Windows User

Amazing Archery! Design, Implementation and Testing

AE2 – Supplementary Report

Includes: Design, Implementation and Testing

Contents

[Design 2](#_Toc467003065)

[Basic Requirements 2](#_Toc467003066)

[See the Target (so the Player can aim at it) 2](#_Toc467003067)

[Allow the Player to Aim Their Bow (by moving their phone) 3](#_Toc467003068)

[Allow the Player to Adjust the Power of Each Shot (by dragging a touch input instrument (such as their finger), downwards across the screen) 4](#_Toc467003069)

[Player Class 4](#_Toc467003070)

[Implementation 5](#_Toc467003071)

[Basic Requirements 5](#_Toc467003072)

[See the Target (so the Player can aim at it) 5](#_Toc467003073)

[Allow the Player to Aim Their Bow (by moving their phone) 6](#_Toc467003074)

[Testing 7](#_Toc467003075)

[Basic Requirements 7](#_Toc467003076)

[Source Script 8](#_Toc467003077)

# Design

## Basic Requirements

The ‘user stories’ for such requirements, are shown below:

* As a user, I want to be able to see the target down range, so I can aim at it
* As a user, I want to be able to aim my bow by moving my phone, so that aiming is simple and interactive.
* As a user, I want to be able to adjust the power of each shoot by pulling down the on the screen.
* As a user, I want to be able to hit the target and receive points for hitting it.
* As a user, I want to see my arrows flying in an arc so that is mimics real arrows in flight.
* As a user, I want to see a score board on the screen so that I can keep track of my score as I play.
* As a user, I want to be able to pause the game at any time.
* As a user, I want to be able to see a high score list so that I know how well I'm doing compared to my previous scores.
* As a user, I want to be able to see the student's name in the corner of the screen at all times so that I can give credit to the student.
* As a user, I want to be able to see a menu upon starting that game.
* As a user, I want to be able to pick from at least 2 shooting ranges.

For this set of ‘user stories’, the design for each ‘user story’ (from the top), is as follows:

### See the Target (so the Player can aim at it)

This project will use the Unity engine and the perspective for the Player; Is a first-person perspective, looking straight towards the target.

A bitmap for the target (so that the Player can see it), will get shown in the scene, allowing the Player to see a target. A flowchart representation of such, is shown below:

I presume that I can simply drag and drop the bitmap into the scene.

The Player (whom would see the target whilst the game is playing), is not present in the game at this stage of design, their design is detailed in the next ‘user story’:

### Allow the Player to Aim Their Bow (by moving their phone)

For this ‘user story’, at a top level, the design for such is as follows:

However, if the phone’s Orientation has not altered since the last pass of this routine, the point of aim will not require adjustment to match the phone’s Orientation, as this would not procure any results for the Player to see. This detail is delved into, below (as well as stepwise refinement, to achieve the intended level of pseudocode to allow the Player to aim by moving their phone):

1. Has the phone’s orientation received adjustment since the last pass of this Aim Adjustment? If so; go to step 2, otherwise; go to step 3.
   1. Get phone orientation.
   2. If the phone’s current orientation is not equal to the phone’s last orientation (of the last frame); go to step 2, otherwise; go to step 3.
2. Move the Player (and so, their perspective) in accordance with the rotation.
   1. Get the difference in orientation, for the phone’s X-Axis (pitch).
   2. If this difference is positive; go to step 2.3, if the difference is negative; go to step 2.4, or if there is no difference; go to step 2.5.
   3. Move the crosshair upwards, then go to step 2.5.
   4. Move the crosshair downwards, then go to step 2.5.
   5. Get the difference in orientation, for the phone’s Z-Axis (roll).
   6. If the difference is positive; go to step 2.7, if the difference is negative; go to step 2.8, or if there is no difference; go to step 2.9.
   7. Move the crosshair rightwards, then go to step 3.
   8. Move the crosshair leftwards, then go to step 3.
3. Store the phone’s current orientation in a variable, for the next pass of this routine (LastPhoneOrientation).
4. Aim Adjustment is now complete.

Of course, the Player’s aim is the point of aim getting adjusted, which will require the Player Character in game (with a 1st person perspective). They will also require a target reticle, to show their current point of aim.

The HUD for the Player Character, at this stage in design, appears as such:

### Allow the Player to Adjust the Power of Each Shot (by dragging a touch input instrument (such as their finger), downwards across the screen)

If the Player presses and holds their touch input instrument at a certain point on the screen, for greater than a certain quantity of time, prepare to modify their shot’s power level. Pseudocode to describe such, is shown below:

1. Get the current list of contact points with the screen.
2. If there are multiple contact points, within X quantity of time; go to step 3, otherwise; go to step 6.
   1. Get the current contact points of this touch screen.
   2. Check each value to determine if they are not null, if more than 2 values in this list are not null; go to step 3, otherwise; go to step 6.
3. Determine whether the Player is dragging upwards or downwards on the screen, if they are dragging upwards; go to step 4, otherwise; go to step 5.
   1. Check each point in this list.
   2. For each point in this list; if the z component of the contact point is greater than the initial contact point’s z component; go to step 4, otherwise; go to step 5.
4. Increase the power of the shot, then go to step 6.
5. Decrease the power of the shot, then go to step 6.
6. Finish Shot-Power adjustment here.

for each (Vector2 ContactPoint in CurrentContactPoints)

{

If (ContactPoint.z > InitialContactPoint.z)

{

ShotPowerLevel += 0.1f;

}

else if (ContactPoint.z < InitialContactPoint.z)

{

ShotPowerLevel -= 0.1f;

}

}

}

}

### Player Class

At this point (as well as looking at the other basic requirements), the Player will require a class for:

* A representation of the bow they are using
* Retrieval and storage of their Current Shot Power Level
* Retrieval and storage of their Current Score
* Retrieval and storage of their Name
* A reference to their HUD
* Functions to modify the above properties

A behavior script (using C#, to represent the Player class), is to receive implementation, for the requirements listed above.

The first version of this class’s UML Class Diagram, is shown below:

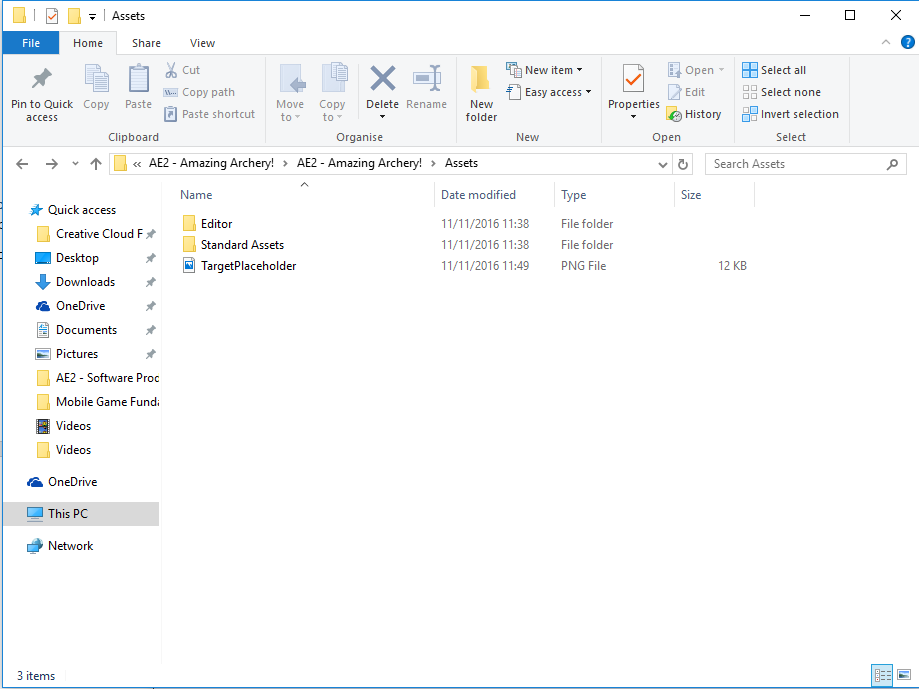


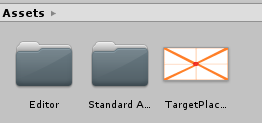
# Implementation

## Basic Requirements

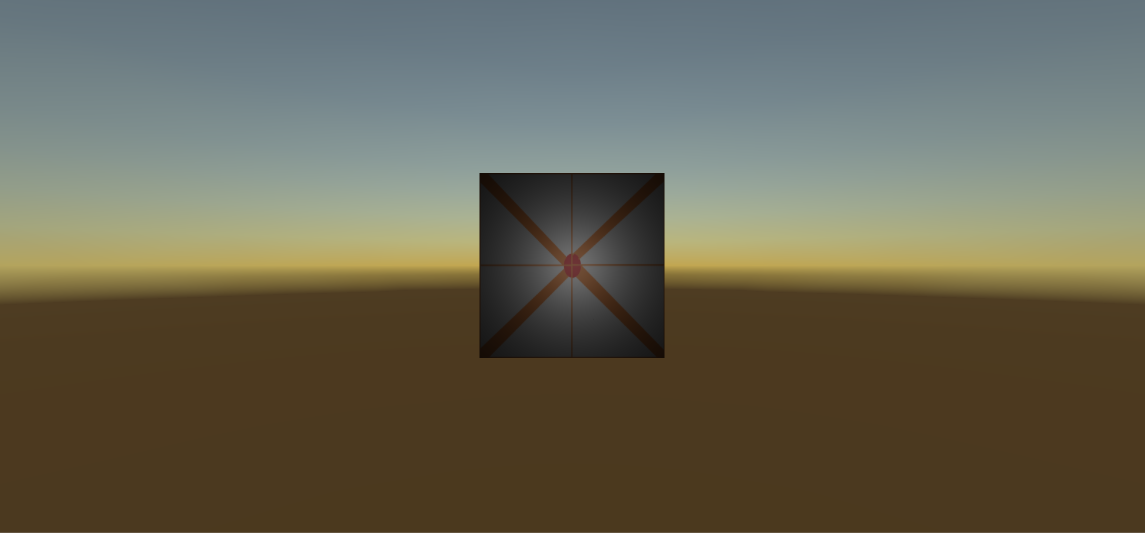
### See the Target (so the Player can aim at it)

First I will create the respective bitmap (so the Player can aim at it). I will use Microsoft Paint for this.

After this, I copy the respective bitmap into the assets folder (via the Windows Explorer):

I can also see this in the Unity’s asset browser:

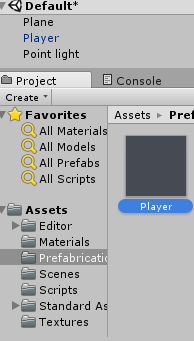
However, I am unable to simply drag and drop this bitmap into the scene (a 🚫 symbol appears), so I create a plane (by pressing the right-mouse button, whilst the focus is on the hierarchy panel of the Unity editor, then in respective context menu: 3D Object->Plane), then drag and drop the bitmap onto it, which is successful (but to see the object, a camera and a light were required, for a viewing perspective and a respective light source, after putting these into the scene in a similar manner to the Plane, as well as altering the transform of all these game objects…):



So, with the alterations to the initial implementation logic; it is now possible to see a target in the game.

### Allow the Player to Aim Their Bow (by moving their phone)

After creating additional folders to add structure to where assets are saved in the project, I dragged and dropped the Camera into the ‘Prefabrications’ folder (see top of next page):

I then added a script to ‘Player’: A C# script known as ‘PlayerBehavior’.

I then implemented the respective functionality for this ‘user story’ (as per the design for this ‘user story’, from the ‘Design’ section), into this script (see Fig. 1 of ‘Source Script’, for the script used in this instance).

Changes from the original design, are that of the following:

* PhoneOreintation type modified to Quaternion (from Vector3), as the orientation of a device’s gyroscope uses a Quaternion instead of a Vector3
* AdjustAimPointForPhoneOrientation would not require a parameter for the above (as it has class scope)

# Testing

## Basic Requirements

Black Box Testing is used here:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Expected Results | Actual Results | Successful? |
| 1 | Precondition: Game is in play mode, with the Unity Editor.  Upon starting the game, the Player can see the Target. | The Target is visible, at the centre point of the Player’s perspective. | The Target is visible, at the centre point of the Player’s perspective. | True |

# Source Script