|  |
| --- |
| Technical Design Document |
| UltraGolf |
| Richard Delamore |

|  |
| --- |
|  |

Contents

[1.0 Revision History Version 2](#_Toc15394261)

[2.0 Development Environment 2](#_Toc15394262)

[2.1 Game Engine 2](#_Toc15394263)

[2.2 IDE 2](#_Toc15394264)

[2.3 Source Control procedures 2](#_Toc15394265)

[2.4 Third Party Libraries 2](#_Toc15394266)

[2.5 Other Software 2](#_Toc15394267)

[3.0 Game Overview 3](#_Toc15394268)

[4.0 Graphics 4](#_Toc15394269)

[5.0 Artificial Intelligence 5](#_Toc15394270)

[6.0 Physics 8](#_Toc15394271)

[7.0 Items Item 8](#_Toc15394272)

[8.0 Game Flow 8](#_Toc15394273)

[9.0 Levels 8](#_Toc15394274)

[10.0 Interface 9](#_Toc15394275)

[10.1 Menu 9](#_Toc15394276)

[10.2 Camera 9](#_Toc15394277)

[10.3 Controls 9](#_Toc15394278)

[11.0 Asset List 9](#_Toc15394279)

[12.0 Technical Risks 9](#_Toc15394280)

# Revision History Version

|  |  |
| --- | --- |
| Version | Description |
| 1.0 | Initial Document |
| 2.0 | Final Update |

# Development Environment

## Game Engine

* Unreal (Version: 4.20.3)

## IDE

* Visual Studio 2017

## Source Control procedures

* GitKraken

## Third Party Libraries

* None

## Other Software

* Photoshop (texturing, particle and interface design)
* Maya (3d modelling and animation)

# Game Overview

Players will navigate 3D platforms and obstacles in a mini golf style environment with a bouncy dodecahedron (UltraBall) to reach to goal while keeping under par.

The par system will be used to show players how many attempts at hitting the UltraBall they have used. Each time UltraBall is launched, the par counter goes up by one. There is no way to reduce the par after it goes up apart from restarting the level.

When the player hits the Finish Zone (represented by a giant Dodecahedron) the level ends. The player will be able to either move on (if under par) or start the level again.

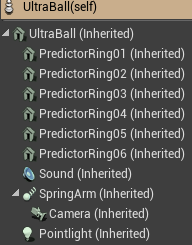
**Minimum System Requirements**

UltraGolf will be a PC game. If time permits I may try to port it to Android.

* Windows Operating System
* Intel i7 CPU
* 16 GB of RAM

**List of Game Objects**

Below is a list of all the main objects that exist independently of level. The placement of these objects within each level will be up to the designer.

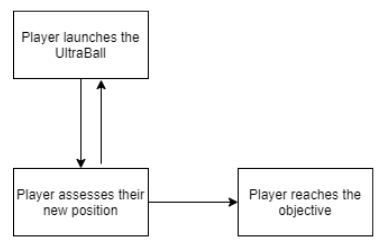
**UltraBall (Dodecahedron):** The UltraBall is the object possessed by the player’s controller. It consists of a static mesh and camera on a spring arm.

**Bumper:** The bumper is an object that propels the UltraBall in a specific direction when the UltraBall collides with the front of it. This allows the player to be quickly pushed in a very specific direction.

**Finish Zone:** The finish zone is a giant Dodecahedron surrounded by a circular particle system, when the player collides with it, the par is calculated. If the player is over par, a UI pops up with the option to restart the level or exit the application. If the player collides with the par area and is below the par, they get a UI that congratulates them, displays the par they got in comparison to the limit, and can press next level to proceed.

**Game Flow**

The basic game loop is very simple. The Par counter will go up once every time the UltraBall is Launched. The player can proceed to the next level if they collide with the finish cylinder and are below the required par. If the player does reach the objective above the maximum par, the level will reset after a failure UI pops up.

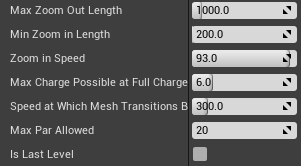
****

The first level will be a tutorial level to help the player get acquainted with the controls. Each time the player completes a level they will be teleported to the next level. Each new level gets progressively more complex than the last. We will develop a minimum of eight levels with possibilities for more. A limited number of assets will be made and rearranged to create more and more complex environments.

# Mechanics

**UltraBall (Dodecahedron):** The main mechanics of the game are all related to the UltraBall and how it interacts with its environment. I have setup physics on the ball and have enabled the player to possess it and apply force through “launches”.

Additionally, I’ve configured the ball so that the designer can balance every aspect of its physics to fit their level design needs.

**Max Zoom Out Length:** The maximum distance the player can Zoom the camera out.

**Min Zoom In Length:** The minimum distance the player can Zoom the camera in.

**Zoom in Speed:** The speed at which the player can zoom in and out.

1. = 1% of Zoom Length per scroll.

100.0 = 100% of Zoom Length per scroll.

**Max Power Possible at Full Charge Up:** This is the maximum amount of launch force that can be applied to the ball when the player releases the Left Mouse button after fully charging the ball.

**Speed at Which Mesh Transitions Back to Complex:** When fired UltraBall will use a simple sphere collider to make gameplay predictable. However, when it drops below this speed (velocity) it will transition back to a complex dodecahedron collider to make gameplay look more real.

**Max Par Allowed:** This is the maximum par acceptable for this level.

**Is Last Level:** This indicates whether this is the last level for the game.

# Graphics

**Perspective**

When the level starts the camera flies over the map to give the player a general overview of the best path to their final zone. It will then instantly transition back to a camera floating behind the UltraBall which will follow UltraBall in a third person perspective.

**Art Style**

The game will use a cell shaded, hand-painted style graphics making extensive use of reflection mappings and emissive maps.

**Polycount**

The poly count will be kept as low as possible to enable quick rendering of graphics and map. Extensive reuse of assets will be applied.

# Physics

The game will use the Unreal Engine physics and collision system with some basic modifications applied to make game more enjoyable to play.

**Modifications to physics and collision handling**

It was decided that the UltraBall should roll like a sphere at fast speeds and like a dodecahedron at slow speeds. This would visually trick the player into thinking that they are shooting a multi-faced object without them having to deal with the unpredictable physics associated with a dodecahedron shaped object.

The below table clarifies when UltraBall will act like a sphere and when it will act like a dodecahedron.

|  |  |  |
| --- | --- | --- |
| **Event:** | | **Change To:** |
| Velocity of UltraBall drops below the designer specified **Speed at Which Mesh Transitions Back to Complex** | | Dodecahedron Collider |
| Player applies launch force to UltraBall by releasing Left-Button. | Launch force is less than 1.0 newton | Dodecahedron Collider |
| Launch force is more than 1.0 newton | Sphere Collider |

# Game Flow

**Level Structure**

Every level will consist of four basic elements: UltraBall, Ramps, Bumpers and Finish Zones. The level layouts will be configured by the Designer and the Artists. Each level will get progressively harder and harder to traverse.

The player will attempt navigate the UltraBall through the map to reach the End Zone in the shortest amount of hits possible. They will be presented with Ramps, Bumpers and Dead Zones which will assist them in reaching their objective.

**Player Objectives**

The objective of each level is to reach the Finish Zone within the required par.

**Evaluation of Player Progress**

Each time the player launches the UltraBall, the par counter will increment. If the player reaches the objective above the maximum par, the level will reset after a failure UI pops up.

**Saving of Game Progress**

At present, saving of game progress isn’t required.

# Interface

**Menu Options and Functions**

**Main Menu:**



**Controls Menu:**

Specifies the game controls for the player:



**Credits Menu:**

Contains a static list of credits: Artists, Designer and Programmer.

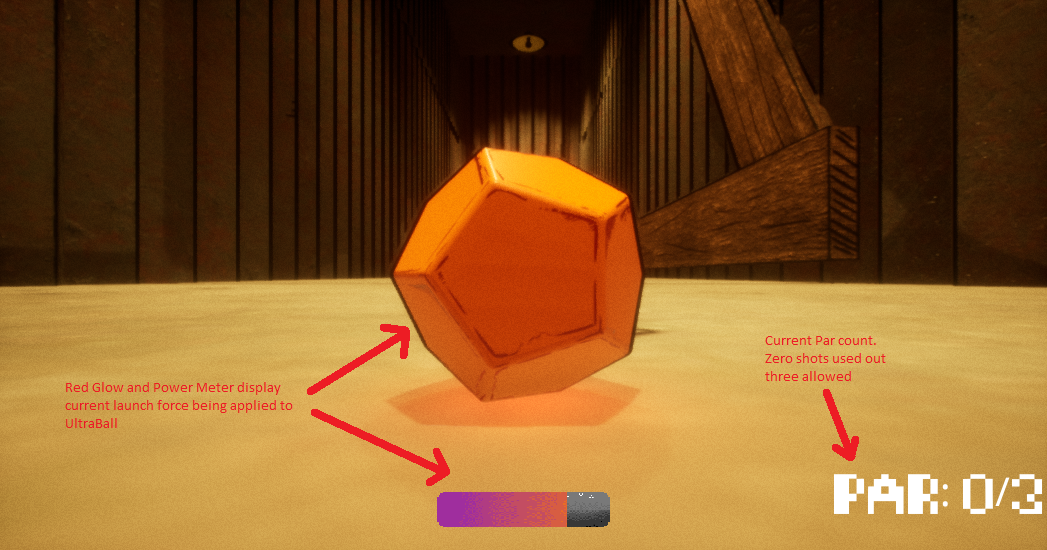


**In-Game:** The in-game interface will be partially diegetic.

Launch Force: If UltraBall isn’t moving and the player holds the Left Button, a charge bar will appear and UltraBall will glow red depending on how much charge is being applied. If the UltraBall is already moving an interface element stating “Nope” will appear.

Par: The current par will be displayed in the bottom right hand corner of the screen.

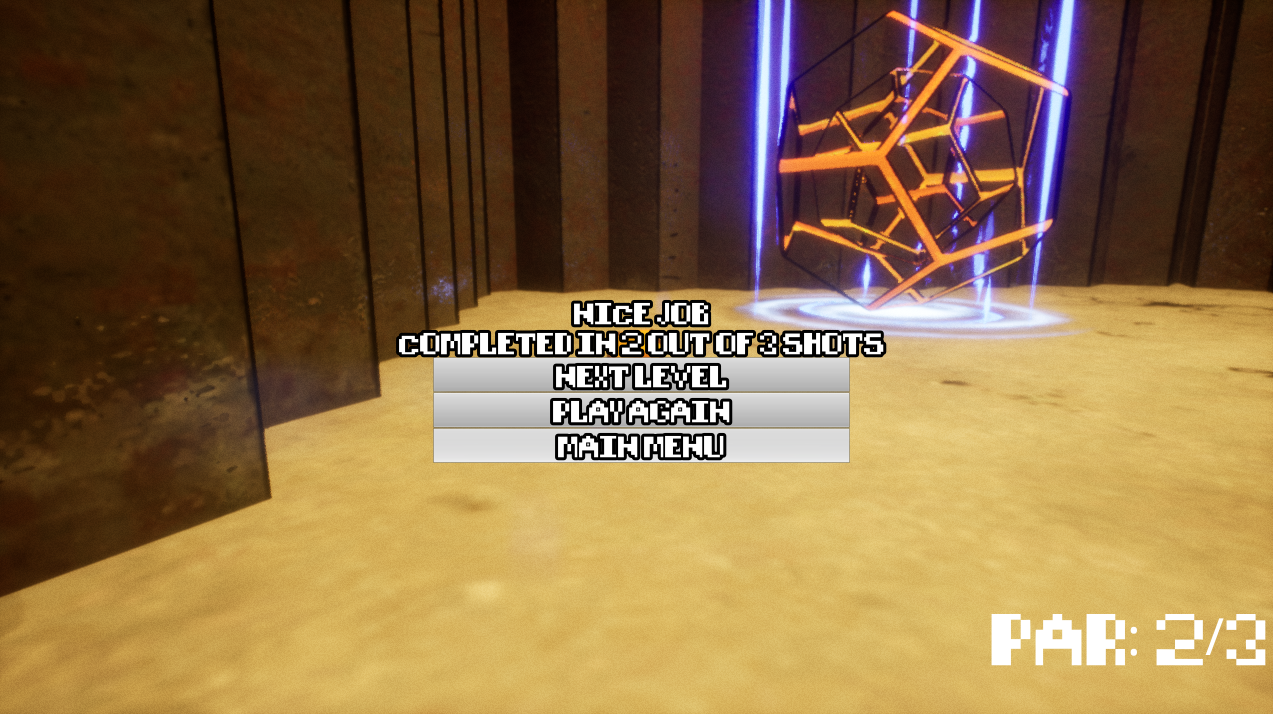
Escape: When the player presses escape, a Pause Menu will appear.



**Pause Menu:**



**Finish Zone:**

****

**Camera Operation and Controls**

**Camera Positioning**: The camera floats behind the UltraBall on a camera spring arm creating a third person perspective for the player. This ensures that the camera doesn’t collide with its surroundings and makes it easy for the player to aim the UltraBall.

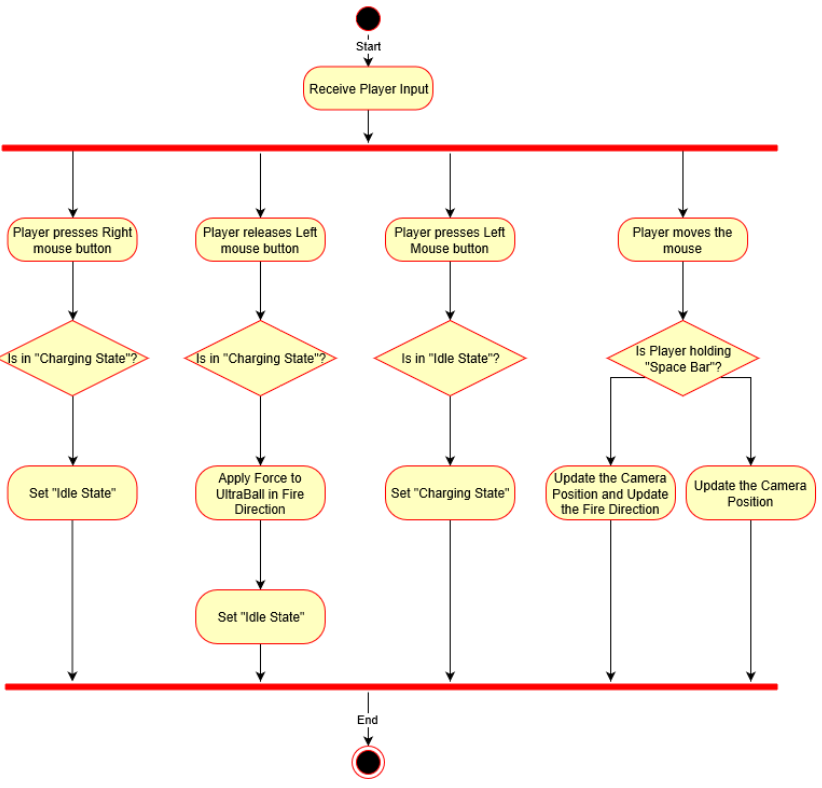
**Camera Zooming:** Players can zoom the camera in and out by using the **scroll wheel**.

**Launching:** Players **left click** and **hold** to launch the UltraBall. The longer they hold, the more power is put into the launch. The designer can specify the maximum power using designer facing variables. The player can only launch UltraBall when it is stationary otherwise they’ll receive an error message.

**Cancel Launch**: Players can press **right click** while they are **holding** **left click** to cancel the left click input, resulting in no action happening after they release left click.

**Activity Diagrams**

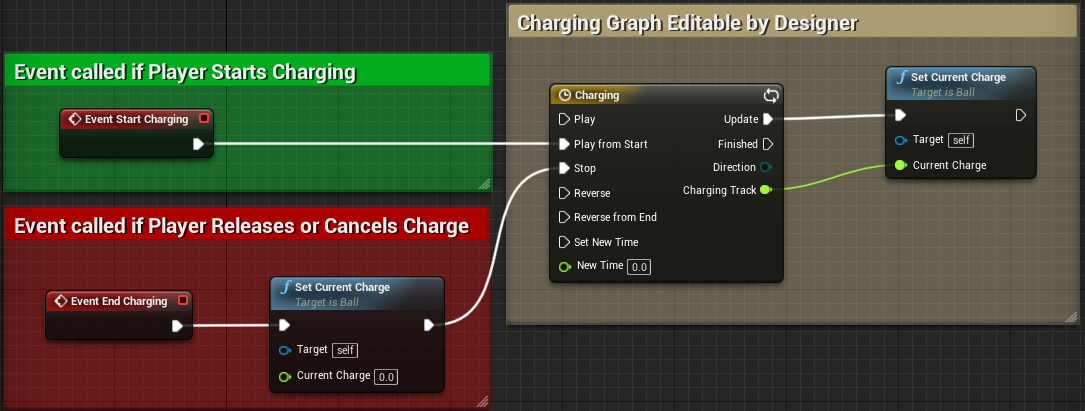
This activity diagram shows the different states UltraBall can exist in and how it enters and exits from them.



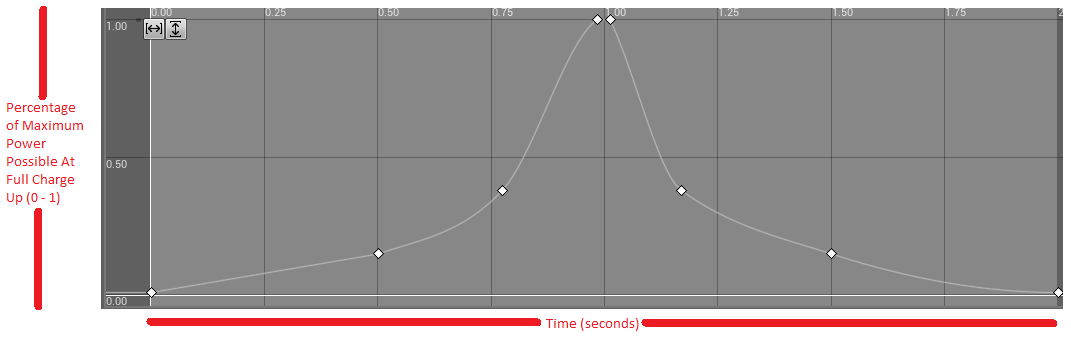
**Charging State:**

When UltraBall enters the Charging State, a graph function is activated which plays on a two second loop constantly until UltraBall exits the Charging State.

The designer can modify the power curve of the graph function through Blueprints. Everything else is managed through C++.



Below is a screenshot of the current power curve.

****

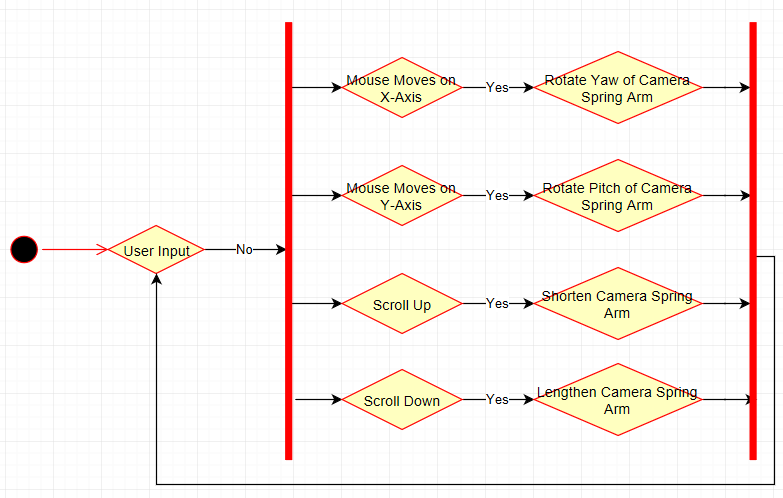
The power curve represents what percentage of **the Maximum Power Possible At Full Charge** would be applied if the Player released the Left Mouse button.

For example, if the Player released the Left Mouse button at the 1 second mark on the power curve, 100% of **the Maximum Power Possible At Full Charge** would be applied.

If the Player released the Left Mouse button at the 1.5 second mark on the power curve, 25% of **the Maximum Power Possible At Full Charge** would be applied.

**Camera Positioning and Zooming System**

As previously stated, the camera floats behind the UltraBall on a camera spring arm that handles camera collisions with the environment. It’s this spring arm that is modified by user input – not the camera directly.



Please note, that the Camera Spring arm length is restricted to the designer specified amounts listed in the Core Mechanics. Additionally, the Pitch of the Spring Arm is clamped between 36° and -70°. This ensures that the camera doesn’t accidently rotate over the UltraBall thus inverting the controls.

# Asset List

Unknown at this stage.

# Technical Risks

The biggest risk I foresee is implementing the visible stretchiness of the ball. This isn’t a required feature, but it will make the game more visually appealing. In essence, I want to enable the ball to squish upon impact with the ground and walls. My initial research says this is possible, but it will require an extensive study of the Unreal animation system and close work with the artists to create the desired models.

This feature is a nicety to make the game more visually appealing and isn’t crucial to the overall development, so it will be implemented after all the core mechanics have been programmed.