|  |
| --- |
| 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 |
| 1.5 | Pivot Point |
| 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 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 shows players how many attempts at hitting the UltraBall they have left. 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 can either move on (if under par) or start the level again.

**Minimum System Requirements**

UltraGolf is a dedicated PC game.

* 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 has been specified by the designer.

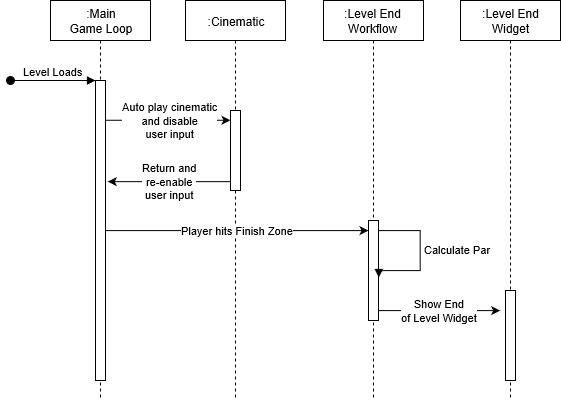
|  |  |
| --- | --- |
|  | **UltraBall (Dodecahedron):**  UltraBall is the object possessed by the player’s controller. It consists of a static mesh, camera on a spring arm and various other objects such as predictor rings used for path prediction. |
|  | **Bumper:**  The bumper is an object that propels the UltraBall in a specific direction when the UltraBall hits it. This allows the player to be quickly be pushed in a specific direction. |
|  | **Finish Zone:**  The finish zone is a giant Dodecahedron surrounded by a circular particle system. When the player collides with it, the level ends and 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 is below par, they get a UI that congratulates them, displaying the par they got in comparison to the limit. They are also presented with the option to proceed to the next level. |

**Game Flow**

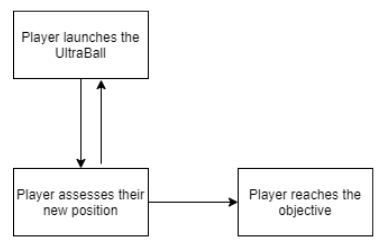
The first level is situated in a bare bone shed. It’s a basic tutorial level designed to help the player get acquainted with the controls. Each time the player completes the level it will get progressively harder as more and more obstacles are included. We have developed eight levels in total with possibilities for more. A limited number of assets have been made and rearranged to create the ever-increasing complexity.

**Level Flow**

The basic game loop for each level is very simple.



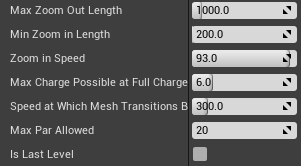
Firstly, the player controls are disabled while a cinematic camera flies through the level giving the player an overview of the best path to the Finish Zone. Then the player controls are re-enabled, and the viewport changes to the camera associated with the UltraBall.

****

The player then needs to navigate the level by applying launching the UltraBall. Each time the player launches the UltraBall the Par counter goes up.

When the player hits the Finish zone, their par is calculated. If they are above par, a failure UI is displayed. If they are below par, a success UI is displayed.

# 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 UltraBall 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 velocity it will transition back to a complex dodecahedron collider to make gameplay look realer. |
| **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 uses a cell shaded, hand-painted style graphics making extensive use of reflection mappings and emissive maps.

**Polycount**

The poly count has been kept as low as possible to enable quick rendering of graphics and loading of the levels. Extensive reuse of assets has occurred.

# Physics

The game uses the Unreal Engine physics and collision system with some basic modifications applied to make gameplay more enjoyable.

**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 acts like a sphere and when it acts 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 consists of four basic elements: UltraBall, Obstacles, Bumpers and Finish Zones. The level layouts have been configured by the Designer and the Artists. Each level gets progressively harder and harder to traverse.

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

**Player Objectives**

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

**Evaluation of Player Progress**

Each time the player launches the UltraBall, the par counter will increment. If the player reaches the Finish Zone 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:

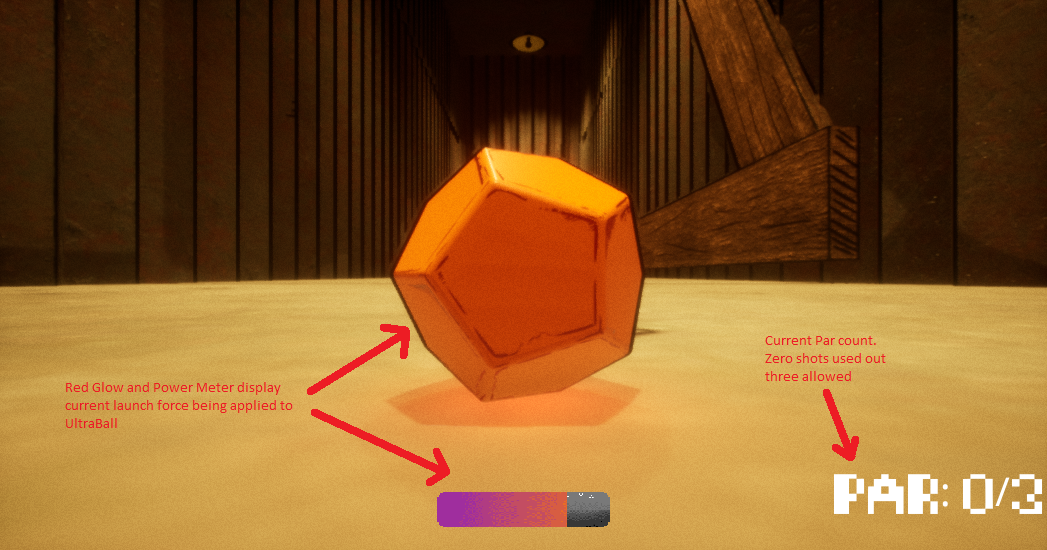


**Credits Menu:**

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



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

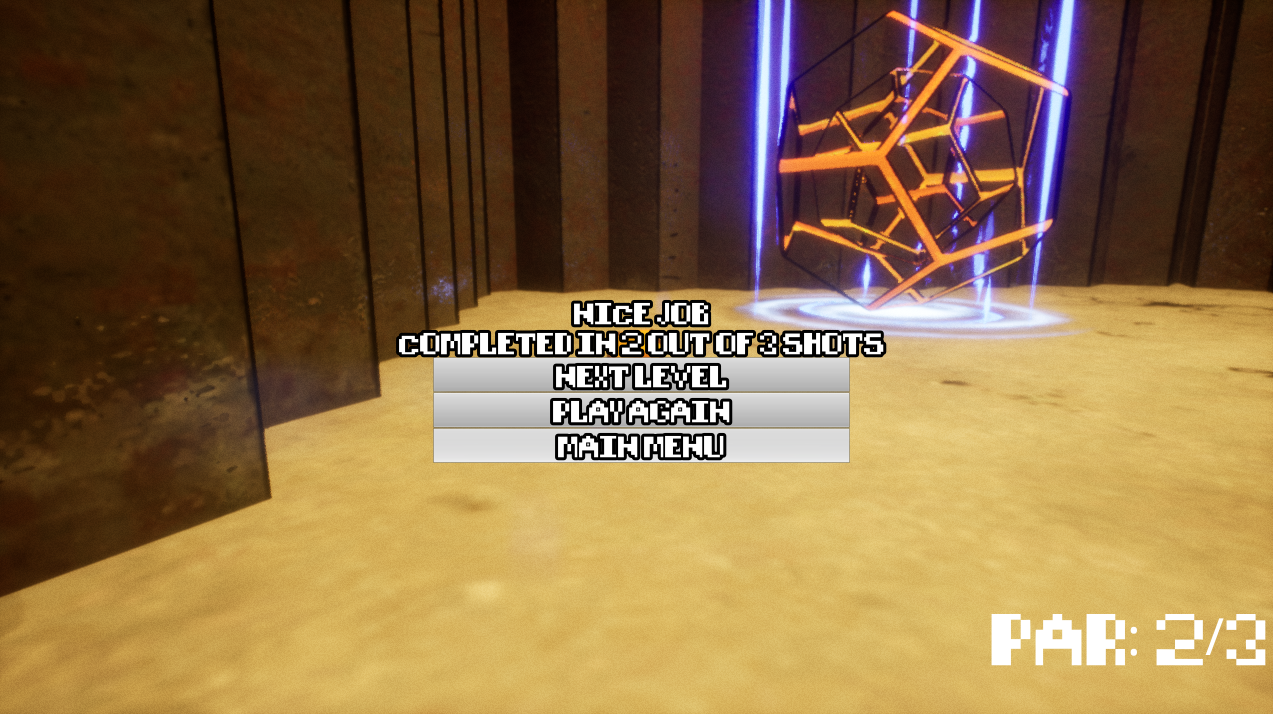


|  |  |
| --- | --- |
| **Launch Force:** | If UltraBall isn’t moving and the player holds the Left Button, a charge bar will appear and UltraBall will glow red depending on how much charge is being applied. If the UltraBall is already moving an interface element stating “Nope” will appear. |
| **Par:** | The current par will be displayed in the bottom right hand corner of the screen. |
| **Escape:** | When the player presses Escape the Pause Menu appears. |

**Pause Menu:**



**Finish Zone:**

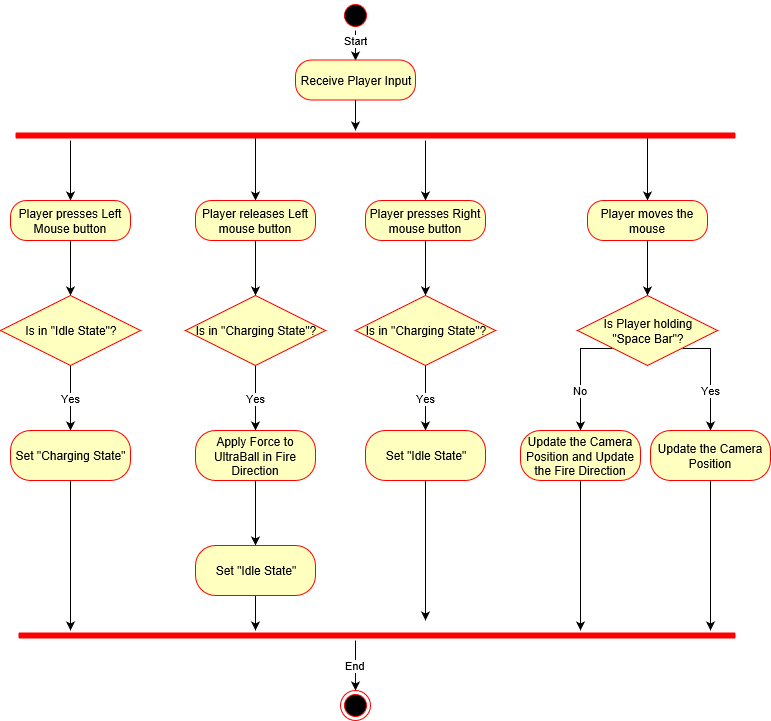
****

**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 can **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 **Max Power Possible at Full Charge Up** variable. The player can only launch UltraBall when it is stationary otherwise they’ll see an interface element stating “Nope”. |
| **Cancel Launch:** | Players can press **right click** while they’re **holding** **left click** to cancel the left click input, resulting in no action happening after they release left click. |
| **Decouple Camera from Shot:** | Players can unlock the camera by holding down the **space button**. This enables them to freely rotate the camera without affecting the shot. |

**Activity Diagrams**

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

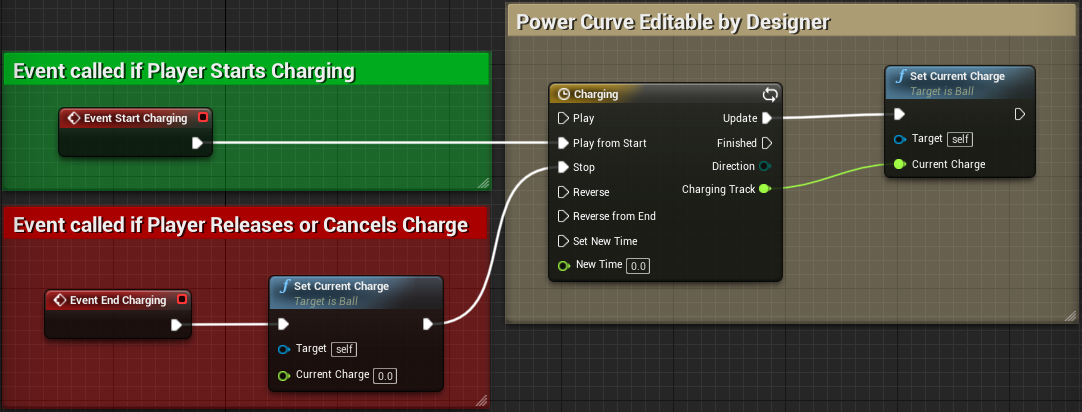


**Charging State:**

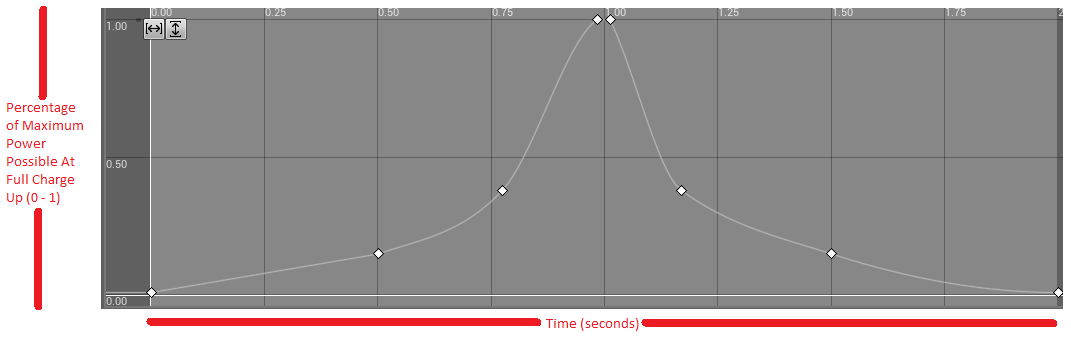
When UltraBall enters the Charging State, a graph function is activated which plays on a two second loop constantly until UltraBall exits the Charging State.

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

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 launching the UltraBall.



Below is a screenshot of the current power curve.

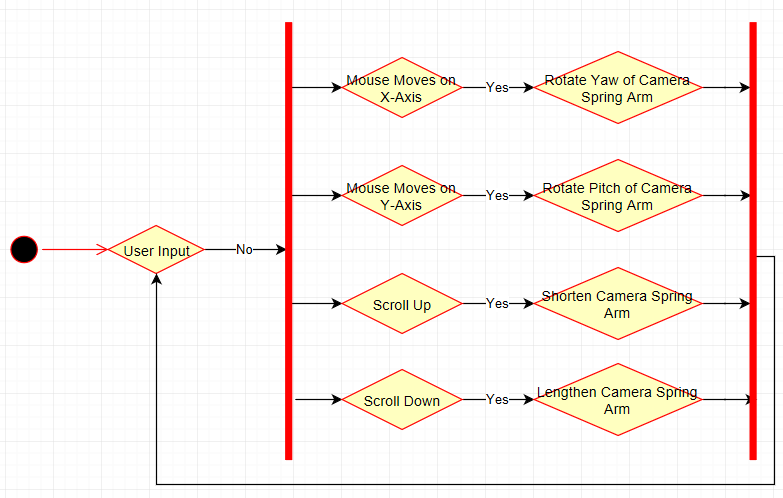
****

For example, if the Player released the Left Mouse button at the 1 second mark of 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 of 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

Below is a list of all Assets used in UltraBall.

|  |  |  |  |
| --- | --- | --- | --- |
| **Meshes** | **Textures** | **Materials** | **Animations** |
| M\_Ultraball | T\_Ultraball\_Base  T\_Ultraball\_Normals  T\_Ultraball\_OCRM | Mat\_Ultraball |  |
| M\_Aim\_Guide |  | Mat\_Aim\_guide |  |
| M\_Bumper45 | T\_Bumper45\_Base  T\_Bumper45\_Normals  T\_Bumper45\_OCRM | Mat\_Bumper45 | An\_Bumper45\_extend |
| M\_Beam2M\_Bigger | T\_Beam2M\_Base  T\_Beam2Mr\_Normals  T\_Beam2M\_OCRM | Mat\_Beam |  |
| M\_Beam2M | T\_Beam2M\_Base  T\_Beam2M\_Normals  T\_Beam2M\_OCRM | Mat\_Beam |  |
| M\_Bench | T\_Bench\_Base  T\_Bench\_Normals  T\_Bench\_OCRM | Mat\_Bench |  |
| M\_Book | T\_Book\_Base\_Alternate  T\_Book\_Normals  T\_Book\_OCRM | Mat\_Book |  |
| M\_Box\_Large | T\_Box\_Large\_Basecolour  T\_Book\_Normals  T\_Book\_OCRM |  |  |
| M\_Brick | T\_Brick\_Base  T\_Brick\_Normals  T\_Brick\_OCRM | Mat\_Brick |  |
| M\_Chair | T\_Chair\_Base  T\_Chair\_Normals  T\_Chair\_OCRM | Mat\_Chair |  |
| M\_Fence | T\_Fence\_Painted\_Base  T\_Fence\_Painted\_Normals  T\_Fence\_Painted\_OCRM  T\_Fence\_Worn\_Base  T\_Fence\_Worn\_Normals  T\_Fence\_Worn\_OCRM | Mat\_Fence\_Painted\_Base  Mat\_Fence\_Painted\_Worn |  |
| M\_Flower | T\_Flower\_Base  T\_Flower\_Mask | Mat\_Flower |  |
| M\_Glass  M\_Glass\_DM | T\_Glass\_Base  T\_Glass\_Normals  T\_Glass\_OCRM | Mat\_Glass |  |
| M\_Gras  M\_Grass | T\_grass  Grass2 | Mat\_Grass  Mat\_Grass\_2  Mat\_Grass\_FoliageOLD |  |
| M\_Hedge | T\_Hedge\_Solid  T\_Hedge\_Mask | Mat\_Hegde |  |
| M\_House | T\_House\_Base  T\_House\_Normals  T\_House\_OCRM | Mat\_House |  |
| M\_Ladder | T\_Ladder\_Base  T\_Ladder\_Normals  T\_Ladder\_OCRM | Mat\_Ladder |  |
| M\_Picnic\_Table | T\_Picnic\_Table\_Base  T\_Picnic\_Table\_Normals  T\_Picnic\_Table\_OCRM | Mat\_table |  |
| M\_Plank | T\_Plank\_Base  T\_Plank\_Normals  T\_Plank\_OCRM | Mat\_Plank |  |
| M\_Shed\_Interior  M\_Shed\_Exterior | T\_Shed\_Base  T\_Shed\_Normals  T\_Shed\_OCRM | Mat\_Shed |  |
| M\_Weeds | T\_Weed\_Base  T\_Weed\_Mask | Mat\_Weed |  |

# Technical Risks

We faced two technical risks during development of UltraGolf.

**Project Pivot**

Half way through the project, the client requested a pivot. This required a complete redevelopment of the core mechanics. An additional day of work was generated by this major pivot putting us slightly behind schedule.

**Visible Stretchiness of UltraBall**

We implemented a system to create a visible stretchiness to UltraBall. It wasn’t a required feature but one it was a feature we believed would make the game more visually appealing. It took several days to develop this feature.

In the end it worked, however, it looked a little weird, made the game more difficult to play, and bloated the code by 80%. As a result, the team decided to scrap this feature all together.