by creating interfaces in this manner we do not need to add any excessive classes to our code base.

enum class EInterfaceEnum : uint8

{

IE\_SpatialInterface,

IE\_DiegeticInterface,

IE\_NonDiegeticInterface,

IE\_MetaInterface,

IE\_NoInterface,

IE\_MyNewInterface

};

Figure . Function calls to add a new interface.

Interfaces->ChangeInterface(EInterfaceEnum::IE\_MyNewInterface);

1. Testing

The following section outlines the testing performed on the system, this includes automated tests provided by unreal engine, test runs of the experiment and a gameplay test plan.

6.1 Automated tests

Another benefit of using Unreal engine is the inclusion of a base set of automated tests, these cover many areas of any game and were applicable for this project.

Several key tests were needed for the system, level loading, blueprint compilation and Oculus rift performance,

As stated before the system needs to be able to run at 75 frames per second to make sure the tester does not suffer from motion sickness or have a poor experience, also making sure the various blueprints we have for the game compile correctly is key, Fg 14, shows the automated test interface of Unreal.

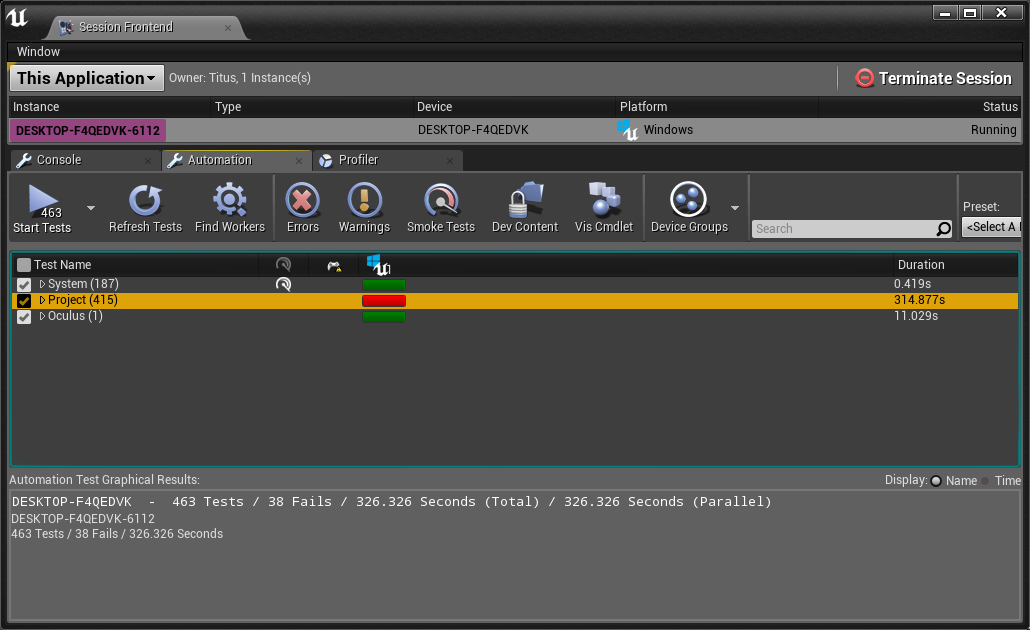


Figure 14. Automated Tests.

From the report you can see how many tests failed out right and how many threw up warnings these tests can be investigated and iterated on

6.2 Experiment test runs

Near the completion of the project dry runs of the experiment were performed, this provided valuable feedback and improvements for the project as a whole, a few things this highlighted was the need for a better objective than just avoiding asteroids, the inclusion of the waypoints for the player to move between helped to this end, the waypoints also make the player have to move through areas of the game arena that they might have otherwise avoided increasing their the chance they will have to dodge incoming projectiles.

In addition to this it became apparent that verbally explaining the task the subject was to perform was not enough, therefore the tutorial page was put together in order to help a user visually be able to know what their purpose was, in addition to the demo they are allowed to play.

Other issues discovered while testing revolved around running the project on machines outside the development environment, driver issues, bugs and crashes were identified and fixed.

6.3 Gameplay test plan

When producing games unit tests and automated tests can only go so far due to the large amount of integration in a game the best form of testing comes in the form of gameplay testing, a test plan was created for each of the systems in the game, Fg 15, shows one example of the test plan.

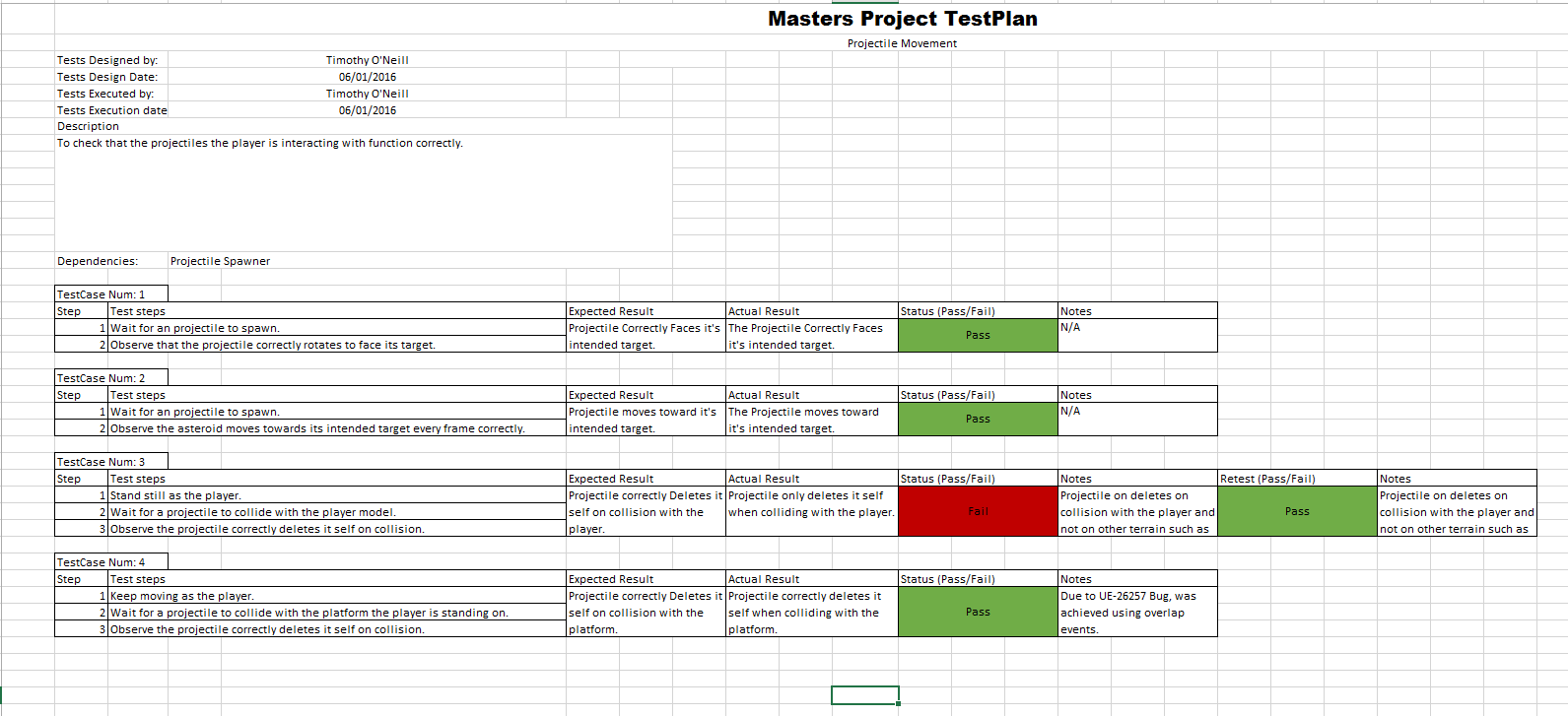


Figure 15. Gameplay test plan.

Each test is defined by a series of steps to be completed followed by the expected out come and then the actual outcome, each test is given a pass or fail any additional notes are appended at the end, this test plan follows the critical path of a user and as such tests the key components of the given area, the acceptance tests are signed off once a given area has a full set of passed tests.

Any bugs found during testing or development were reported on the trello board for the project [7] bugs are marked with minor, major or critical to order their priority, Fg 18, shows an example bug report.

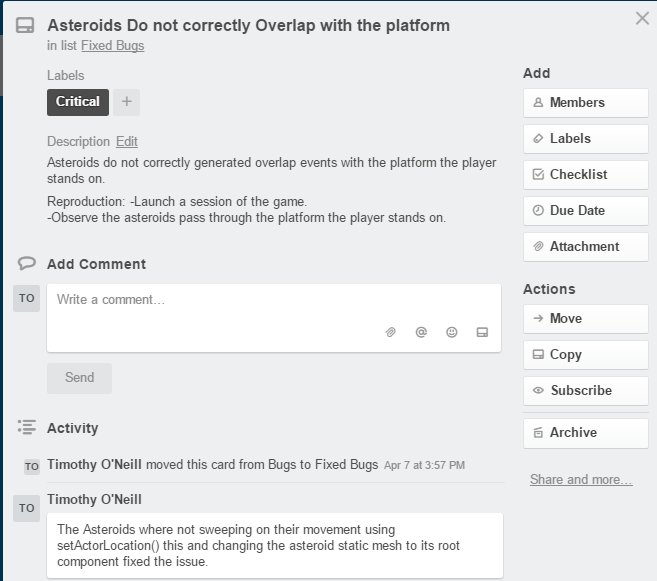


Figure 18. Trello bug report.

Using these testing procedures lead to the project being made in a robust manner allowing for ease of use during the experiment and longevity of the code base.

1. Conclusions

The project as whole was completed over a course of four months the commit logs on GitHub [9] Fg 19 show this.

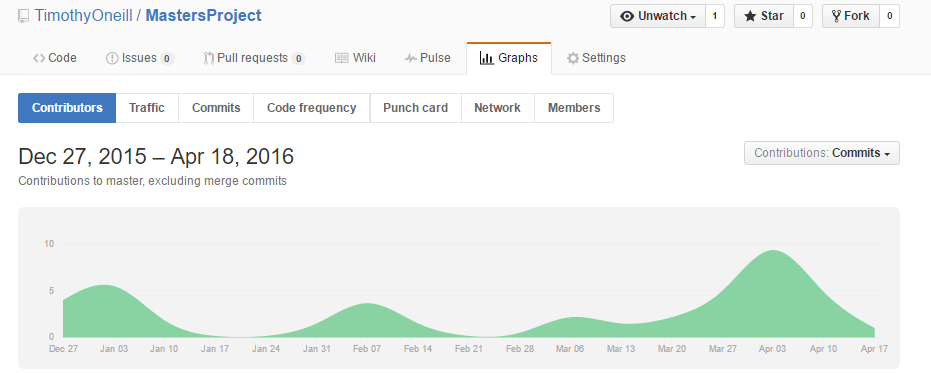


Figure 19. Commit log of project.

The project has presented interesting problems and challenges such as learning Unreal Engine and VR all in the same boat, the original set out functionality was created with no compromises and fully delivered, this included completion of the test plan, following this all experimental runs were conducted, in all it has been a challenging fulfilling project and I have been able to learn and use my previous skills successfully.

7.1 Future work

From experiment runs one of the biggest pieces of feedback was the addition of some sort of feedback on how the player was performing be it a score or a visual cue for when they get hit by a projectile, the reason this didn’t get implemented into the current build was time constraints and design flaws, feedback to the player would have to be in the form of the current interface being shown meaning interfaces like spatial would be difficult to design and the time investment to solve the problem was too great.

Another improvement that would help further reinforce the research would more testing scenarios outside the current setup, such as giving the player different objectives than just collect and dodge or even trying different view types such as 3rd person this would help identify the best interfaces for each camera type as well as cross validate the existing results.

References

[1] "Oculus Rift Specs - DK1 Vs DK2 Comparison - Rift Info". *Rift Info*. N.p., 2016. Web. 28 Apr. 2016.

[2] "Developer Center — Documentation And Sdks | Oculus". *Developer.oculus.com*. N.p., 2016. Web. 28 Apr. 2016.

[3] "User Interface Design In Video Games". *thewanderlust.net*. N.p., 2010. Web. 28 Apr. 2016.

[4] "Alien: Isolation". *Alienisolation.com*. N.p., 2016. Web. 28 Apr. 2016.

[5] "Bullet Train - Unreal Engine 4 Official Tech Demo". *YouTube*. N.p., 2016. Web. 28 Apr. 2016.

[6] "Unreal Engine | Iheadmounteddisplay". *Docs.unrealengine.com*. N.p., 2016. Web. 28 Apr. 2016.

[7] Trello.com*/b/j1z4YbdA/masters-project*. N.p., 2016. Web. 28 Apr. 2016.

[8] "Blueprint FAQ And Tips - Epic Wiki". *Wiki.unrealengine.com*. N.p., 2016. Web. 28 Apr. 2016.

[9] "Timothyoneill/Mastersproject". *GitHub*. N.p., 2016. Web. 28 Apr. 2016.

[10] "Google C++ Style Guide". *Google.github.io*. N.p., 2016. Web. 28 Apr. 2016.