

**Lost Cause**

**Report 1**

**Students:** Stephen Long & Jamie Hegarty

**Student Numbers:** 20035687 & 20053535

**Supervisor:** Brendan Lyng

**Course:** BSc (Homs) in Entertainment Systems

**Year:** 2014-15

Contents

[1.0 Game Overview 2](#_Toc402744383)

[1.1 Game Concept 2](#_Toc402744384)

[1.2 Feature Set 2](#_Toc402744385)

[1.3 Genre 2](#_Toc402744386)

[1.4 Target Audience 3](#_Toc402744387)

[1.5 Game Flow Summary 3](#_Toc402744388)

[1.6 User Interface 3](#_Toc402744389)

[2.0 What already exists 4](#_Toc402744390)

[3.0 Methodology 5](#_Toc402744391)

[4.0 Technical 7](#_Toc402744392)

[4.1 Target Hardware 7](#_Toc402744393)

[4.2 Development hardware and software 7](#_Toc402744394)

[4.3 Development procedure and standards 9](#_Toc402744395)

[4.4 Game Engine 9](#_Toc402744396)

[4.5 Network 10](#_Toc402744397)

[4.6 Languages 11](#_Toc402744398)

[4.7 Version Control 11](#_Toc402744399)

[4.8 User story management 12](#_Toc402744400)

[5.0 Iteration 1 13](#_Toc402744401)

[5.1 Defining Iteration 1 objectives 13](#_Toc402744402)

[5.2 Iteration 1 development 13](#_Toc402744403)

[5.3 Iteration 1 learning experiences 17](#_Toc402744404)

[6.0 Management 18](#_Toc402744405)

[6.1 Detailed Schedule 18](#_Toc402744406)

[6.2 Budget 19](#_Toc402744407)

[7.3 User Stories 20](#_Toc402744408)

[7.4 Risk 21](#_Toc402744409)

[7.4.1 Skills Risk 21](#_Toc402744410)

[7.4.2 Team risk 22](#_Toc402744411)

[7.4.3 Technological Risks 22](#_Toc402744412)

[References 24](#_Toc402744413)

Report 1 – Technological Feasibility

# Game Overview

## Game Concept

After surviving a plane crash, you find yourself stranded on an island. You don’t know where you are, or if help will ever arrive. With darkness looming, and the ever present danger of the unknown, you have but one goal, TO SURVIVE. Lost Cause is a co-op survival experience that uses the latest in VR technology to fully immerse players in an ever changing game world that tests both their wits, courage, and desire to survive.

## Feature Set

* Virtual reality experience via the Oculus Rift
* Play Co-Op with a friend in multiplayer mode
* Selection of puzzles
* Xbox 360 controller support
* Collectable resources, managed via a fluid inventory system
* Intuitive craft system
* Player crafted tools, weapons and structures
* 3D player models with animations
* Skills and experience system
* Ability to save/load your progress
* Dynamic Player UI that includes hunger, thirst, stamina, sanity, and health meters
* Full audio effects
* Dynamic environment with day/night cycle
* Voice chat for ease of communication while wearing the rift
* Interactive map and mini map to aid in world traversal
* Fog of war map system
* Fully interactable, inhabited game world
* Complex enemy AI
* Discoverable story line
* Motion blur features
* Intuitive Combat System
* Procedurally Generated Maps

## Genre

Virtual Reality survival game

## Target Audience

*Lost Cause* is targeted towards PC gamers that have access to an Oculus Rift. It is not a “pick up and play” type game, but more so a gaming experience that invites users to manage their resources and consider consequences to their actions. The game is targeted towards gamers above the age of twelve as it contains elements of violence. The game is not targeted specifically towards males or females, as we feel that anyone can have an interest in survivalist type games.

## Game Flow Summary

The game begins with players emerging from the wreckage of a plane crash. Initially players have a limited number of items, and must collect resources in order to survive. As players progress through the game, exploring the island, and collecting resource in order to survive, they will advance their character attributes, allowing them to progress further into the island. The game is an open world survival game and so it does not contain level by level progression. There are a number of ways in which the game can end. Players can be either successful or unsuccessful in their attempt to escape the island. The end state is determined by the decisions made by the player as they progress through the game.

## User Interface

**Look:**

Lost Cause is a 3D survival game played from a first person perspective. The game is played via the oculus rift, which helps give the game a more immersive feel as players are not only looking at a screen containing the game world, they are actually encapsulated within it. The game will adopt a realistic graphical style in order to further immerse players in the game world. The game will be set on an island which will itself contain a vast array of environments ranging from sandy shores to heavily forested areas. Each environment will have its own distinct characteristics, with for example, forested areas having a much darker tone in contrast to a brightly lit beach area. Each environment will also differ in wildlife and ecology adding to the immersiveness of the game world. Our aim is to create a game that leaves a lasting impression on the player. As the game puts a huge emphasis on survival and isolation, we decided that a more realistic graphics style would best complement those key elements, especially seeing as the game will be played via the oculus rift. We felt that a graphic style such as *cell shading* would not be suitable to a game such as *Lost Cause* as the cartoon-like style would not compliment a game that tries to highlight the difficulties of survival in a foreign wilderness.

**Feel:**

The feel will be that of a lone survivor trying to survive a seemingly inhospitable habitat of an unexplored island, while desperately awaiting rescue. The player will feel an increasing sense of helplessness as resources dwindle and the challenge of survival increases. As the player progresses, the story and the history of the island will become clearer. There will be a strong emphasis on ambiance and immersion, as the island will be thriving with life.

There are a number of core aesthetics we aim to evoke from the players of our game. We want players to feel that they are not only controlling an avatar, but that they are themselves part of the struggle to survive. We also want players to really become encapsulated by the ecology of the game. One of our main goals in designing this game is to really cause the players to feel a sense of isolation. Evoking such player aesthetics is crucial to our gameplay. We want players to genuinely feel lost and isolated within the game world.

**Visual Style:**

As mentioned above in the *Look* section, we have chosen to adopt a photo realistic graphical style. This visual style best compliments the survival based gameplay. We felt that any other graphical style would cause the game to come across less life like, and so we agreed that a photo realistic visual style was crucial to create an authentic survival simulation. As one of our main goals is to create environments in which players can become fully immersed, we agreed that the environments themselves need to look as realistic as possible.

We created a short prototype scene that shows the visual style we aim to apply to our game. Shown below are some screenshots from that prototype scene. This scene is not reflective of our overall product, but is used to give an impression as to the visual style we are going to apply to the game.



The image above shows a prototype air crash scene.

# What already exists

The Oculus Rift is a relatively new technology that is not yet even available commercially. As a result there is not an extensive selection of games available. There have been a number of games ported to include Oculus Rift functionality, however the amount of multiplayer games available for the rift is extremely limited. There are a number of survival games available across multiple different platforms, however these tend to mostly be single player experiences. Whereas our aim is to create a multiplayer survival experience that is playable via the Oculus Rift.

There are a number of demos available that display some features and applications of the rift. These features can range from accelerometer usage to different methods of menu traversal. We intend to reference some of these demos in order to make the gameplay experience as fluid and engaging as possible. There are a number of SDK’s and open source projects that help developers incorporate Oculus Rift compatibility into their games, and so we intend to closely reference these throughout the course of development.

In relation to networking, we have chosen to use the Photon Unity Networking solution. There is documentation available on how to integrate the Photon Server into Unity based projects, and so we will closely reference this documentation when developing the multiplayer aspects of the game.

As mentioned above, there are a number of survival games available for different platforms. One such game is The Forest. The Forest is a single player survival game released for PC in 2014. This game has a similar premise to the game we intend to create, whereby the player is stranded in a woodland area, and their primary goal is survival. Our game is also predominantly set in a forested landscape, and so we may choose to draw on elements of The Forest for inspiration. Although there are a number of similarities between both our game, and The Forest, there are also a number of significant differences, the most notable being our emphasis on multiplayer compatibility. We will also examine other games, such as Day-Z or Rust that put the player in a position where they must manage the resources of a landscape to survive. Don’t Starve is another game that puts players in a foreign environment and makes them fend for their own survival. This game is not played from a first person or third person perspective, but rather from an isometric type 2D perspective. Although we have no intention of applying the same visual style to our game, there are some elements of *Don’t Starve* that we find appealing. For example, seeing as we intend to make our game as realistic as possible, we may choose to use the same approach as Don’t Stave uses in the way that once players are placed in the game world, they are not given any instructions on what to do.



Rust (2013) The Forest (2014)

# Methodology

For the development of this project we plan to operate using a scrum methodology. Scrum acts as an alternative to the traditional waterfall method of development. In a scrum framework the development is incremental and iterative, meaning that in each cycle or sprint a feature is implemented, tested and approved. Traditionally in a scrum environment you may have teams consisting of roughly 7 people, but in our case as there is only 2, so some modifications need to be made. Firstly there is no one defined product owner but instead we both contribute to those responsibilities. Of course we then also need to act as the development team where we must show a range of abilities from development and programming to testing and documentation (James, 2014). We have also incorporated daily scrum meetings into our development process where we cover what has been done since the last meeting what’s to do and any problems or blockers we have. This helps the process move more smoothly and gives us a clear understanding of what each other is doing. Currently we have defined 31 user stories, all with associated priorities, and we also have 5 categories defined for the progress of each one.

* Backlog: When a user story is in the backlog it means that no one is currently working on it and it is on the pile to be picked up.
* In Progress: The state is the in progress state, this means that someone has committed to delivering the user story and it is in the process of being developed
* Peer Review: This is the testing and review stage. Once a user story has been developed it is then tested by both members, once it is deemed successful then the other peer will review the changes to the code base or project etc. to both offer insight and improvements while also ensuring both parties are completely aware and comfortable with the contents of the project.
* Done: Once a user story has passed peer review we then tag it “done” meaning it should not need to be addressed again.
* Blocked: In the event that a user story cannot proceed due to something outside of the an individual’s control we then list it as blocked, meaning it will need to be re-addressed at a later date.

Here is a diagram to show the process:

 (James, 2014)

# Technical

## Target Hardware

The target hardware for this project is a PC or Mac running with Oculus Rift hardware. The Oculus Rift itself is also a target hardware. In the next section, we will discuss the Oculus Rift development kit in more detail.

## Development hardware and software

The main hardware we will be using throughout development will be our own personal computers. As our game is played via the Oculus Rift, we will also be using Oculus Rift development kits. There are currently two Oculus Rift development kits, the DK1 and the DK2, available for purchase. We had initially planned to do our development with the DK1 as one of us had already bought a DK1 last year, and the other had been given permission from the college to use one of the DK1 kits purchased by the college. However after speaking with our project supervisor about the superiority of the DK2 kit, we had a request approved by the college to purchase two DK2 kits to use for our development. One of the main benefits of getting to use the DK2 is that the resolution our game will be able to run at will be greatly improved. The supported resolution for the DK1 is 1280 x 720 whereas the DK2 supports a greatly improved 1920 x 1080. Included below is a table that highlights some of the key differences between the DK1 and DK2.

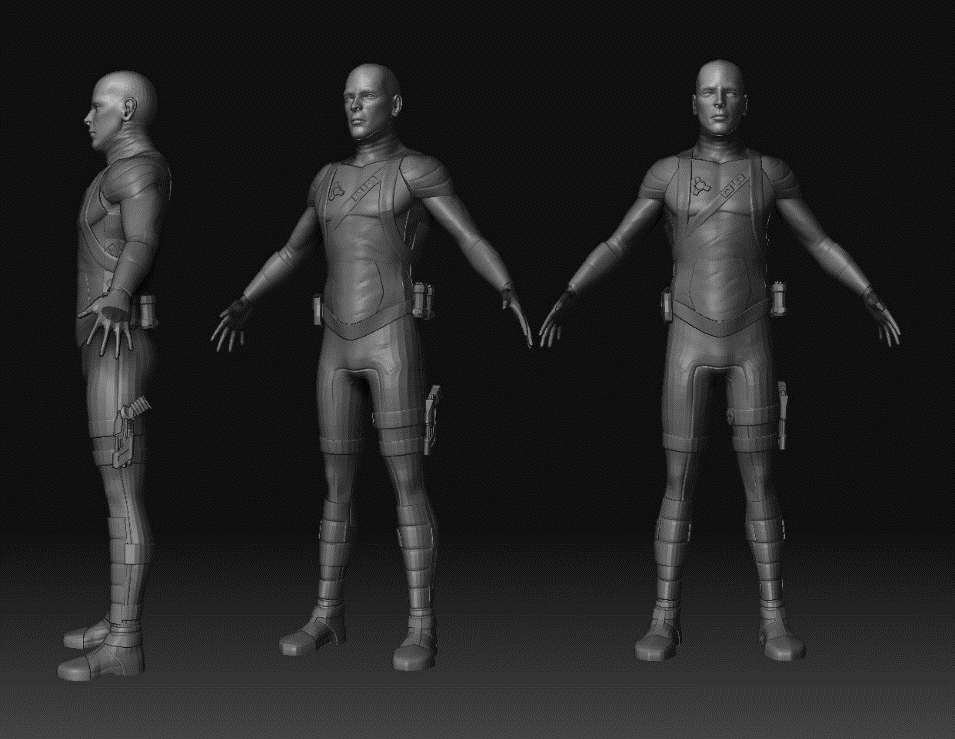


(Popa, 2014)

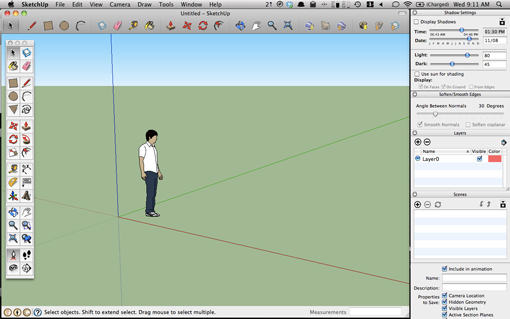
Additionally we will be making use of photo editing such as Photoshop to design icons, textures, menus and other such graphical features for our project. We justified the decision to use Photoshop as we have both gained experience working with the software during our graphical design module in 2nd year.

As our game will also be incorporating a multiplayer feature we needed to decide upon a networking solution. We agreed that Photon Unity Networking (PUN) was the best option for this. We have chosen to use this product for a number of reasons. We found it to be the most highly rated multiplayer tool outside of Unity’s built in service. It comes highly recommended based on feedback from users on the Unity forums. Our decision to use PUN will be further discussed in detail in section *5.5 Network*.

Our game world will contain a large number of different models once it’s complete. We intend to use a number of asset packs and open source models to populate the game world, however at this early stage it can be difficult to identify the models we will need to create ourselves and the models which we will be able to use from asset packs and open source sites. However when we do need to create models, we have agreed that we will use Maya to do so. We both have some experience using Maya, and have become familiar with the interface through our 3D modeling module and feel we will be able to create models to match the standard of our game.



The models that we choose to include in our game are extremely important. There needs to be a consistent visual style throughout the game, ranging from tools to large environments, and so it’s essential that when we add assets to our game, that they are consistent with each other. Google sketch up has an extensive library of models that can also be imported into Unity. There is a far greater variety of models available in the Google SketchUp 3D warehouse than there is in the Unity asset store, and so we have agreed that it may also be necessary to use Google SketchUp for modelling purposes.

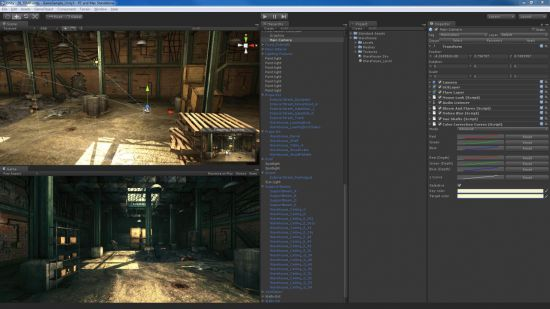
 (Google SketchUp)

## Development procedure and standards

For the development of this project we will be following an agile procedure, which means it will be an iterative development process providing us both with development and testing responsibilities. We decided upon this approach as while on placement both of us were fortunate enough to gain experience working in environments that adopted an agile approach to development. It also seems that an agile approach is the clear option when it comes to delegating and splitting up a project of this magnitude. The agile approach is explained in further detail in the *Methodology* section above.

## Game Engine

The engine that our project will be developed in is the Unity 3D game engine. This was an easy decision for us to come to for a number of reasons. Firstly, Unity is a free development platform, which means that there is no initial purchase cost, nor is there any licensing overhead. Other game engines such as the Unreal game engine, are priced at $19 per month, as well as incurring a 5% royalty fee on any sales that come from games created using the Unreal engine. We both have experience using Unity, as we have worked on a number of Unity projects throughout the course of our degree. This allows us to spend more time focusing on other elements of the game, such as game mechanics, and Oculus Rift integration, instead of having to become familiar with a new game engine interface. One of the most important factors in our decision to use Unity was that Unity supports both the Oculus Rift and the Photon Unity Networking solution which are integral parts to our project. Unity has an asset store feature which is essentially a collection of thousands of models that can be imported directly into a Unity project. This feature will prove useful for when we need to add models into our game. Unity also supports the importing of models from a number of other modelling software packages such as Maya, Google SketchUp, and Blender. We felt that its compatibility with such a vast array of different modelling tools was an added bonus to the game engine.



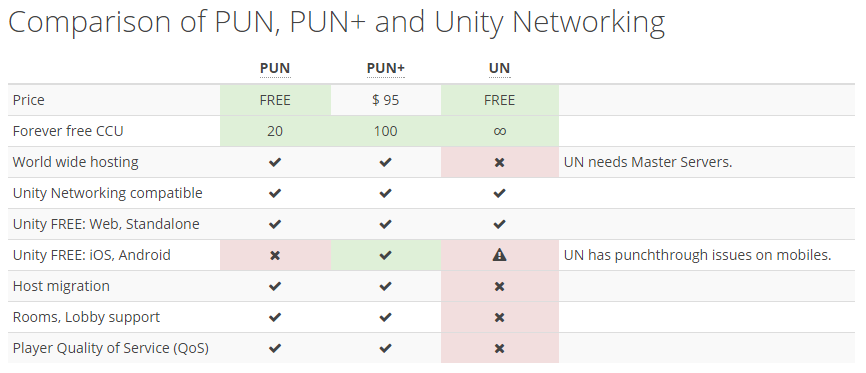
## Network

One of the most important features of our game was multiplaye. In order to add a multiplayer feature to our game, we had to decide on which networking solution to use. Unity comes with its own built in Networking solution (Unity Networking), however we were unsure if this solution would be idea for our project. After examining a number of other networking solutions, we agreed that the Photon Unity Networking (PUN) solution was the best approach to take.

Although Unity networking is often perceived to be Peer to Peer, it actually implements a server-client architecture. One player acts as a server, and all other clients then connect to that player (i.e the server). This approach of having one player acting as the server can cause a number of issues. Connectivity can be affected, as connections are often dropped as a result of the server acting player leaving the game. There can also be initial connectivity difficulties as a result of existing firewalls. Unity networking often has issues regarding NAT punch-through. It can often cause unintended functionality. The PUN solution does not encounter this issue as it has a dedicated server and so there is no need for NAT punch-through. As one player acts as the server, latency can become an issue as players are relying on the connection of the player acting as the server (Exit Games, 2014). PUN also uses a server-client architecture, however players do not have to act as a server, as there is a dedicated Photon Server in place. By having a dedicated server in place, we don’t have to worry about a game only running at the speed of the player acting as the server. We also don’t have to worry about the game failing if a connection is dropped as players are not connected to each other, but rather to the dedicated server.

Unity does support an alternative to their standard server-client approach. Developers can also avail of the Unity Master Server. The master server is essentially a dedicated server which can be used to host games. The master server will run an instance of a game, and clients will then connect to that master server. However the Unity Master Server is renowned as being extremely unreliable and a connection to the master server at any given time cannot be guaranteed. As a result, the master server is usually only used for testing purposes, however we were aware that it was a feature of Unity’s networking solution.

Regarding price, both Unity’s Networking solution and the Photon Unity Networking solution are free, and so price was never really an influential factor when deciding on a solution. There is a Pro version of PUN (PUN+) that offers more features than the free version, however these features were not relevant to our project and so we agreed that the free version was more suitable for us.



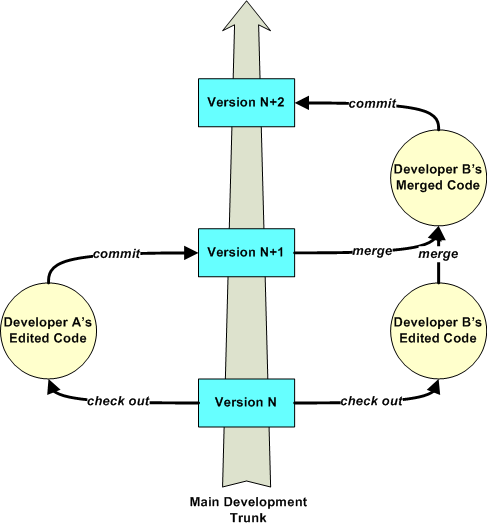
(Exit Games, 2014)

## Languages

For the development of this game we will be using C#. Unity supports both JavaScript and C#, however the version of JavaScript supported by Unity is not the conventional JavaScript but instead a version modified for Unity often referred to as Unity Script. We chose to use C# as we felt it would be more beneficial to gain experience using a language that would not only be applicable to Unity based projects, but also to other coding projects outside of Unity. Any experience gained using C# in Unity can be applied to other projects however had we chosen to code this project using UnityScript, we would only be able to apply that knowledge to other Unity projects. Based on feedback from the Unity community, C# seems to be the more popular of the two supported languages too. There is a lot more documentation and support for the Oculus Rift in C# than there is in UnityScript. This was also a factor in our decision.

## Version Control

The version control mechanism we chose to use for this project is Subversion or SVN. We came to this decision as our research showed that it is much easier to integrate SVN into a Unity project than it is with GIT. This is because SVN can seamlessly handle prefab and object files created in Unity as opposed to GIT which had some trouble in this area. To test this for ourselves we prototyped with SVN and GIT and found SVN to be the much smoother process.

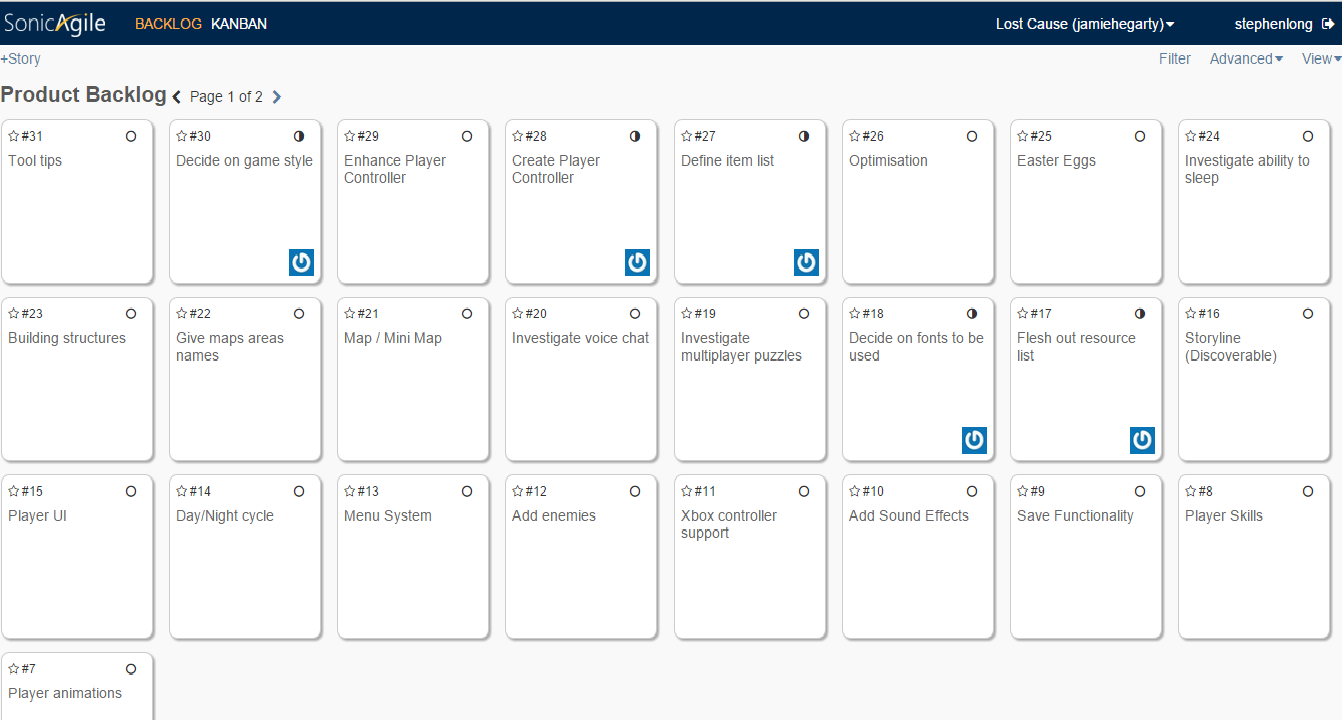


(Rice University, 2010)

## User story management

For the user story management aspect of our project we decided to use an online platform called Sonic Agile. After sampling a handful of different options, this platform proved to be most enjoyable experience, with not only a nice user interface but also an array of features allowing us to manage our user story states, add comments and tags as well as a host of other smaller features. Compared to other products on the market Sonic Agile is also free for development teams of two people so this also factored in our decision.

Listed below is a backlog of all user stories we have created so far for our project. Sonic Agile makes it easy to assign user stories to different people, and so work remains easily distinguishable.



# Iteration 1

## Defining Iteration 1 objectives

We had a number of main goals that we wanted to achieve upon completion of iteration one. Listed below are the goals we set. These can also be seen in the *7.0 Management* section.

* Adding multiplayer functionality
* Add Oculus Rift support
* Create our Character Controller (i.e our Player Prefab)
* Decide on a game style

## Iteration 1 development

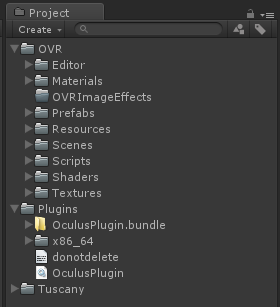
**Deciding on a game style**

Iteration one was a learning experience for both of us. After researching different visual styles, we agreed upon a style that we felt would suit our game. This can be seen in the screenshot taken from the prototype we created (in the *Visual Style* section).

**Adding Oculus Rift Support**

For the first iteration of development we aimed to get to a basic level of Oculus Rift functionality. For this we wanted to be able to have a player start the game with their Rift connected, and then move and look around as normal. To get this functionality in game, the Oculus Developer portal supplies a downloadable SDK and Integration kit.

Once we had both downloaded and installed the SDK and other required drivers, we were able to import the Oculus Rift Integration kit into Unity. This provided us with a number of resources ranging from prefabs to scripts and textures (see below).



But for this early iteration we were only concerned with the basic “OVRPlayerController”. Once we added the player controller to our game scene, we could now start the game using our Oculus Rift, with player movement and head tracking functionality.

Aside from adding the first stage of functionality to our game, this user story also contained tasks of research and investigation with regard to further development and practices. The Oculus Rift developer portal also provides documentation on correct development practices for the Rift which we needed to cover. Inside this documentation it highlights the difficult in creating a user interface for a player wearing the Rift, as items such as health bars and other information cannot be drawn on the screen in a conventional manor, instead need to be presented as an object in 3D space in front of the player.

So we identified this as an early obstacle and began putting time into researching the best way to accommodate for this. Eventually after considering and prototyping a variety of options including the Rifts own GuiHelper class we decided that the best tool for the job would be Next Gen UI. This is an asset available on the Unity Store designed to help in the process of creating user interfaces in Unity (which can be found here: <https://www.assetstore.unity3d.com/en/#!/content/2413>). We decided on this tool based on a number of reasons. Firstly I asked Oculus and Unity developer communities about this topic and the majority recommended NGUI as the right choice, additionally it is a very popular asset with good support and finally the process by which is renders the GUI, using a render plane positioned in front of the player camera allows us overcome the obstacle mentioned previous with regard drawing elements on the screen.

We have gotten to a stage where we have a prototype project set up with NGUI but have yet to integrate it into our main project with the Oculus Rift.

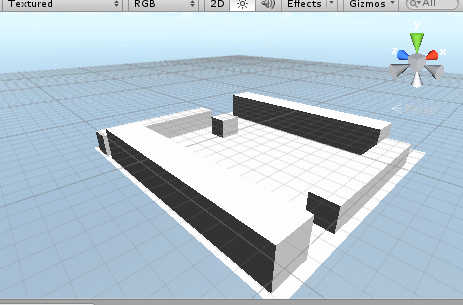
**Adding multiplayer functionality**

In order to add functionality whereby players could connect to a server, we needed to become familiar with PUN. This would involve learning how to connect to the server, and how to send information about objects to all clients connected to that server.

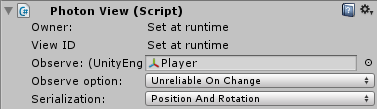
Creating the server was a relatively painless experience during which no issues were encountered. Firstly we registered with PUN in order to avail of their networking solution. We then received an Application ID which is essentially an ID that allows us to connect to one of the photon servers. This Photon Server hosts our game world, and the clients then connect to the server. All PUN related settings can be managed via the PUN wizard which is pictured below.



We then created a simple game world consisting of a plan and some cubes. This would act as a test environment where players would connect and move around.



Once the server was set up, and PUN had been included in the project, the next step was to create a script that allowed clients to actually connect to the server. When the game begins, the client connects to the server using the settings that have been set in the PUN settings section of the PUN wizard. Once a client connects to a room (which is an instance of our game world running on the server), an instance of a player is then created. The client can then control than player and navigate the game world. To send information about an object to other players, a photon view must be attached to the object. A photon view is a component that “observes” changes related to that object. For example, the image below shows that we attached a photon view to the Player prefab. The “observe” value is set to the players transform (position and rotation). This means that every player’s position and rotation information will be sent to the server, which will in turn distribute that information to all other connected clients. So if Client A moves forward, this information will be “observed” and sent to the sever, which will then send that information to Client B. Client B will then observe Client A’s movement.

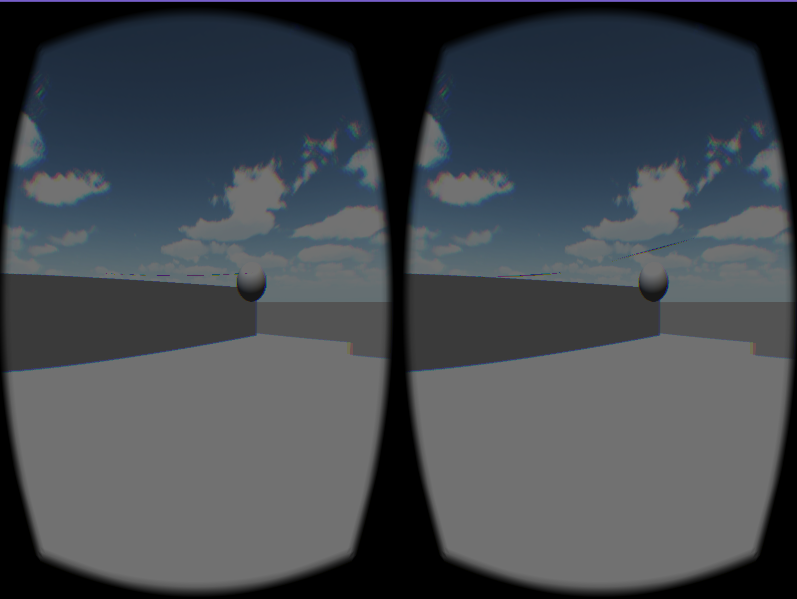


We encountered a number of issues when trying to get a functioning character controller working once an instance of it was created. Initially the “player” object that was spawned once a client joined the server, was an instance of the OVRPlayerController, which is the standard Player Controller prefab that comes with the Oculus Rift SDK. However we found that once more than one client connected to the server, there were a number of issues that resulted in head orientation and player movement being controlled by incorrect clients. For example, when Client A would attempt to move forward, this would actually cause Client B to move.

In order to debug the issue, we decided that instead of trying to integrate Oculus Rift functionality straight away, we would instead attempt to successfully instantiate a player that uses Unity’s standard FirstPersonController prefab. We were able to successfully create instances of the FirstPersonController prefab, and so we then attempted the same feat using a ThirdPersonController prefab. Again, this worked with minimal issues. After some debugging we were able to conclude that seeing as we were able to create instances of players using both the FirstPersonController and the ThirdPersonController, the issue we were experiencing must have been originating from a script that comes already attached to the OVRPlayerController. To combat this issue, we created a custom Player prefab.

**Creating the player prefab**

We created our own custom Player prefab by adding the OVRCameraController and some First Person Controller related scripts to an empty game object. This gave us a Player prefab that moved like Unity’s standard FirstPersonController prefab, but also supported the Oculus Rift. We intentionally deactivated a number of scripts on the Player prefab, and then only activate them once the player has been spawned. This stops clients from controlling each other as a client only gets control of a player instance once it has been spawned. The image below shows the players view when they connect. The sphere in the image represents another players head. In the image below, two clients have connected, and can move around independent of each other.



The player prefab we currently have will develop a lot more throughout the course of the project. As we add more functionaltity.

## Iteration 1 learning experiences

We certainly learned a lot about our project from the first iteration. We learned a lot about time management as some of our objectives took longer to achieve than others. For example, enabling clients to connect to the server without any issues proved to take longer than we had anticipated. However regardless of this, we now have a much better understanding of PUN, photon components, the setup of our player prefab, and we also learned a lot about what is involved in developing for the oculus rift and the additional steps and challenges that are required for even simple tasks. A major learning point as mentioned above was to get graphical elements appearing on screen. We prototyped and learned how to use the built in GuiHelper class with the Rift, but it is very limited in its functionality and all we could produce from it was a simple text output on screen. Although the Oculus Rift SDK comes with documentation, the documentation is not very extensive and we have identified that a lot of the learning is going to come through trial and error, and there will be extensive prototyping along the way.

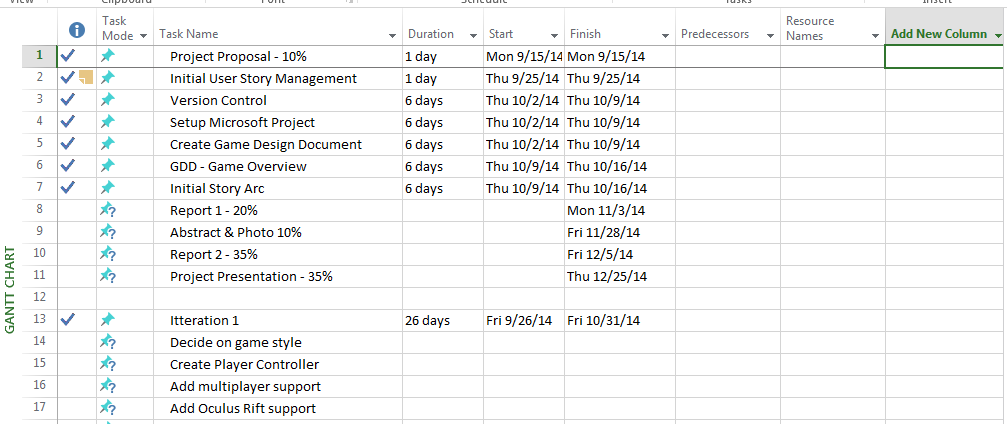
# Management

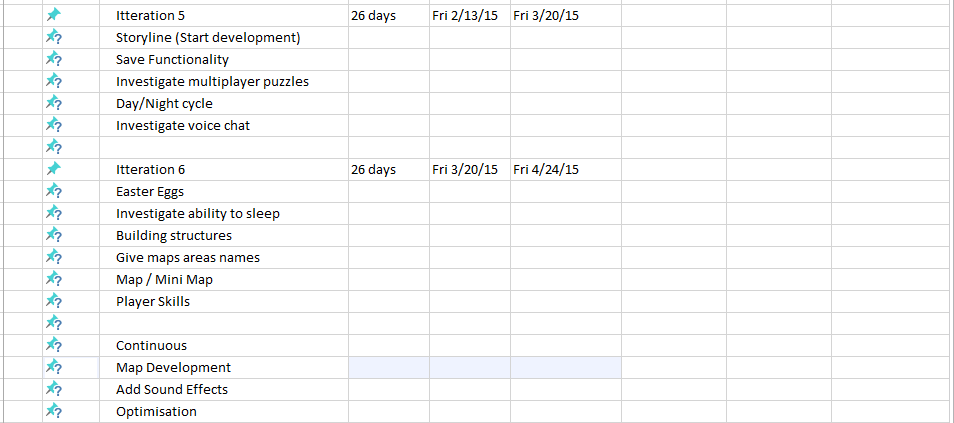
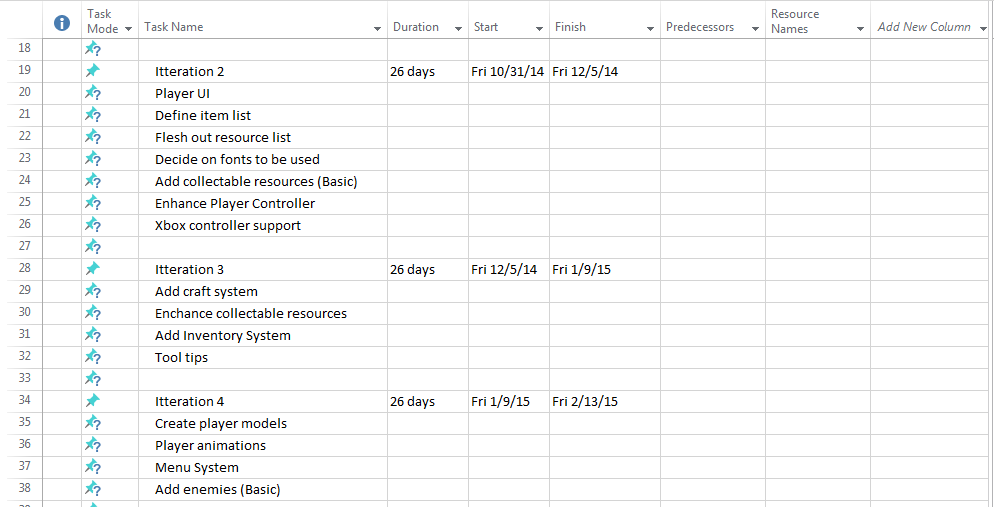
## Detailed Schedule

For managing the project deadlines and our own personal milestones we have decided to use Microsoft Project, a project management tool that lets us create tasks with start and finish dates, mark the completion of these task as well as other useful features.

So far we have broken down the development process of our project into 6, 5 week iterations bringing us from the start of the year to the end. These have been ordered according to what we predict the scale of the work involved may be as well as the order in which we feel things need to be completed. It is worth noting that over the course of development some of these may be moved around or changed as of right now it is quite difficult to give accurate time estimates.

Here you can see our project deadlines listed as well as our 6 iterations of user stories.





Additionally to Microsoft project, we will also be using a product called Sonic Agile to manage our user stories (covered in section 5.8).

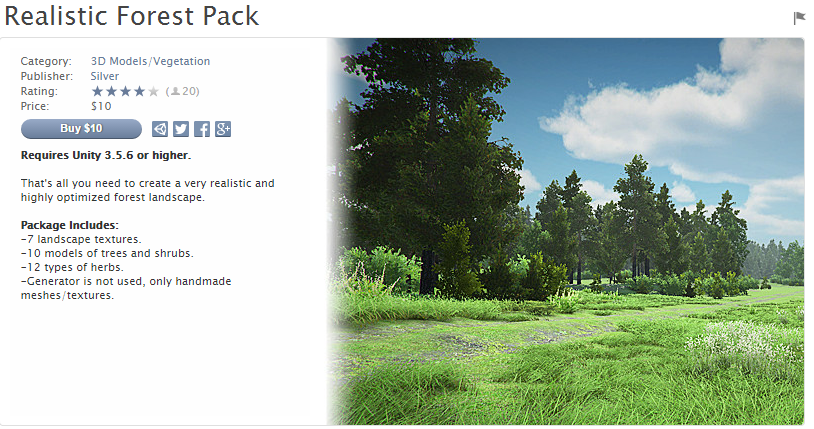
## Budget

After assessing the different technologies available for us to use throughout development, we determined that it would be possible to develop our project using only free technologies, or technologies that we already had access to.

However, as mentioned in an earlier section, after speaking with Brendan Lyng, and discussing how the feature set of the DK2 is a lot more impressive, and as a result would contribute to a higher quality project, Brendan put forward a successful request for the college to purchase two DK2 kits for us to use. Seeing as the DK2 kits cost $350 dollars each which brings the total cost of our project to $700 dollars, or rather €555. As a result of this, our current project now has the following financial breakdown:

|  |  |
| --- | --- |
| **Hardware/Software** | **Cost** |
| Unity | Free, as we both have the PRO version |
| Oculus Rift DK1 | Free |
| Oculus Rift DK2 | €555 |
| Maya | Free |
| Photoshop | Free |
| Sonic Agile | Free |
| SVN | Free |
| Photon Unity Networking (PUN) | Free |
| Google SketchUp | Free |
| \*Game Assets | €50 |

\*We are aware that we will need to populate our game world. Although we intend to do this mostly through the use of open source software and free to use asset packs, we are aware that in some cases the assets we desire may not be available for free, and so we are willing to spend some money on assets if we deem them necessary for the game. Take for example, the asset pack below. If we felt that such as pack would add to our game, we would consider making the purchase. The amount assigned to the game assets in the table above is just an approximation and is subject to change.



## User Stories

Currently we have defined 31 user stories, this list is however growing all the time as we research our project more and identify additional items that need to be covered. Here is our current list:

* Add in game tool tips
* Decide on game style
* Enhance Player Controller
* Create Player Controller
* Define item list
* Optimization
* Investigate Easter Eggs
* Investigate ability to sleep
* Building structures
* Give maps areas names
* Map / Mini Map
* Investigate voice chat
* Investigate multiplayer puzzles
* Decide on fonts to be used
* Flesh out resource list
* Storyline (Discoverable)
* Player UI
* Day/Night cycle
* Menu System
* Add enemies
* Xbox controller support
* Add Sound Effects

## Risk

The project itself could be summarized as one large risk. We genuinely believe that we have set ambitious goals for this project. Most games that have a dedicated staff, take a lot longer than nine months to complete. However we have still set ourselves the target of delivering a complete, well-rounded product. Also, the fact that we have decided to undertake a collaborative project is a risk in itself as we will undoubtedly be held to a higher standard than those who are doing individual projects.

### Skills Risk

There are a number of different technologies used in our project. Some of these technologies we have used before, and some we have not. As a result, there are going to be cases where we are learning as we develop. As a result, the development of this project will test our skills in a number of ways.

**Management Risk**

The project is due to be completed over the course of nine months. Creating an initial timeline for iterations and user stories for a project before any development work is done on that project, is an extremely difficult task, and one that is an immense test of managerial and planning ability. We are aware that failure to set realistic iterative goals, and to successfully divide user stories amongst both team members, could end up being detrimental to the success of our project.

**Adding Networking Capability**

Perhaps one of the biggest risks taken in this project, was our decision to include multiplayer functionality. Last semester, during our *Networking and Multiplayer Games* semester, not a single person was able to successfully instantiate a server and get two clients connected without any bugs. Deciding to add multiplayer functionality to a project of a much larger scope, was a huge risk. However we are confident in our ability, and we understood that if given time, we could get it to work.

**Oculus Rift SDK Risk**

Another large risk we took was to develop the game for the Oculus Rift. As mentioned in the *What already exists* section of this report, the rift is a new technology. As a result, a lot of developers are still themselves trying to become familiar with the SDK, and are still themselves learning how it works.

**Modeling ability**

As was mentioned earlier in the report, there will almost certainly be occasions where we need to create our own assets for the game. However, creating game assets can be a lengthy process, and so we will need to learn to create models that meet the standard of our game world, as well as being time efficient.

**Coding ability**

The coding for this project will be done in C#. Although we do have experience using C#, it isn’t as extensive as our experience with some other coding languages. As we progress through development, we are aware that our knowledge of C# may be tested, and that we will constantly be learning as we progress.

### Team risk

We are aware that in the case that there may be conflict throughout the course of development, that we must still work to complete the project to the best of our ability. We have agreed that under any circumstances where a conflict arises, user stories will still be assigned, and peer review will still have to take place for the good of the project. Although it’s extremely minimal, we are aware that there is some risk involved in undertaking a team project. Conflict may even arise due to social reasons outside of the project, however, as mentioned above, we are very aware that if conflict does arise, we must continue to work as part of a team regardless.

In relation to the assigning of user stories to team members, we understand that there is a risk that one person may be assigned more work than the other, or work that they feel is too difficult to undertake alone. In order to successfully complete this project, we will have to work well as part of a team. There will be cases where one person’s progress may be dependent on the other person’s completion of a task. There is always the risk that one person may struggle with the workload they have been assigned for a given iteration.

The majority of Final Year Projects that will be undertaken will be individual projects. We acknowledge that by being one of the few collaborative projects, we may be held to a higher standard due to the fact that we have two people working on the project. We are aware that we may be expected to make further advances for each iteration of our project, than individuals that are undertaking solo projects.

### Technological Risks

Before we began the project, we did a lot of research regarding the technologies we would use. This ranged from the game engine we would use, to the modeling software we would use. We wanted to minimalize any technological risks before we began development. We are very confident that all the technologies we have chosen to use will work in conjunction with each other. As of now, due to the completion of iteration one, we have already confirmed that the two main technologies we will use for our game, PUN, and the Oculus Rift, are able to work in harmony with our project. However, regardless of this, we do understand that there is always the risk of an initial struggle with all these technologies. For example, even though we have both used Unity before, Unity is a game engine that has endless libraries, and so there will be times where we struggle with functionality, or misunderstand how to implement some functions. This is the same for all technologies we will use. It will take time to become familiar with all the technologies that we will use for our development. We are however, confident that all technologies that need to be compatible for this project, are compatible.

# References

Exit Games, 2014. *Photon Unity Networking Intro.* [Online]   
Available at: http://doc.exitgames.com/en/pun/current/getting-started/pun-intro  
[Accessed 25 October 2014].

James, M., 2014. *Scrum Reference Card.* [Online]   
Available at: http://scrumreferencecard.com/scrum-reference-card/  
[Accessed October 2014].

Rice University, 2010. *Subversion Source Control General Information.* [Online]   
Available at: https://www.clear.rice.edu/comp310/Eclipse/Subclipse/subversion.html  
[Accessed 25 October 2014].

Exit Games, 2014. *Photon Unity Networking (PUN) Compared To Unity Networking (UN).* [Online]*.*Available: [http://doc.exitgames.com/en/pun/current/reference/pun-and-un/](http://doc.exitgames.com/en/pun/current/reference/pun-and-un/%20) [Accessed 10 Oct 2014].

Justin Popa., 2014. *Oculus Rift: DK1 vs DK2.* [Online]  
Available at: [http://in2gpu.com/2014/08/10/oculus-rift-dk1-vs-dk2/](http://in2gpu.com/2014/08/10/oculus-rift-dk1-vs-dk2/%20)   
[Accessed 15th Oct 2014].