



Kuan-Ting Chin

33430072

Gamification utilizing beacons to pinpoint player locations or spawn interests with Augmented Reality on mobile devices.

MSc Computer Games Entertainment

Postgraduate Thesis

Department of Computing

Richard Leinfellner

2016

Table of Contents

<u>Abstract</u>	2
<u>Section 1: Introduction</u>	3
1.1 – Project Description	3
1.2 – Aims and Objectives	3
<u>Section 2: Contextual Report</u>	4
2.1 – Literature Review	4
2.1.1 – Virtual Reality	4
2.1.2 – Augmented Reality in Mobile Gaming	5
2.2 – Market Research	6
2.2.1 – Mobile Gaming Overview	6
2.2.2 – Current AR Games on mobile	7
<u>Section 3: Development</u>	10
3.1 – Project Plan	10
3.2 – Final Artefact and Overview	10
3.2.1– Tools utilized and code architecture	10
3.2.1.1 – Menu Control	15
3.2.1.2 – User Interface	20
3.2.1.3 – Player Rotation and Controls	23
3.2.1.4 – Enemy A.I	25
3.2.1.5 – Beacons	28
3.3– Risks and problems encountered	31
3.3.1 – Beacons android build	31
<u>Section 5: Conclusion</u>	34
<u>Section 6: References</u>	35

Abstract

The main purpose of the thesis is to research potential gamification utilizing physical technologically devices which will provide users with new experiences in mobile gaming in Augmented Reality space as well as understanding the current inclination of the mobile games market, with the recent release of titles such as Pokemon GO. The project will go through multiple iterations and builds for the android device in order to use triangulation with physical mini beacons.

Section 1: Introduction

1.1 Project Description

The project will focus on utilizing beacons bought from estimate to try pinpoint the location of the player via triangulation, as well as potentially utilize the beacons to spawn interesting events or objects. The main build of the game will be built for android, and tested using the NVidia Shield Tablet. The main project will be created using Unity3D, Vuforia APK and free art assets from the unity store/online, as the art aspect will not be the main focus of the project, with the scripting part of the project will be done via C#.

1.2 Aims and Objectives

The primary aim of this project is to see the viability of external hardware being able to influence gameplay on mobile game devices. For this purpose, Augmented Reality will be explored as opposed to virtual for better integration of the technology. With this in mind, I hope to achieve an interesting outcome of beacons not only processing player location via triangulation, but also to trigger events within the game or serve as a basis of reference.

Section 2: Contextual Report

2.1 Literature Overview

2.1.1 Virtual Reality

With new technology prominent to the general public now in the gaming industry to the likes of Oculus Rift or the HTC Vive Gear for virtual reality, it has brought new possibilities and opportunities for games companies to compete and monopolize a new gaming sector. The tech behind the immersion factor for recent devices is 2 screens which allow users to calibrate their IPD, or Interpapillary distance. Oculus has concerns for the competition (Lemon, 2014), Lemon reports in his article that Nate Mitchell of Oculus “[was] worried it might impact the entire fledgling industry”. He goes on to say that “It’s very hard to create presence, and it’s very easy to break the illusion” which brings Mitchell to conclude that there’s a major problem that competitors can’t fix if the hardware is lacklustre which would result in poor experience, giving off potential backlash on the entire technology.



Figure12.1:How games appear on the Oculus. Note the double screens for the IPD measure.

2.1.1 Augmented Reality

With that in mind, Augmented Reality has also become a huge phenomenon, initially thanks to Google and its google glass technology, which focused primarily on attempting to “[Replace] the Smartphone With glasses” (Albanesius, 2012). This was a huge breakthrough research for the entire tech industry, as it provided a step forward into blending the fine line between virtual and real world. Although a huge step forward, and having been sold to consumers for a limited time, Google eventually pulled the technology off the consumer market and instead wanted to “...continue to invest in its **Glass at Work**...and plans to release a new version of its wearable “when its ready”.”(Curtis, 2015). This could primarily be linked to the fact that the hardware and technology behind it was lacking, such as battery life, image problems and a lack of application support.

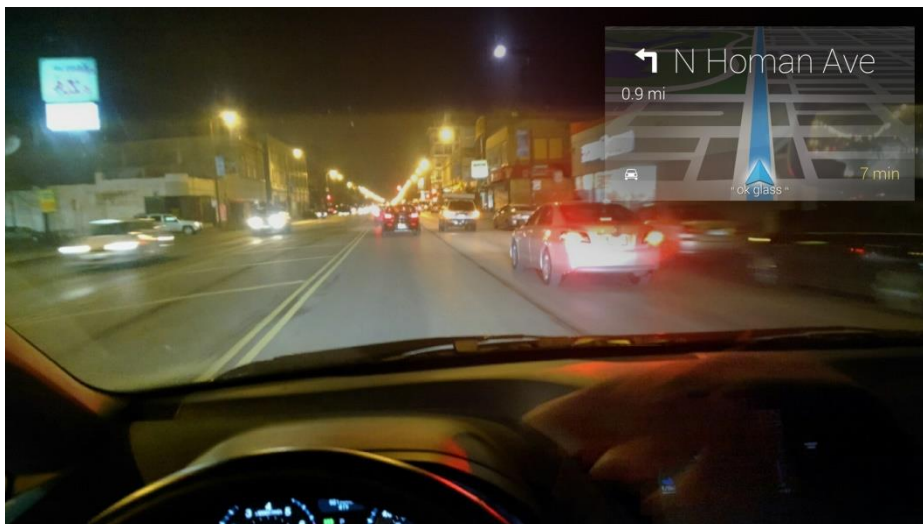


Figure 1.2: Example of Google Glass user interface.

A major issue to be found with google glass was the hostility users faced when wearing the glasses. This prompted google to “issue guidelines for its customers if they encountered any curiosity or hostility” (Kalinauckas, 2015). The other major issue that correlates to this is privacy. The main privacy concern ranged from a fear of a breach of privacy when users were out, thus being banned from certain establishments, to political legislations, especially during 2013 where the tensions between Russia and Ukraine were high.

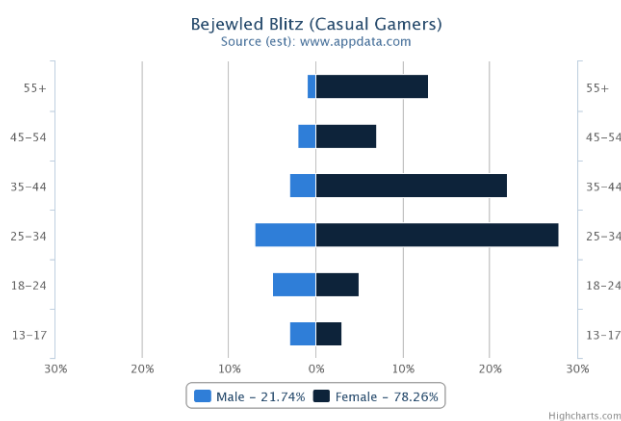
That being said, Augmented Reality is not all doom and gloom, as famed director, Peter Jackson “believes that augmented reality technologies...will be used “as much as, if not more” than smart phones [in ten years]”(Jarvis, 2016). Jackson believes that “Once you can create the illusion of solid objects anywhere you want, you create new entertainment opportunities.” The most recent mobile release, Pokémon GO, is a perfect example of Augmented Reality being a global phenomenon, which will be covered in the market research section.

2.2. Market Research

2.2.1 Mobile Gaming Overview

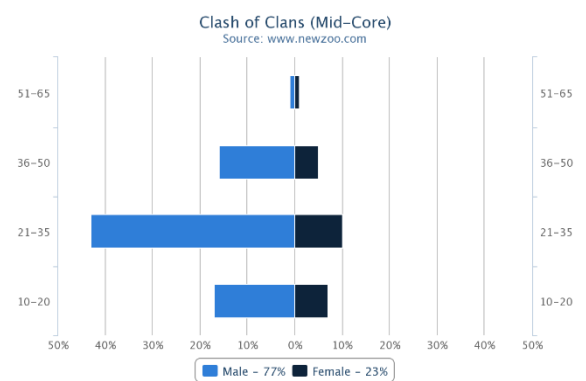
Mobile gaming has come a long way and has opened up a huge market for many consumers, ranging from casual players to more hard-core and serious players. With “500 games launched per day on iOS” (Graft, 2015) it was a significant figure just to indicate how rapidly the mobile gaming market was growing, and just how fierce the mobile market competition is. Graft also listed figures of the mobile platform during a panel from Mike Rose, where low end figures ranged from 0-2000 sales, mid end from 2000-30000, and high end from 30000 to 2.5 million sales, with Mike Rose on closing that “Mobile is risky, but yields big success”.

A demographic of different gamer types was listed on the Magmic Developer website(Mason, 2013) in 2013 garnered interesting gamer habits, age, and gender on mobile gaming compared to Call of duty on the pc/console, listing interesting habits or traits of each as well.



Demographic breakdown of Clash of Clans (Mid-Core):

Touted as the king of mid core games with peak revenues of (est) \$700,000/day.



Demographic breakdown of Candy Crush Saga (Unicorn):

This "King" title struck a silver bullet in 2011 with its ease of play and addictive progression loop.

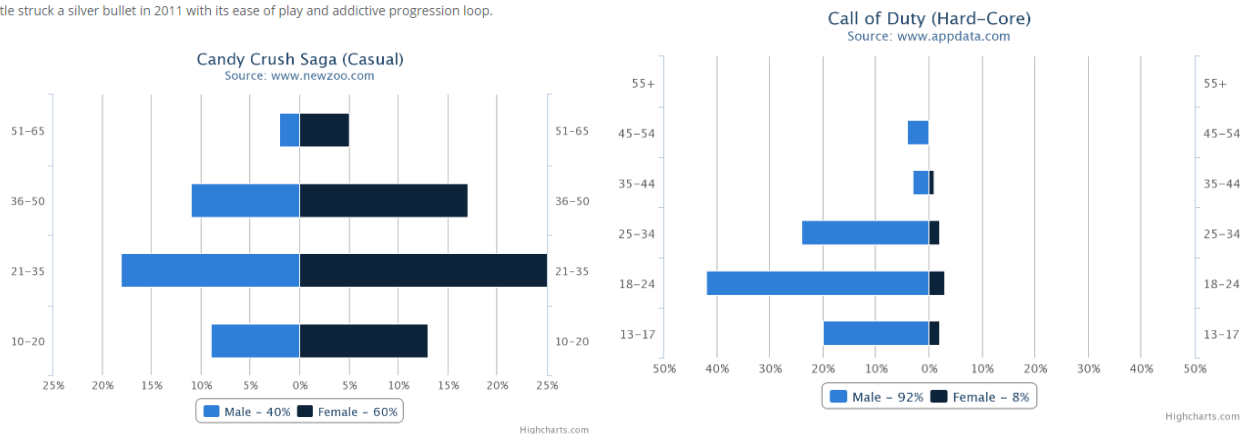


Figure 2.1-Figure 2.4: Mobile Games demographics compared to the likes of Call of Duty. The graph also distinguishes gamer types, such as casual players to more hardcore players. Each demographic listed had player habits, where it deduced the player's session times, as well as occurrences where the player would normally play.

2.2.2 Current AR games on mobile

As stated previously, Pokemon GO has achieved phenomenal success. Developed by Niantic, Inc., the company behind Ingress, the mobile Augmented Reality MMO game, The game allowed Nintendo's shares to skyrocket by 35%, and although the game was "[not] the first AR game, what it did have was brand recognition and an existing audience to tap into." (Willis, 2016).

The game utilizes technology similar to that of google maps, primarily due to John Hanke, the founder of Niantic, having worked on Google Earth and Google maps. According to Hanke, the game also determines pokestops and gyms, in game locations of interest via geo-tagged photos from google, ranging from "Things that were public artwork, that were historical sites, that were buildings with some unique architectural history or characteristics..." (Bogle, 2016). The game also uses characteristics from the maps, such as whether the current location the player is in is a mountainous regions, or if there is water nearby, which would prompt certain pokemon to appear depending on the terrain. The mechanics of the game is simple: to walk around and catch pokemon determined by the player location, whilst providing the effect of augmented reality built into the gameplay. The player is then able to also battle at gyms, representing the player's respective teams.

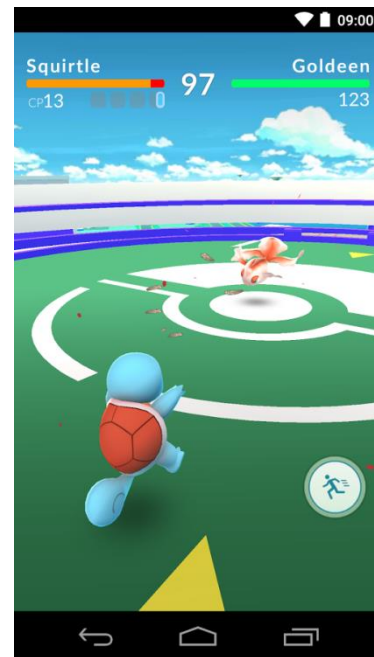


Figure 2.5: Gameplay of pokemon GO, the pokemon appears on screen with the option to toggle augmented reality on or off.

Figure 2.6: Gym battles in Pokemon GO.



Figure 2.7: Map of player location in game, with location of pokemon and pokestops nearby on the map.

Ingress, the game Niantic worked on previously, is also one of the influential technology factors that were used to build Pokemon GO, and an interesting competitor on the mobile AR market. In Ingress, players are split between 2 teams, with the main objective of dominating territory via building portals at certain landmarks and locations. This technology is what Pokemon GO uses to determine the noticeable landmarks for pokestops and gyms.

Despite being a huge phenomenon, Pokemon GO also suffered criticism, specifically linked to injuries of players, ranging from slight injuries to death. Pokemon GO was also considered a threat to security in both Iran and Russia, banning the device. Raspopina cites “on Monday Iran became the first country to ban the augmented reality game, citing security concerns...in Russia officials have been whipping up hysteria over possible hidden dangers in the western made game” (Raspopina, 2016).

Section 3: Development

This section will highlight the development of the artefact from initial prototyping with various iterations to the final artefact.

3.1. Project Plan

The project in question is a mobile artefact with the ability to incorporate Augmented Reality and utilize physical beacons for means of player triangulation. This in theory should work similar to pokemon GO's player tracking, but on a significantly downscaled proportion. The game will follow a first person endless gauntlet shooter, with enemies constantly spawning and providing the player with opportunities to fight back by means of attacking by themselves or from dropping traps within the virtual/augmented space.

3.2. Final Artefact and Overview

The final artefact will primarily focus on implementing classes for players to choose from, before being taken into the main portion of the game. The artefact will be built for android, but specifically for the NVidia Shield Tablet to test, and implementation of augmented reality will be done utilizing Vuforia APK for Unity. As for the physical part of the artefact integration, the project will be utilizing estimate beacons as the beacon of choice. The beacons will also use an inbuilt plugin on Unity known as iBeacons, developed by Kaasa Solution. More in-depth explanation of the tools utilized will be covered in the next section.

3.2.1. Tools and Logic background

As stated previously, the project will utilize Vuforia APK, iBeacons, and estimate beacons. The Vuforia APK provides image tracking of planar images and 3D target in real time, allowing the image registration to occur and provide developers with feedback on the tracked image. This also enables developers to position and orient virtual 3D objects within the virtual space in relation to real world images through the development device camera.

Vuforia's website also includes a development portal, which provides users with the appropriate license key for the Unity APK, and also a target Manager.

Target Manager is Vuforia's database of scanned images or objects, which will distinguish and allocate targets usable for the developer depending on any unique features the target image or object has, and which specific database the developer uploaded the files into. Once the developer has uploaded images or data of their choice, Vuforia then creates the applicable data within its database, and provides a number of information returned to the developer, such as the features vuforia determines to be points of interest, and an augmentable rating system on how efficient the image will show up in the build.

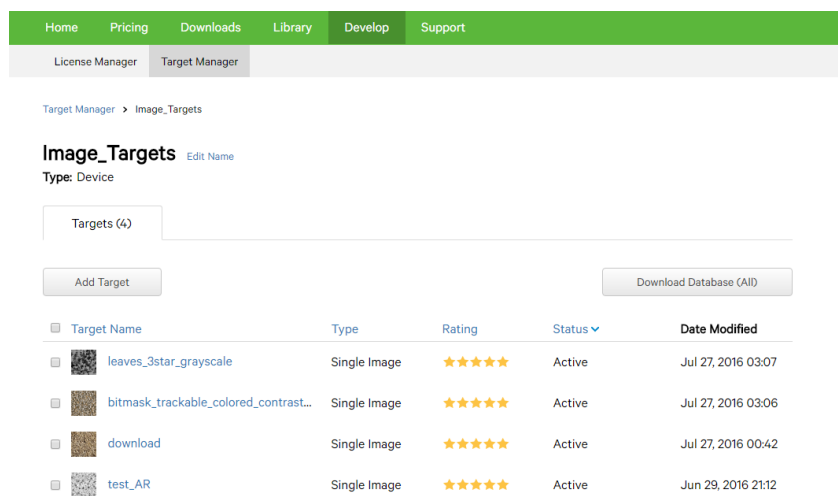


Figure 3.1: Vuforia developer portal. The target manager is the database tool, which allows users to upload images or 3D objects they'd like to use.

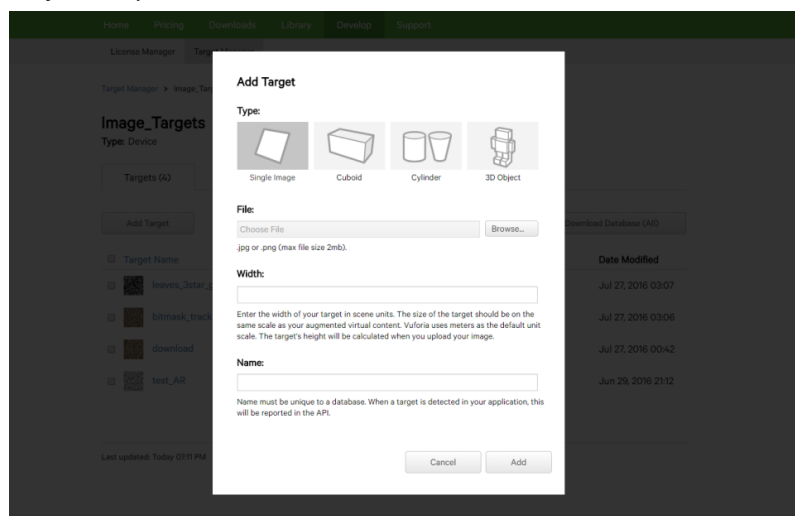
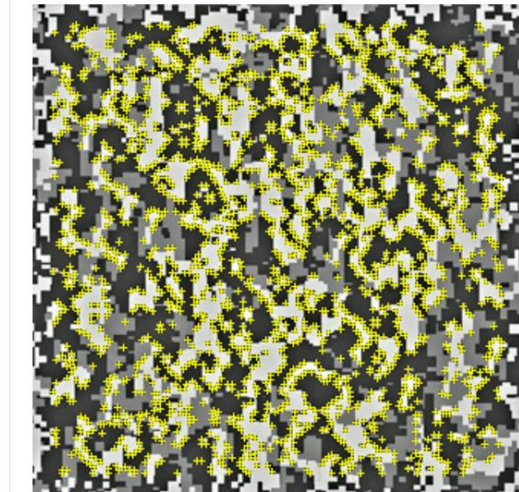


Figure 3.2: Once the user selects add target, a window appears allowing the developer to choose the type of target they want, be it an image, 3D object, or such, then allowing the user to choose the relevant file and name.

bitmask_trackable_colored_contrasted

Edit Name Remove



Update Target Hide Features

Type: Single Image

Status: Active

Target ID: 360bdd06286542afa52f104e76dc31db

Augmentable: ★★★★★

Added: Jul 27, 2016 03:06

Modified: Jul 27, 2016 03:06

Figure 3.3: Uploaded image, with excellent feature points indicated by yellow plus signs.

In order for Vuforia to use 3D image scanning however, it would require the user to download the Vuforia Object Scanner APK on an android device and utilize an Object Scanning target image. This too, is similar to the 2D image scan, but will feature 3D objects for point features. The image is placed within the confines of the ‘scanning’ area, and an image similar to borders would appear displaying the object that is to be scanned within the scanning template zone.

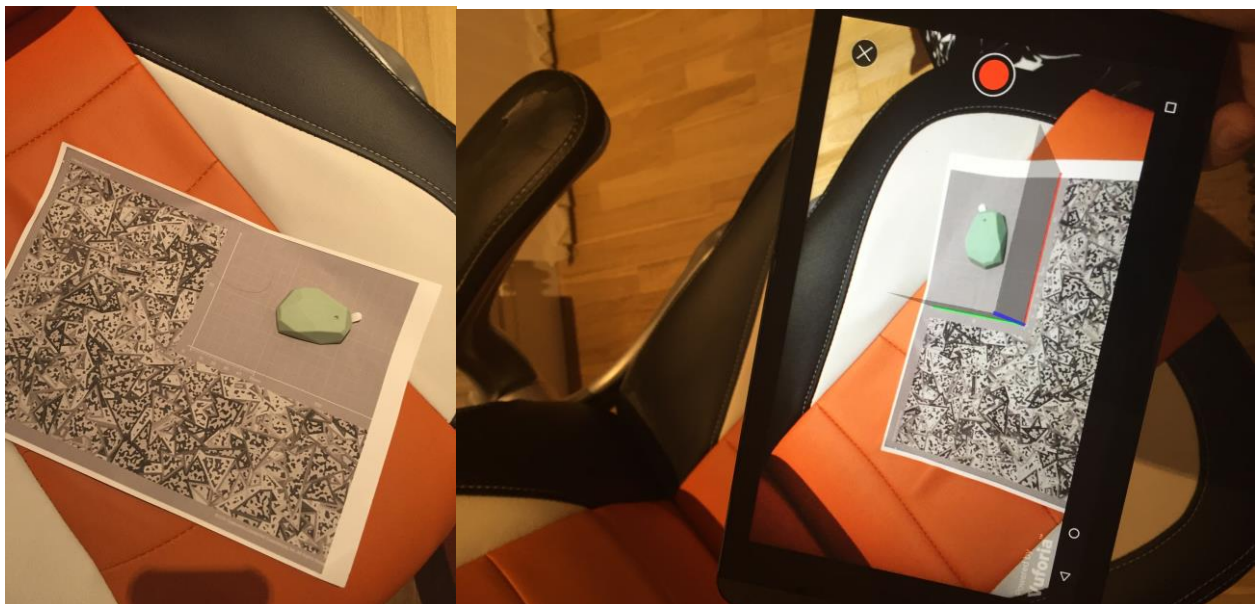


Figure 3.4: Example of the 3D Object scanner with the image tracker. Notice the barrier for the 3D objects.

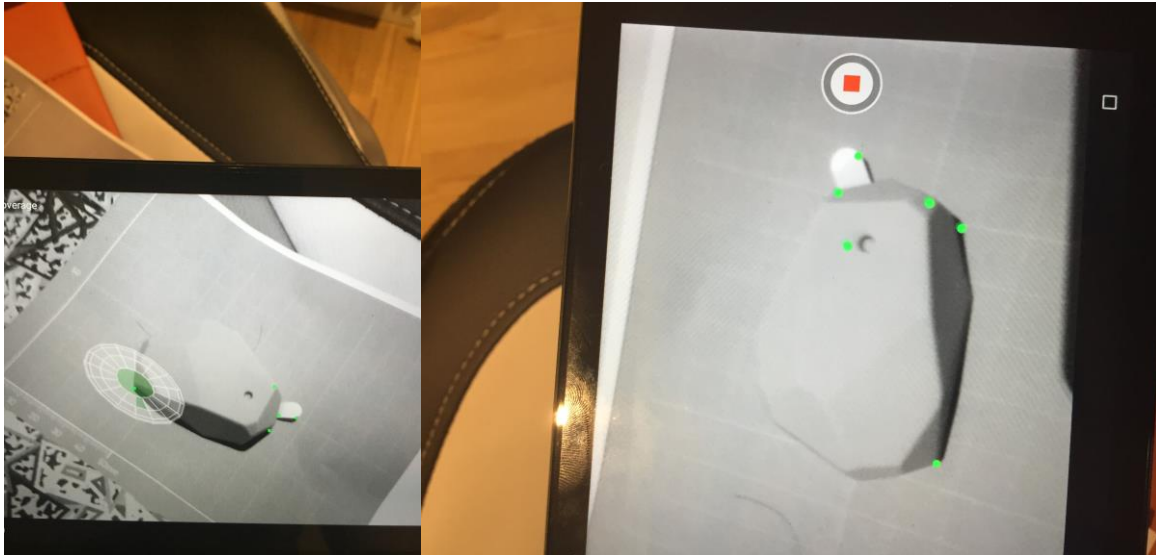


Figure 3.5: Scanner detecting the object and plotting unique feature spots with green dots.

The estimate beacons also utilize a similar developer portal, but rather than a database online to download, it uses cloud technology to modify each and every individual beacon, modifying the property such as generating new UUID's (Universally Unique Identifier), Major and Minor broadcast packet channels, which are essentially channel segregates for beacons. In order for modifications to be applied from the cloud onto the beacons, the use of the estimate cloud app is required. As the app is available for both iOS and android, using it on either to change the beacons do not have any known implications.

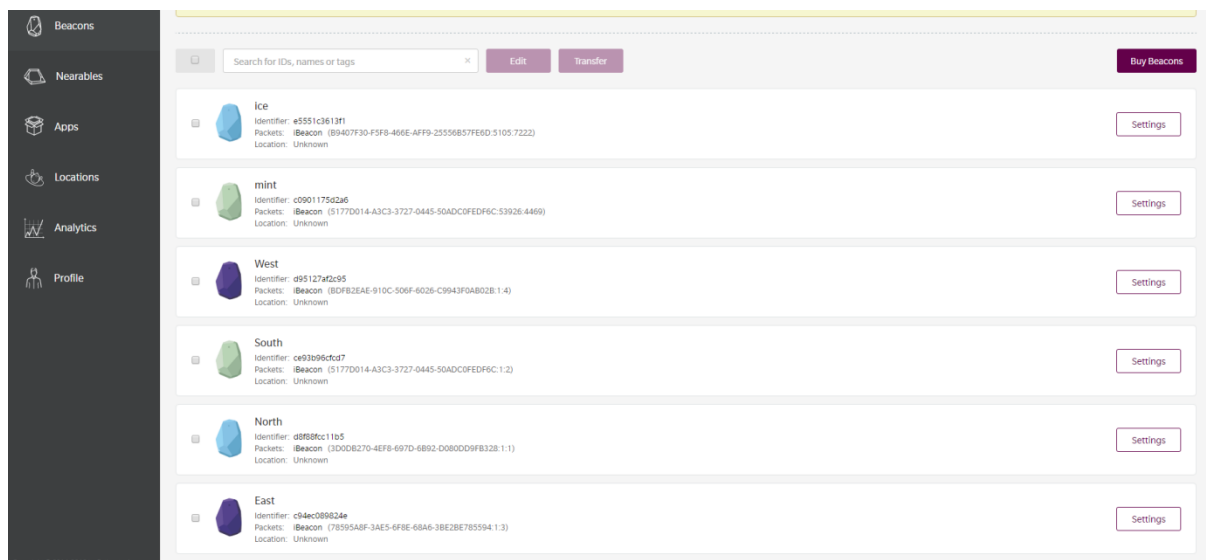
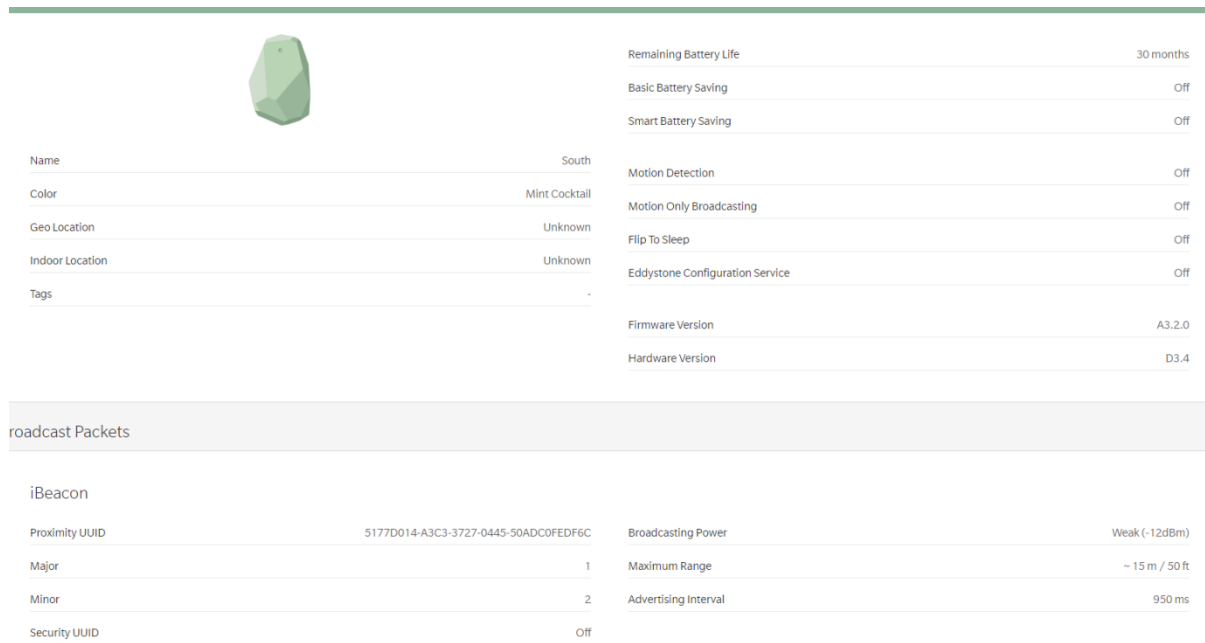


Figure 3.6: Estimote cloud user interface with relevant beacons attached to developer account.



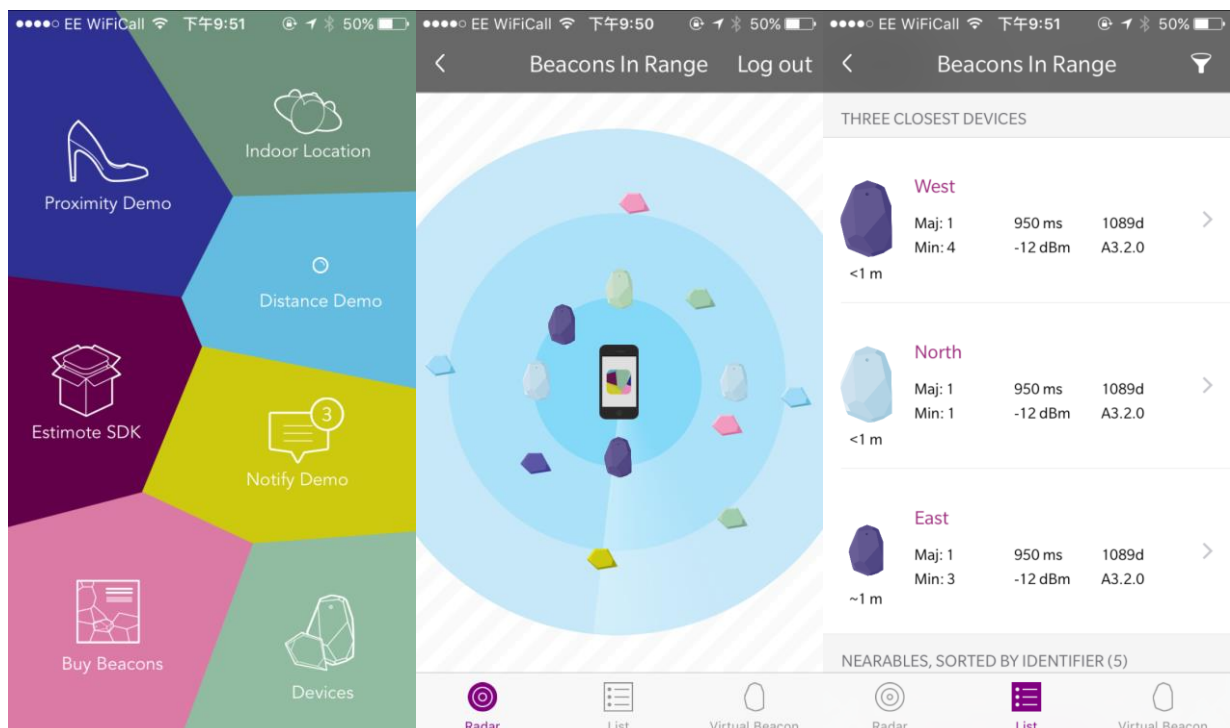
The image shows the configuration interface for an Estimote beacon. At the top left is a green hexagonal beacon icon. Below it, a table lists various settings:

Name	South	Remaining Battery Life	30 months
Color	Mint Cocktail	Basic Battery Saving	Off
Geo Location	Unknown	Smart Battery Saving	Off
Indoor Location	Unknown	Motion Detection	Off
Tags	-	Motion Only Broadcasting	Off
		Flip To Sleep	Off
		Eddystone Configuration Service	Off
		Firmware Version	A3.2.0
		Hardware Version	D3.4

Below this is a section titled "Broadcast Packets" which contains an "iBeacon" configuration table:

Proximity UUID	5177D014-A3C3-3727-0445-50ADC0FEDF6C	Broadcasting Power	Weak (-12dBm)
Major	1	Maximum Range	~ 15 m / 50 ft
Minor	2	Advertising Interval	950 ms
Security UUID	Off		

Figure 3.7: Individual beacons, with editable information.



The image shows the Estimote beacon app interface on a smartphone. The app has a colorful hexagonal menu on the left with options: Proximity Demo, Indoor Location, Distance Demo, Estimate SDK, Notify Demo (with a '3' badge), Buy Beacons, and Devices. The main screen displays a radar view of beacons in range, with a central smartphone icon and concentric circles representing range. To the right, a list titled "THREE CLOSEST DEVICES" shows details for three beacons:

Direction	Distance	Maj	Min	Interval	Power	Firmware
West	<1 m	1	4	950 ms	-12 dBm	A3.2.0
North	<1 m	1	1	950 ms	-12 dBm	A3.2.0
East	~1 m	1	3	950 ms	-12 dBm	A3.2.0

Below this list is a section titled "NEARABLES, SORTED BY IDENTIFIER (5)". The bottom navigation bar includes icons for Radar, List, Virtual Beacon, Radar, List, and Virtual Beacon.

Figure 3.8: Estimote beacon app. Detects beacons in range, as well as applies changes the user had made to the cloud onto the beacons.

In order for the beacons to work, the beacons required an addon in Unity known as iBeacon to work. iBeacon is a sender and receiver addon for Unity, allowing beacon detection as well as creating a virtual beacon of itself.

3.2.1.1. Menu Control

The menu of the artefact is simple: to create something which instantly gives the user an idea that augmented reality is the main part. By providing the user 4 targets, the user is able to pick a character, depending on what target image the player currently scans over.

In order to create the ability to pick a character and scan images, the project downloaded the database holding 4 images, an image for each class. Once imported into Unity, the project will have a Vuforia tab on the top of the unity development window, covering the documentation, and an ARcamera prefab, which has all prebuilt functions to start utilizing augmented reality.

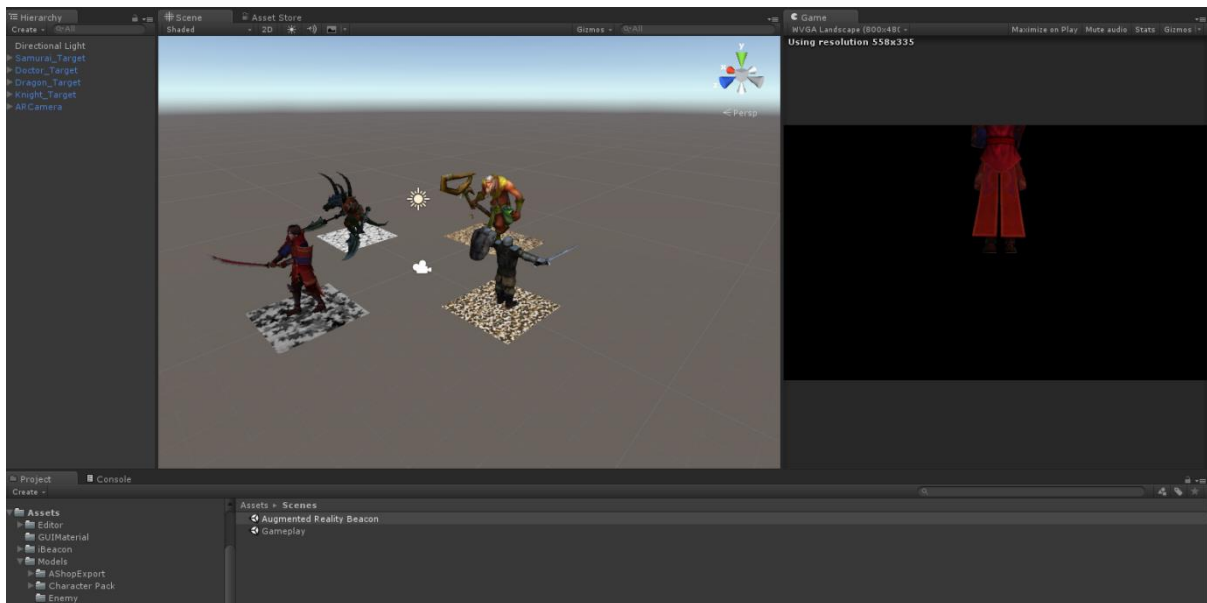


Figure 3.9.1: Scene with implemented Vuforia APK.

Once in the scene hierarchy, the inspector will require the user to have a license key from the vuforia website, which is free to sign up for granted the project is not to be for commercial use. Since the camera is replaced with the ARcamera prefab, this requires a compatible webcam type device for use during development playtests. The issue however, is that webcam compatibility only works with 32 bit unity, instead of the 64 bit version.

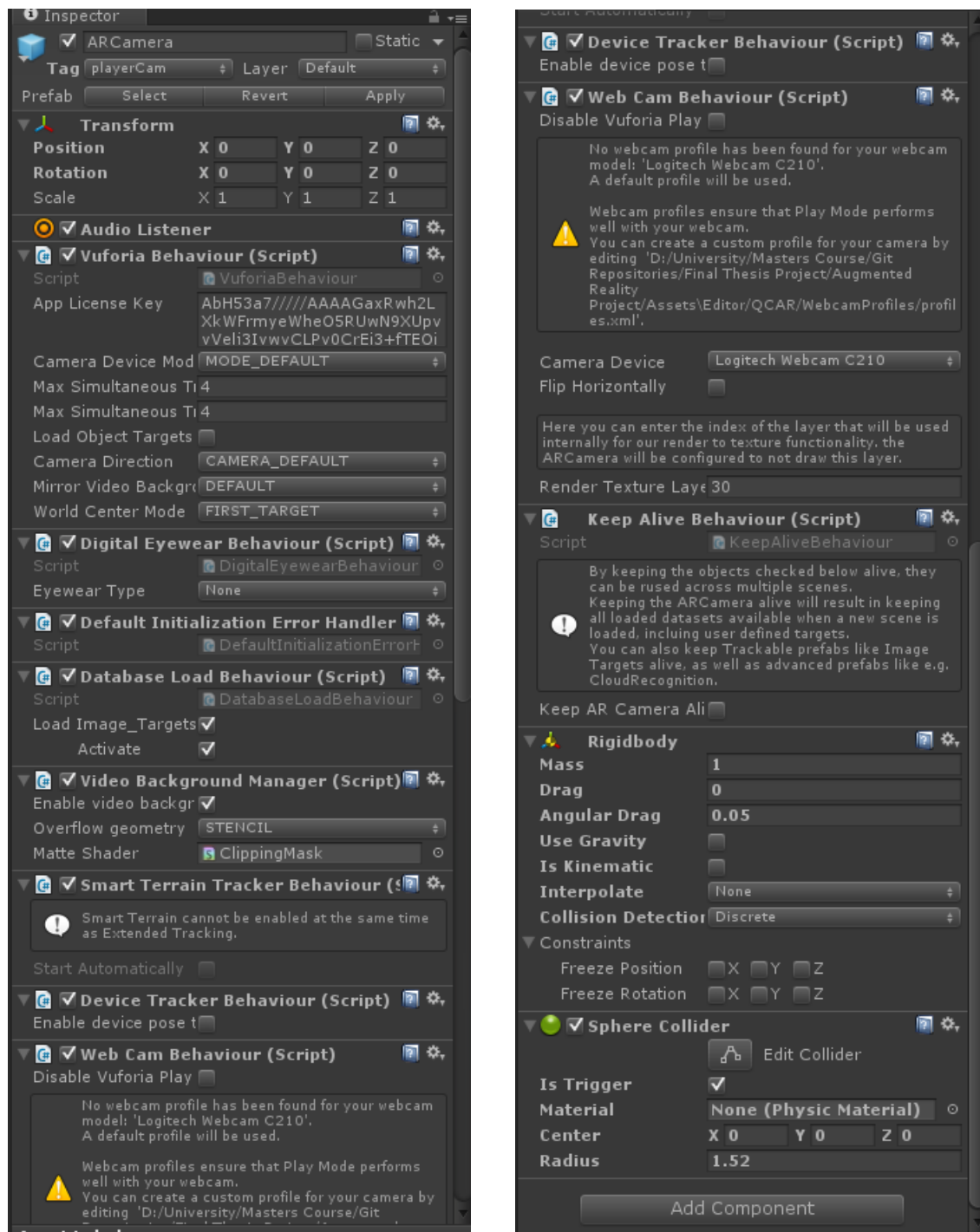


Figure 3.9.2: Vuforia Behavior script requiring the license code, and prebuilt webcam behavior.

The max simultaneous setting limits the amount of possible scanned images on the screen at once. Also noticeable is the camera device default to a Logitech for development purposes, but once built, will run through the android devices native camera just fine.

There is a script created which handles events when an image appears on the screen, known as testPopUp.cs. within the script is a function OnTrackableStateChanged:

```
public void OnTrackableStateChanged(
    TrackableBehaviour.Status previousStatus,
    TrackableBehaviour.Status newStatus)
{
    if (newStatus == TrackableBehaviour.Status.DETECTED ||
        newStatus == TrackableBehaviour.Status.TRACKED ||
        newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
    {
```

This is a function from the vuforia behavior script, distinguishing when a tracked object is detected, and when it is tracked. This provides developers with the ability to toggle events and such when they detect images are currently being tracked from one state to another.

The script also declares an OnGui function to create a button to show up when a targetable image appears, one which matches within the vuforia database, and allows the user to click the button prompting the user to select the chosen character. The project utilizes some free prebuilt characters, so options and art direction may differ differently from final experience, but the main priority is again to research potential possibility of the hardware for mobile gaming.

```
void OnGUI()
{
    if (mShowGUIButton)
    {
        // draw the GUI button
        IEnumerable<TrackableBehaviour> tbs =
TrackerManager.Instance.GetStateManager().GetActiveTrackableBehaviours();
        foreach (TrackableBehaviour tb in tbs)
        { f (GUI.Button(mButtonRect, "Select Hero"))
            {
                if (tb.name == "Samurai_Target")
                {
                    Selected = true;
                    // Debug.Log("Samurai Selected");
                }
                else if (tb.name == "Dragon_Target")
                {
                    Selected = true;
                    // Debug.Log("Dragon Selected");
                }
                else if (tb.name == "Knight_Target")
                {
                    Selected = true;
                    // Debug.Log("Knight Selected");
                }
                else if (tb.name == "Doctor_Target")
                {
                    Selected = true;
                    // Debug.Log("Doctor Selected");
                }
            }
        }
    }
}
```

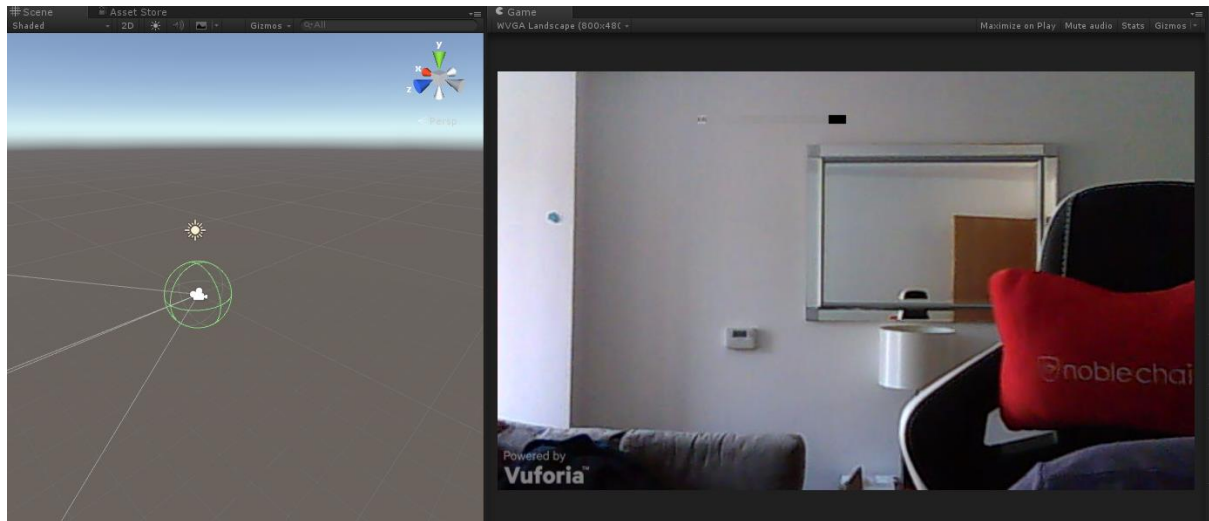


Figure 3.9.3: Example image of Vuforia, utilizing webmcam camera. No images shown.

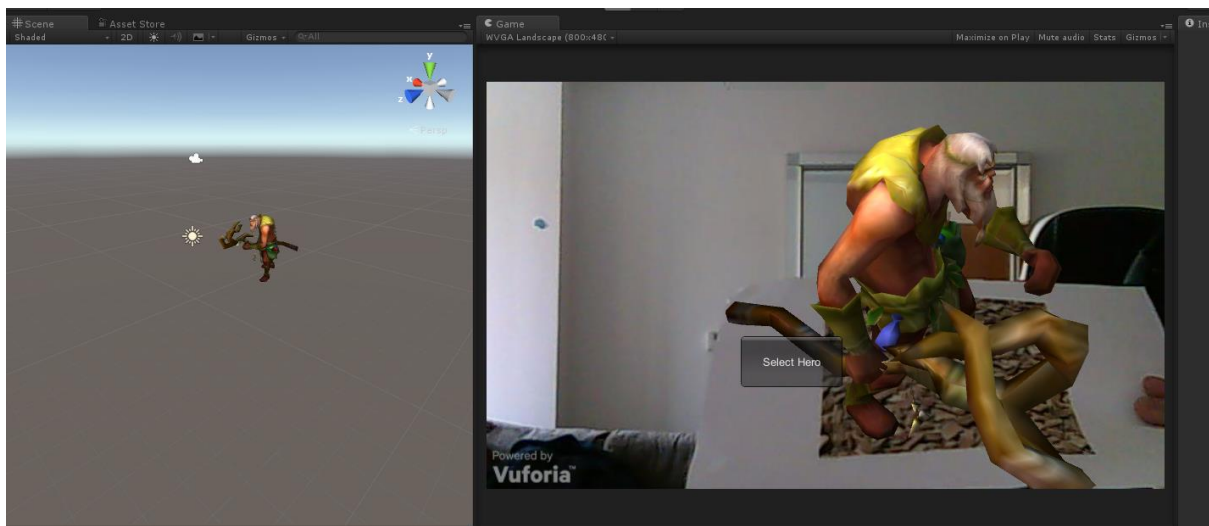


Figure 3.9.4: Image appearing via scanned target.

An interesting feature that Vuforia offers is the ability to use smart terrain tracking, or extended tracking. This feature allows the consumer to scan the target image, but then remove the target image from the scene, without much deterrence of the visual object. This is due to the ability for Vuforia to not only scan the target image, but its surrounding as well. Once it has scanned the target Image, it memorizes certain key terrain elements, and keeps the visual object on screen. This can all be done with the incorporation of a single line of code in the update function.

```
bool success = track.PersistExtendedTracking(true);
```

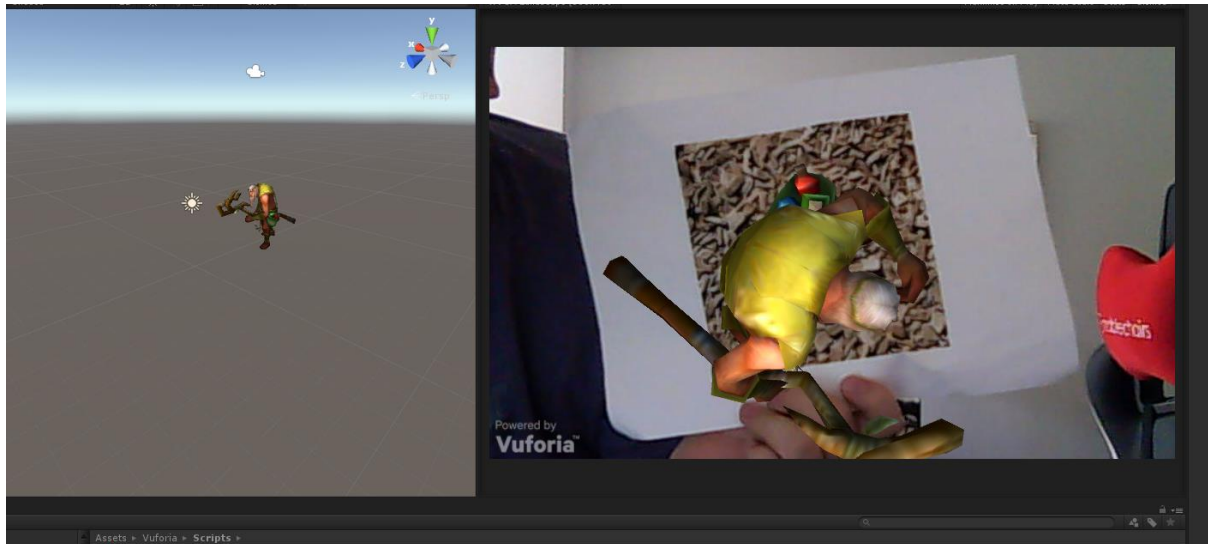


Figure 3.9.5: Notice the image target required to spawn the caster character.

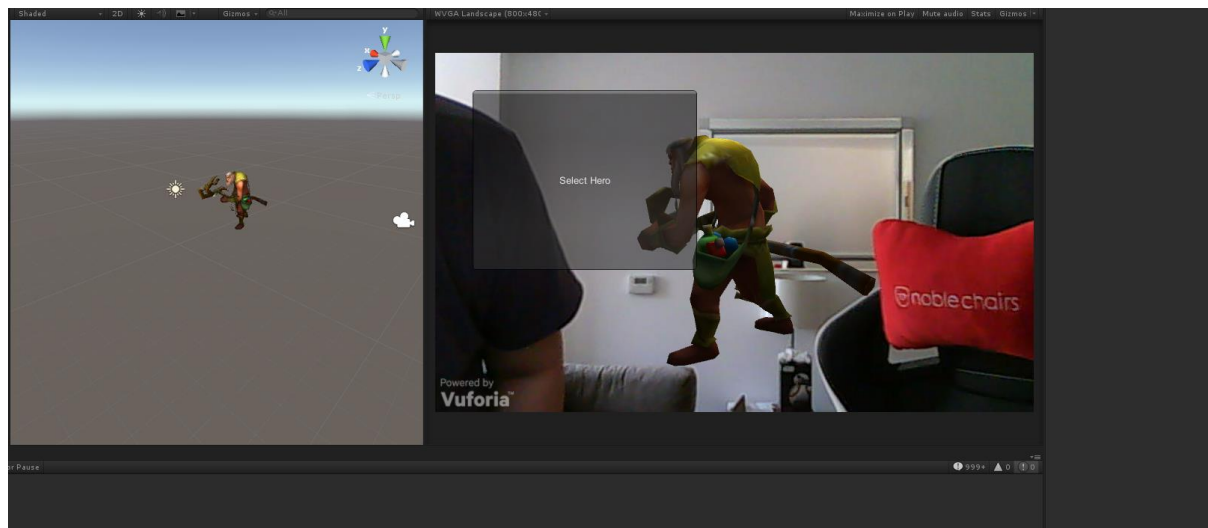


Figure 3.9.6: Removed image target, object is left behind thanks to background information.

This feature alone just signifies how far the technology has come, and for mobile nonetheless.

Once the user has made their choice, the selected virtual object would be magnetized towards the user's screen before switching to the main part of the game. This is done via the Selected function.

```
if(Selected)
{
    Vector3 absorbField = player.position - transform.position;
    float index = (5 - absorbField.magnitude) / 5;
    GetComponentInChildren<Rigidbody>().AddForce(-10f*absorbField*index);
    Selected = false;
}
```

Due to the artefact focusing more solely on the beacons implementation, only the caster class is playable in the artefact.

3.2.1.2. User Interface

The main game required the scene set up similar to the menu scene, having the vuforia ARcamera prefab in the hierarchy as the main camera once again. The modification to it would be to include the player character as a child, as this would allow the character to rotate or move in accordance to the camera's current position and rotation. The player's UI is created utilizing the canvas tool, with simple elements like health, crosshair, and attack button.

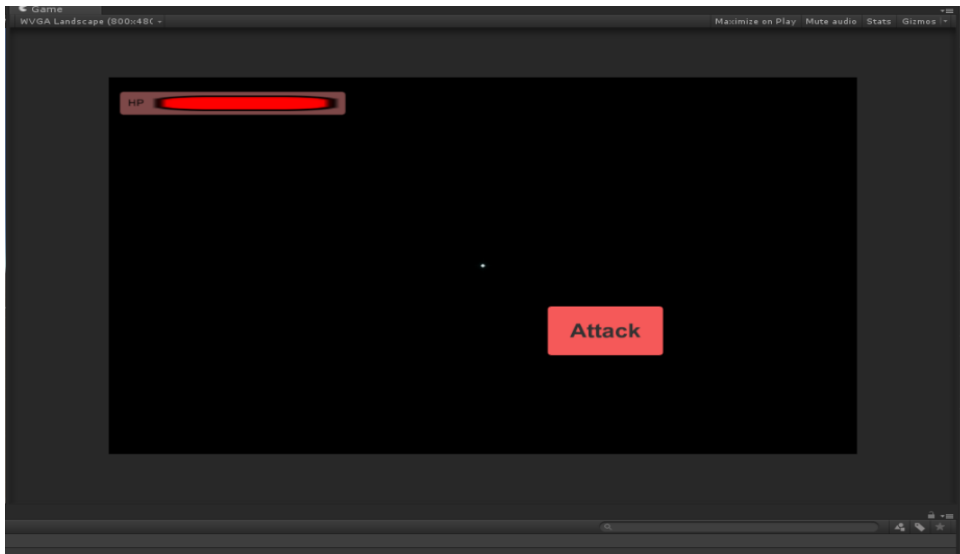


Figure 3.10.1: User interface player sees. Black screen due to the vuforia camera.

The reason canvas was use as opposed to using something simple such as OnGui is the simplicity of implementing a user interface which would scale relatively well with devices and allow orientation of the device to not hinder the UI positions. The health bar is also more visual as opposed to just numerical figures. The method of doing so is to utilize the mask and filler component. The filler element is essentially the visual filling for the health bar which would determine the max length of maximum possible health whilst the mask is the remaining health amount that would show on the player character.

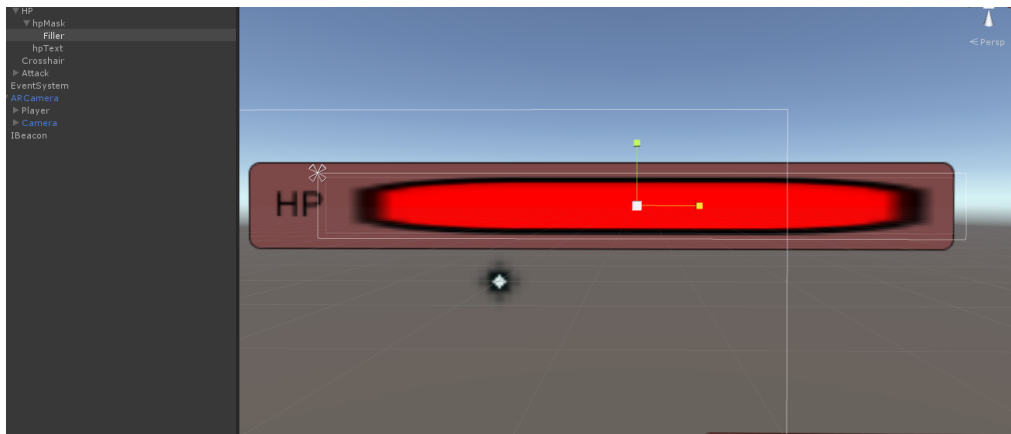


Figure 3.10.2: HP element of the canvas, with the filler and mask component as its subsequent children.

Once the mask and filler were setup, the playerScript was created in order to handle the health bar being shifted from one end to the next. This was done via HandleHP and hpMap function within the playerscript.

```
void HandleHP()
{
    filler.fillAmount = hpMap(playerHP, 0.0f, 100.0f, 0.0f, 1.0f);
}
private float hpMap(float val, float minHP, float maxHP, float retMin, float
retMax)
{
    return (val - minHP) * (retMax - retMin) / (maxHP - minHP) + retMin;
}
}
```

The main calculation function to handle the health UI was the hpMap. The function takes in the val value, which is player health, minHP which is the minimum possible health, maxHP, the numerical maximum health stats wise, and retMin and retMax. retMin and retMax are values which correspond to the current size of the health image. Since the hp mask length is between the values 0 and 1, it would only make logical sense to input the values of 0 and 1 for retMin and retMax respectively.

The calculations are then applied to each respective value in the hpMap function parameter. This allows us to then assign the fill amount to actively reflect the returned value of the remaining hpMap of the player.

The crosshair was simply an image that is placed in the direct center of the screen, and the attack button is simply the button feature found within the canvas. In order to link the button

to the attack function, it requires the function to be directly linked to the event. For reference sake, the function created is named OnClick()

```
public void OnClick()
{
    bulletInstance = Instantiate(bullet, projectileEmitter.transform.position,
Quaternion.Euler(new Vector3(0, 0, 0))) as GameObject;
    bulletInstance.GetComponent<Rigidbody>().AddRelativeForce(transform.forward *
400);
}
```

Once created and public, the canvas button settings in the hierarchy would require referencing of the specific script, followed by the specific event. This allows the user to fire a projectile when pressed. The point where the bullet is instantiated is actually an empty gameobject in front of the player character, due specifically to collision clipping issues.

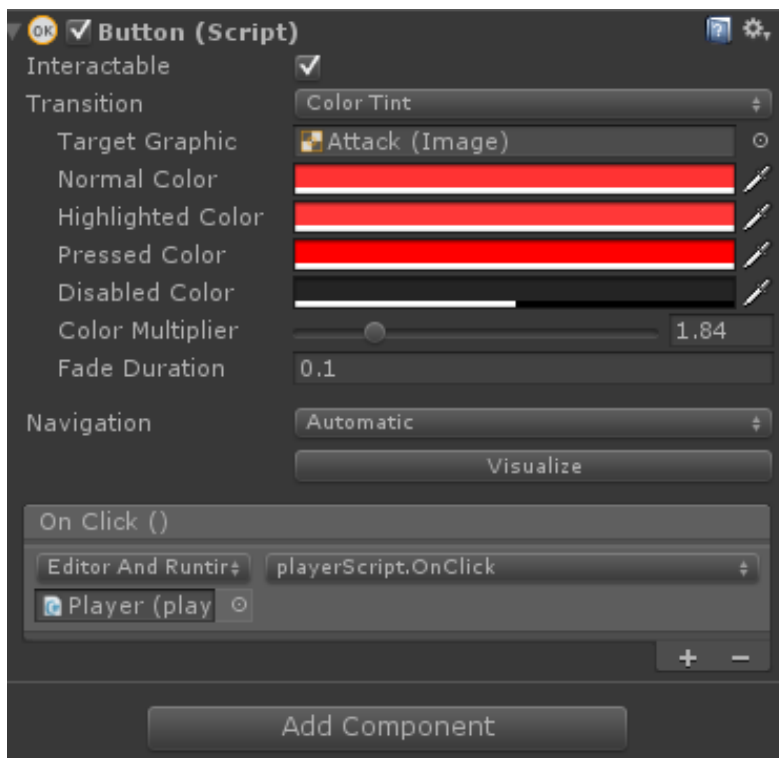


Figure 3.11: The OnClick function implemented into the button function.

The player script also contains within the update function to handle the projectiles the player spawns. As the projectiles should ideally not be instanced indefinitely onto the screen, a simple destroy function of the spawn bullet instance over a set amount of time helps resolve the issue.

```

void FixedUpdate()
{
    HandleHP();
    Destroy(bulletInstance, 3.5f);
}

```

3.2.1.3. Player Rotation and Controls

Even though Augmented Reality relies on the camera the device uses, moving it about and such actually will not impact the virtual world's camera. As such, it is necessary to incorporate a gyroscope system.

```

private void UpdateCameraBaseRotation(bool onlyHorizontal)
{
    if (onlyHorizontal)
    {
        var fw = transform.forward;
        fw.y = 0;
        if (fw == Vector3.zero)
        {
            cameraBase = Quaternion.identity;
        }
        else
        {
            cameraBase = Quaternion.FromToRotation(Vector3.forward, fw);
        }
    }
    else
    {
        cameraBase = transform.rotation;
    }
}
private static Quaternion ConvertRotation(Quaternion q)
{
    return new Quaternion(q.x, q.y, -q.z, -q.w);
}
private void ResetBaseOrientation()
{
    baseOrientationRotationFix = GetRotFix();
    baseOrientation = baseOrientationRotationFix * baseIdentity;
}

```

The excerpt shows a simple function of updating the camera rotation for the gyroscope taken from A. Dezhurko's(2013) gyroscope controls. The base theory is calculating the quaternion locations as well as potentially resetting the base rotation for the device. The problem with the code however, was the requirement of requiring a tap of the reset button in order to set the gyroscope to work and there were issues of 'camera rolling sideways' once a certain point was rotated to in the y axis.

As such, modifications were done to the code, however, the movement and such felt very unnatural and jerky at best. Thus incorporation of the FPV gyroAccel script was done, which proved to be efficient and smooth.

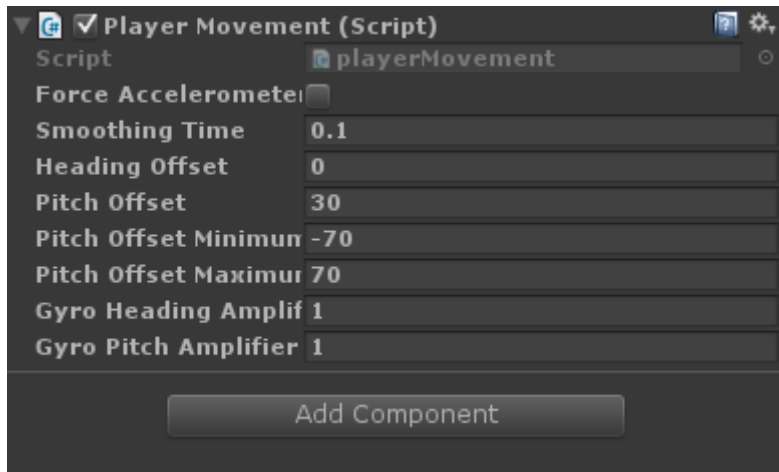


Figure 3.12: the FPV gyroAccel script incorporated into player movement script.

The script handles not only the device rotation, but also potential noise soothing, which meant any slight movement would not prompt the device to be jerky. The script also provides a gravity vector, which would help determine the x y and z orientation of the device, negating the need for calibration.

```
void UpdateGyroscopeOrientation()
{
    Quaternion gyroQuat = Input.gyro.attitude;
    Quaternion A = new Quaternion(0.5f, 0.5f, -0.5f, 0.5f);
    Quaternion B = new Quaternion(0f, 0f, 1f, 0f);
    gyroQuat = A * gyroQuat * B;
    //
    float devicePitch;
    float deviceRoll;
    GetDevicePitchAndRollFromGravityVector(out devicePitch, out deviceRoll);
}
```

The UpdateGyroscopeOrientation function is the main function that integrates the gravity vector in order to automatically determine the android devices current orientation.

```
if (!forceAccelerometer)
{
    UpdateGyroscopeOrientation();
}
else
{
    UpdateAccelerometerOrientation();
}
```

In the fixed update, we would prompt the 'if statement' in order to determine whether to toggle the accelerometer for gyroscope functions or to fix the current device's orientation.

```

void CheckHeadingAndPitchBoundaries()
{
    if (heading > 360f)
    {
        heading -= 360f;
    }
    if (heading < 0f)
    {
        heading += 360f;
    }
    //
    if (pitchOffset < pitchOffsetMinimum)
    {
        pitchOffset = pitchOffsetMinimum;
    }
    if (pitchOffset > pitchOffsetMaximum)
    {
        pitchOffset = pitchOffsetMaximum;
    }
}

```

The boundaries are equally important for the device in question, as we want full unhindered view for the player, thus allowing the boundary function to have a limit of -360 to 360.

3.2.1.4. Enemy AI and Game Logic

Once the player is thrown into the main game, the game will immediately begin, prompting enemy spawn to happen. The method of spawning is to have the enemy spawn randomly, but within a given specific distance. What this means is the enemy targets will appear within a radius from the point of interest, and between the maximum and minimum spawning zone, almost similar to a donut shape.

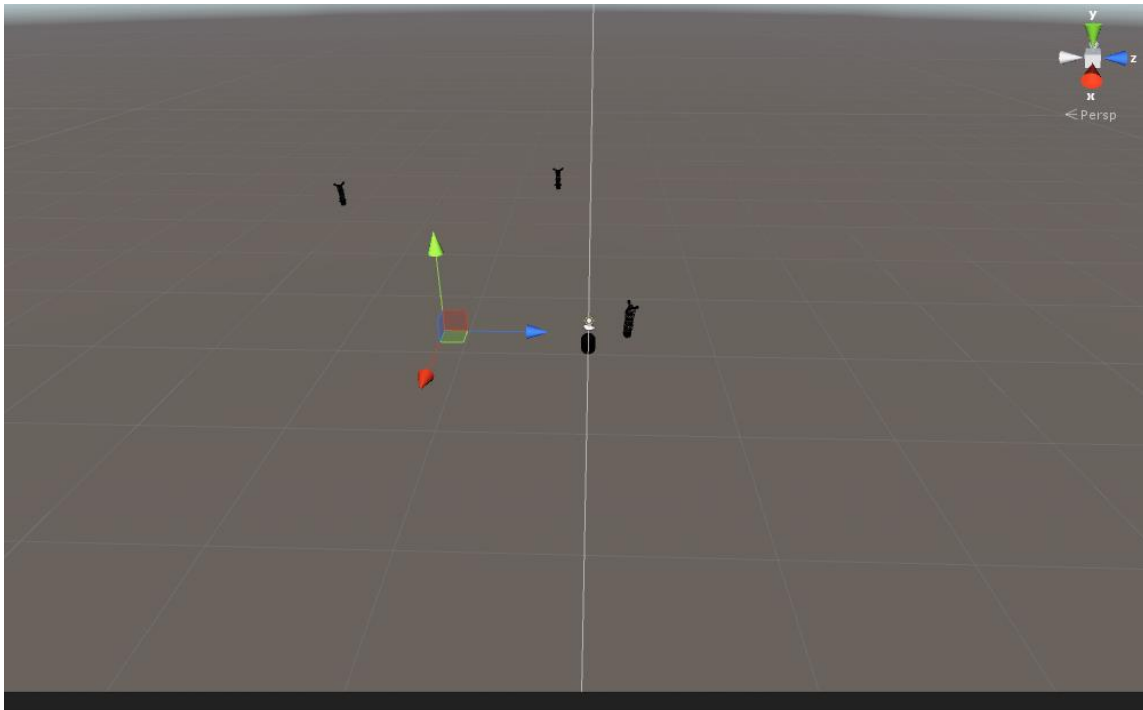


Figure 3.13: Enemies spawning around the point of interest, some spawning close, and others further away

```
public class GameManager : MonoBehaviour
{
    private float spawnRadius = 20.0f;
    private float Max = 80.0f;
    private float Min = -80.0f;
    public Transform Location;
    public GameObject Enemies;
    private Vector3 mobSpawnLocation;
    private Vector3 ran;
    private int maxAmount = 8;
    private int currentAmount;
    private bool startSpawn = true;
    private float spawnTimer = 2.0f;
    private float xPos;
    private float yPos = 0.0f;
    private float zPos = 0.0f;
    private float minEnemyDensity = 6f;
    private float minSpawnDistance = 20f;

    void Start()
    {
        currentAmount = 0;
    }
}
```

From the previous snippet of code, there is a variable known as current amount. This is purely to count enemies currently in the scene, and uses that information to determine whether to execute the code of spawning enemies or not. The update creates a list of enemies, which would store the objects with the tag “Enemy” in order for the ability to handle deletion once it dies. The list also helps to count the current amount of enemies to the current amount.

Once the enemy spawns, it is required to move towards the player character, attempting to harass and shoot projectiles at the player. Within the AI script, it also handles the logic for the each individual enemy.

```
void Start () {
    target = GameObject.FindWithTag("Player").transform;
}

// Update is called once per frame
void Update () {
    transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(target.position - transform.position), rotSpeed *
Time.deltaTime);
    cdTime -= Time.deltaTime;
    if (Vector3.Distance(transform.position,target.position) > 3.0f )
    {
        transform.position += transform.forward * moveSpeed * Time.deltaTime;
    }
    if (hp <= 0)
    {
        Destroy(this.gameObject);
    }
    if (cdTime <= 0)
    {
        bulletInstance = Instantiate(bullet, transform.position,
Quaternion.Euler(new Vector3(0, 0, 0))) as GameObject;
        bulletInstance.GetComponent<Rigidbody>().AddRelativeForce(transform.forward * 600);
        cdTime = 5.0f;
    }
}

void OnCollisionEnter(Collision collide)
{
    if (collide.gameObject.tag == "projectile")
    {
        dmg = Random.Range(5, 12);
        hp -= dmg;
    }
}
```

The script finds the player through its tag, and receives the transform. The transform is then fed into the update function for the enemy instance to begin rotating towards the target, as well as slowly move towards it. The enemy also contains a cdTime function, which prevents the enemy from spamming projectiles towards the enemy. With that in mind, the enemy also contains the function similar to the player outline previously, both to take damage on collision with any projectile or damage and remove instance projectiles after a certain time.

3.2.1.5. Beacons

As stated in the project outline, the point of the project is to integrate beacons for use with the artefact. Before beginning, the beacons require initial setup on the estimate cloud service. The estimate beacons are currently developed for the iBeacon packets and thus, the project required the use of the iBeacons plugin for Unity.

Edit Settings

You can change beacons settings remotely. Set them here and they will be updated when you approach the beacons with the Estimote iOS App, or an app with the Estimote iOS SDK integrated. [Learn more about remote beacon management.](#)

- Device
 - iBeacon
 - Name: South
 - Geo Location: [Dropdown]
 - Tags: [Dropdown]
 - Basic Battery Saving (?): Off
 - Smart Power Saving: Off
 - Motion Only Broadcasting: Off
 - Flip to Sleep: Off

Cancel Save Changes

Edit Settings

You can change beacons settings remotely. Set them here and they will be updated when you approach the beacons with the Estimote iOS App, or an app with the Estimote iOS SDK integrated. [Learn more about remote beacon management.](#)

- Device
 - iBeacon
 - Proximity UUID: 5177D014-A3C3-3727-0445-50ADC0FEDF6C
 - Major (1 to 65535): 1
 - Minor (1 to 65535): 2
 - Secure UUID (?): Off
 - Custom Advertising Interval: On
 - Interval (100 to 10000 ms): 950
 - Broadcasting Power (dBm): -12

Cancel Save Changes

Figure 3.14: The one beacon is set to South, with a UUID and major/minor values set.

Once the setup has finished, and the iBeacons plugin is present in Unity, the iBeacon setup can begin. The Unity tools bar will add an iBeacon option, asking to run the iBeacon on app Start or runtime.

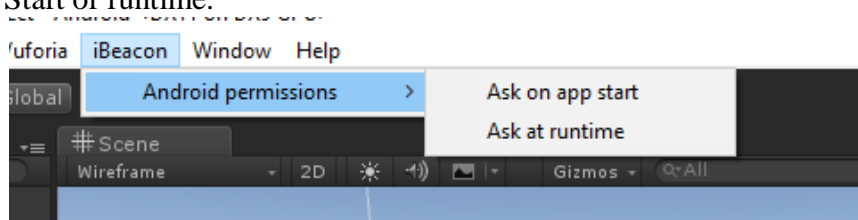


Figure 3.14: The options for iBeacon on android devices.

Once done, Unity will allow the developer to decide whether to use the iBeacon script as a receiver or sender. Sender just allows the device itself to emit signal and act as a beacon in its stead. When the beacon gameObject is created and in the hierarchy, the object will contain 2 scripts, of which one will capture data as the receiver.

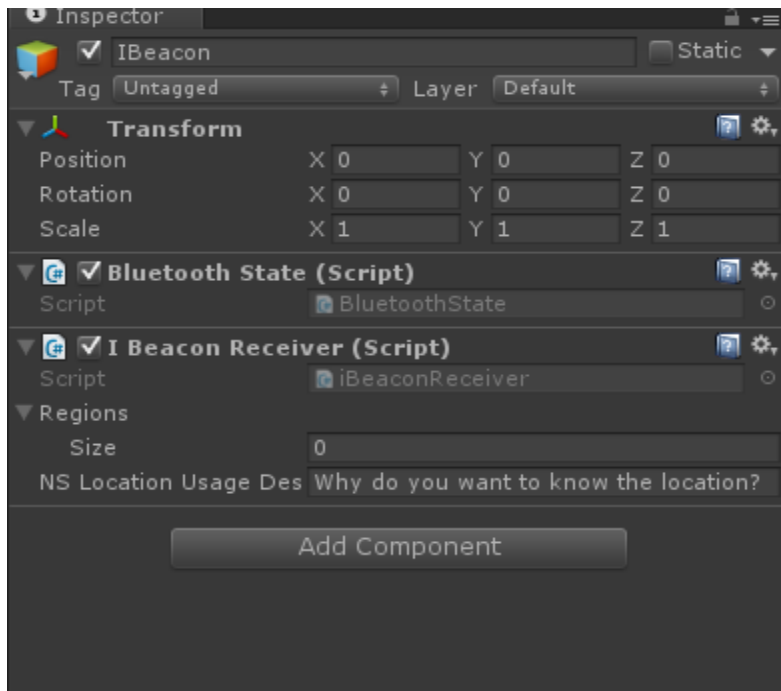


Figure 3.15: The scripts. Notice the state and receiver.

The iBeacon receiver script will then have the regions and size modifiable variable. This is essentially the size or amount of beacons the developer would like to use to track and detect. Since the project is going to triangulate the player position, utilizing 4 beacons would essentially help pinpoint to a more accurate degree.

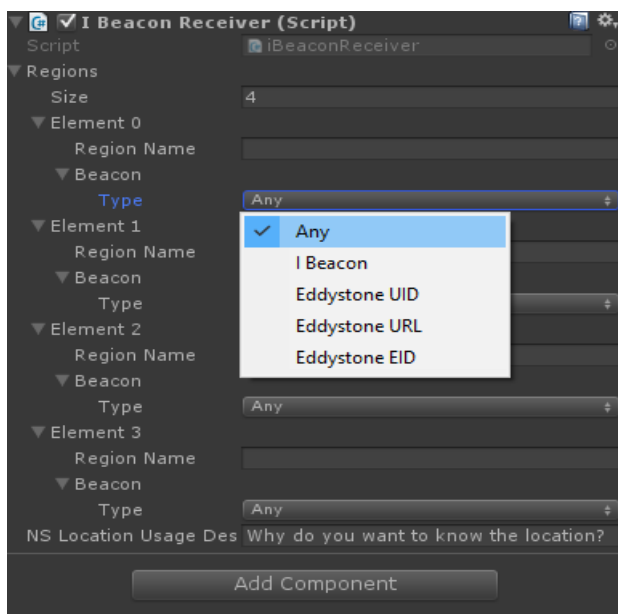


Figure 3.16: Expanded options for iBeacons

The beacon types listed are the different packets iBeacon supports. Since Estimote utilizes iBeacon packets, the only option would be to use it. Once decided, the component expands even more options listing UUID, Major, and Minor. Each of the variables listed can be found via the estimote cloud.

Region	Region Name	Beacon Type	Beacon UUID	Beacon Major	Beacon Minor
North	North	I Beacon	3D0DB270-4EF8-697D-6B92-D080DD9F8	1	1
South	South	I Beacon	5177D014-A3C3-3727-0445-50ADC0FEC	1	2
East	East	I Beacon	78595A8F-3AE5-6F8E-68A6-3BE2BE7855	1	3
West	West	I Beacon	BDFB2EAE-910C-506F-6026-C9943F0AB	1	4

NS Location Usage Des Why do you want to know the location?

Add Component

Figure 3.16: Relevant details filled in.

Once the details were correct, iBeacon should work on the android device as there is a lack of hardware to do testing on the development machines. However, this was not the case for this project, and the next part will highlight the reason.

3.3. Risks and problems encountered

Essentially, the artefact was never complete, due to the most crucial part of the project, the beacons, not working as intended, primarily due to severe lack of documentation online and development ability. This has led to contingency plans of, instead of pinpointing the player's location with beacons, to use it as a point of reference to spawn enemies. But even so, attempting to do so has proven to be unsuccessful.

3.3.1. Beacons android build

The primary issue was attempting to build the project. The iBeacon would have issues attempting to build the APK, and subsequently removing the iBeacon factor entirely on Unity allowed the project to be built with no sorts of problems whatsoever. The primary issue was the following when attempting to build:

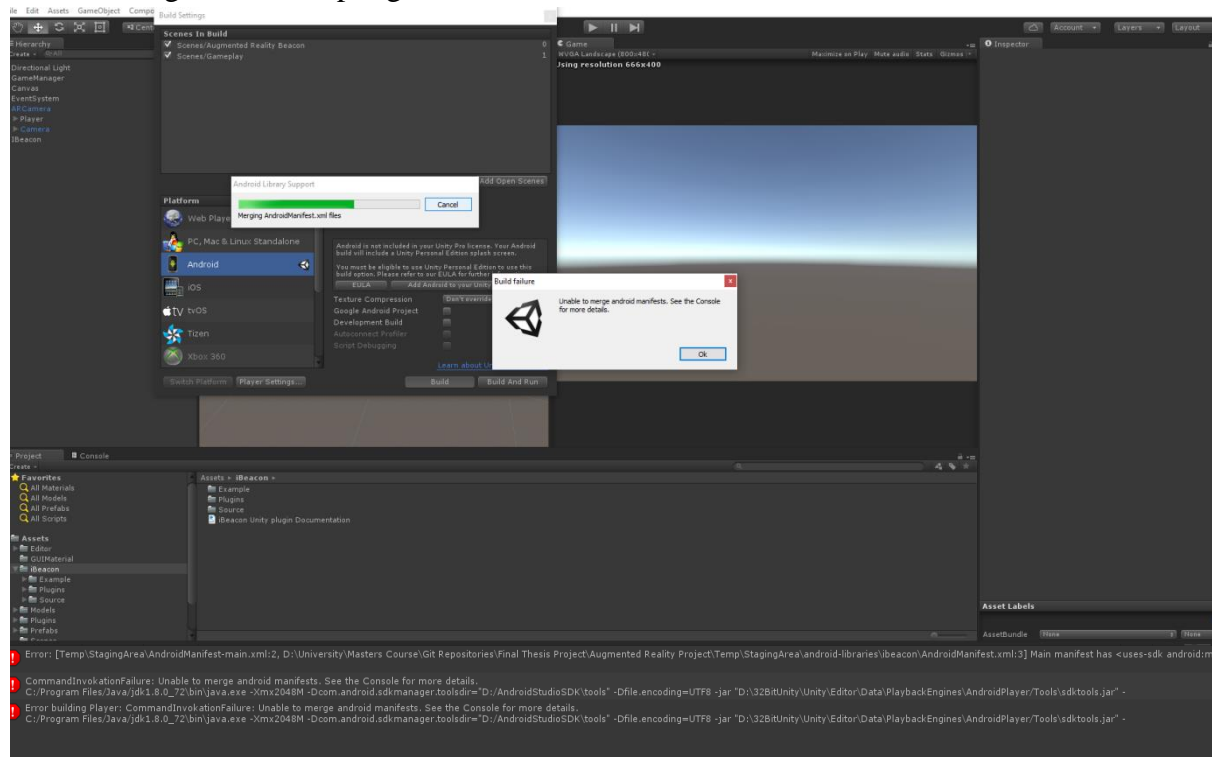


Figure 3.17: Issue about android manifest merge.

The issue is in regards to the build and actual information being fed in the android xml. Since vuforia and iBeacon requires a minimum API level 15, but even if the player settings were changed to reflect the android xml, or the other way round, the error would still persist. Even on forums, the issues were pretty widespread, with issues resolved with a variety of methods. Having tried and attempted all known methods, such as uninstalling the android SDK and reinstalling, updating the SDK, updating Unity, the problem was still persistent.

Upon researching further as well, estimote currently lacks its indoor location SDK for unity3D. The SDK provides users with the ability to essentially map out a room in order to better utilize the beacons and its full capabilities. Various online forums also highly recommended against the use of beacons to triangulate the player position ("Is it possible to

use other iBeacons for triangulation?,” 2014) specifically due to the fact that “..on the 2400 Mhz you will have a lot of reflections, multipath plus phase shift” (Naumann, 2014)



Figure 3.18: Issue about android manifest merge.

Having said that, the Vuforia part of the menu also proved to be slightly problematic. Since the point of the menu is to magnetize the virtual object towards the player, if the user were to move too close to the object, the target will automatically detect the collision with the camera, and instantly switch to the next scene.

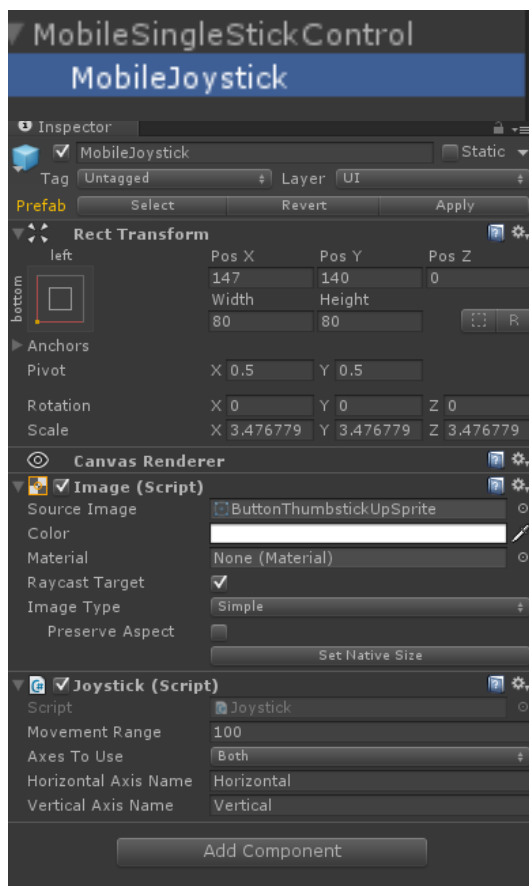
The project has been rebuilt a few times as well in order to try attempt bypass the issue, in an attempt to remove the meta data files. This method not only delayed development time and resources, but did not work either. After further research, beacons also have issues in performance, specifically the accuracy of the beacons. This meant unless the user was in a complete empty room, with no sort of radio interference or the sorts, then the beacons technology would be ideal. This however, also meant that once any disturbance was present, diminishing returns would occur on the beacons, prompting inaccuracy.

And due to that, the beacon implementation for the artefact project is completely removed, as implementation is just simply not possible within the time frame. This means that the original idea of implementing character movement based on triangulation was not possible to be implemented and also utilizing beacons as a spawn point. As such, the enemy spawn location will still be based around the player, meaning that despite the player moving, the spawn radius and distance will update accordingly.

In order to compensate, the project will utilize a virtual joystick for the user to move in the direction they choose via dragging the virtual joystick. This is done via modifying the `playerMovement` script, which would track horizontal and vertical movement.

```
float horizontal = CrossPlatformInputManager.GetAxis("Horizontal");
float vertical = CrossPlatformInputManager.GetAxis("Vertical");
CharacterController control = GetComponent<CharacterController>();
moveDirection = new Vector3(horizontal, 0.0f, vertical);
moveDirection = transform.TransformDirection(moveDirection);
moveDirection *= speed;
control.Move(moveDirection*Time.deltaTime);
```

This allows the player to move in the horizontal and vertical direction. The reason that the script utilizes `CrossPlatformInputManager` instead of the traditional `Input`, is to access the virtual joystick. The joystick in question is from the standard unity package, and simply appears similarly to how the canvas UI and buttons do.



3.19. Joystick and scripts.

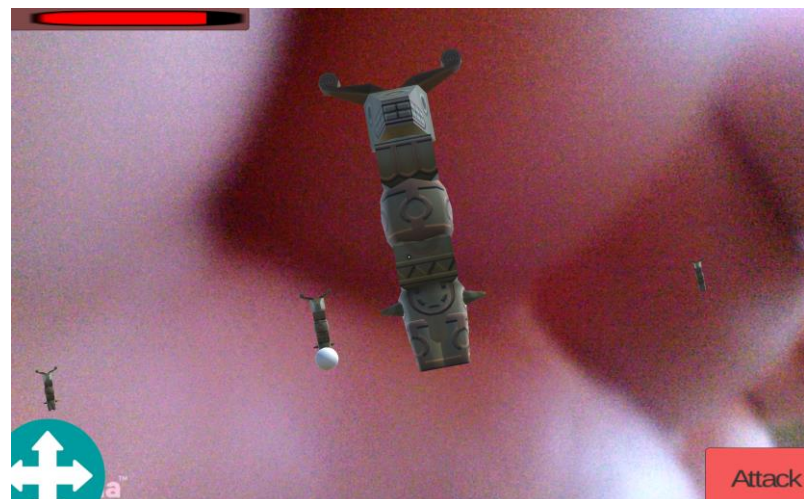


Figure 3.20: updated artefact to reflect contingency changes

Section 5: Conclusion

The project, although due to technological failures, was an interesting exploration into virtual reality and augmented reality. This is due to the fact that beacons, although not optimized for gaming due to diminishing returns of signals and distance, it provides opportunities more for casual games that does not demand accuracy. This includes egg hunt games, since the beacons can provide a proximity checker of far, close, and near.

The project could have been managed better, in the sense of abandoning potential ambitious ideas and tackled a different direction with implementation of tracking the character. The general gameplay could be better handled and instead of trying to fix a problem that was beyond control, focus more on touching up gameplay logic or add more features.

If given another opportunity to tackle this project, I would definitely attempt to do so, but essentially do something on a smaller scale and not try to bypass or explore a method of player tracking that would essentially result in the effort being for naught. Ideally, in the hopes of utilizing beacons as power up towers which may give the player buff or debuffs dependent on how close or far the player is.

This project provided me with insight on just how simple augmented reality is to implement, and the wide range of possible features a developer could do to it. With the accessibility for novice developers to more experienced ones, potential of technology growth is undeniably high. Due to this, I personally feel that maybe Peter Jackson's prediction of augmented Reality being bigger than mobile phones and virtual reality a dream come true in the future.

Section 6: Bibliography

Mason, M. (2013, December 19). *Demographic breakdown of mobile Gamers*. Retrieved July 12, 2016, from <http://developers.magmic.com/demographic-breakdown-casual-mid-core-hard-core-mobile-gamers/#prettyPhoto>

Graft, K. (2015, March 2). *500 games launched per day on iOS last year (and other digital sales facts)*. Retrieved July 12, 2016, from http://www.gamasutra.com/view/news/237811/500_games_launched_per_day_on_iOS_last_year_and_other_digital_sales_facts.php

Jarvis, M. (2016, April 20). *Peter Jackson: "AR will be bigger than mobile in 10 years."* Retrieved July 13, 2016, from Develop-Online, <http://www.develop-online.net/news/peter-jackson-ar-will-be-bigger-than-mobile-in-10-years/0219439>

Lemon, M. (2014, August 18). *Oculus Doesn't Want Competitors To Damage VR Experience*. Retrieved July 13, 2016, from Escapist, <http://www.escapistmagazine.com/news/view/136912-Oculus-Doesnt-Want-Competitors-To-Mess-Up-Virtual-Reality>

Albanesius, C. (2012, April 4). *Google "project glass" replaces the Smartphone with glasses*. Retrieved July 20, 2016, from Consumer Electronics Reviews, Ratings & Comparisons, <http://uk.pcmag.com/consumer-electronics-reviews-ratings/64927/news/google-project-glass-replaces-the-smartphone-with-glasses>

Curtis, S. (2015, January 15). Google halts sales of Google glass. *The Telegraph*. Retrieved August 3, 2016, from <http://www.telegraph.co.uk/technology/google/11349334/Google-halts-sales-of-Google-Glass.html>

Kalinauckas, A. (2015, February 16). *7 problems with: Google glass - E & T magazine*. Retrieved August 10, 2016, from <http://eandt.theiet.org/magazine/2015/02/google-glass.cfm>

Bogle, A. (2016, July 11). *How the gurus behind Google earth created "Pokémon go."* Retrieved August 10, 2016, from Mashable, <http://mashable.com/2016/07/10/john-hanke-pokemon-go/#F2WD1LMonmql>

Raspopina, S. (2016, July 12). Why is Russia so distrustful of Pokémon go? *The Guardian*. Retrieved from <https://www.theguardian.com/world/2016/aug/10/why-is-russia-so-distrustful-of-pokemon-go>

Willis, C. P. (2016, July 28). Pokénomics: How Pokémon go can turn success into dollars. *Newsweek*. Retrieved August 15, from <http://europe.newsweek.com/pokenomics-explaining-success-pokemon-go-phenomenon-financial-nintendo-484951?rm=eu>

Inc, P. (2016). *Vuforia 6 platform unveiled*. Retrieved August 24, 2016, from <https://www.vuforia.com/>

Estimote. *Estimote*. Retrieved August 24, 2016, from <http://estimote.com/>

Is it possible to use other iBeacons for triangulation? (2014). Retrieved September 1, 2016, from <https://www.quora.com/Is-it-possible-to-use-other-iBeacons-for-triangulation>