Dissertation Type: Enterprise

DEPARTMENT OF COMPUTER SCIENCE

A Mixed Reality Shot Assistant for Pool and Snooker

Finn Alexander Wilkinson

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 20th May, 2021

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

This project fits within the scope of ethics application 97842, as reviewed by my supervisor, Andrew Calway.

Finn Alexander Wilkinson, Thursday 20th May, 2021

Contents

List of Figures

List of Tables

List of Listings

Executive Summary

With a notoriously high learning curve, cue sports can be one of the most time consuming activities to improve at. Given that more and more people are becoming interested in professional snooker, it would be no surprise for the uptake in competitive cue sports play to increase. With this increased uptake comes practice, and lots of it. Currently, there are two main options for improving your accuracy in cue sports. Either train alone with standard drills, or hire a coach to give you feedback whilst doing these same drills. I propose to utilise Mixed Reality technology, specifically the Microsoft HoloLens, to aid players in their shooting ability so that they can improve at a quicker rate than current training methods allow. After interviewing some experienced player's for their ideas, I created an interactive application using the Unity development suite that displays the estimated trajectory of a cue ball and any ball that it would collide with. By displaying the trajectory path, a user can improve their aiming abilities by learning the correct game mechanics, as well as learning about how the cue ball reacts to different collision types.

The main functionality of the system comes from the fully custom post-collision path estimation system. Through the use of geometry and approximating the real world physics surrounding cue sports, I was able to successfully estimate the post-collision direction of a cue ball after it collided with a table cushion or another ball. Different methods were used for each collision type, but the underlying collision detection system was the same. Similar to ray tracing technology, I fired a collection of rays into my application scene from the cue ball to detect which object would be hit. This use of multiple rays allowed me to distinguish which object would be hit first if a number of balls were present in front of the cue ball, as well as accurately predict thin-cut shots between the cue ball and another ball. The resulting system was proven through extensive evaluation to be very accurate for ball-to-ball collisions.

Evaluation of the project concluded that whilst my application is successful in improving the accuracy of beginner players by up to 300%, there is some work to do in order to create a fully functional version. User feedback showed that some of the pre-shot set-up was not the easiest to achieve, nor was accurately aiming a shot in some instances, but it did show that I created an intuitive application to navigate. My evaluation also showcased the stability and reliability of the Microsoft HoloLens itself, with all test users expressing their interest and excitement in the technology. Overall, I was able to create an application that was useful to beginner cue sport players by lowing the entry learning curve into the sport, and that clearly outlined the potential and capabilities of mixed reality technology.

Summary of Achievements

- I spent approximately 30 hours researching into current cue sports training methods via interviews and online research, current technology trends, and the use of technology within cue sports.
- I spent over 200 hours iteratively implementing a bespoke application for the Microsoft HoloLens to aid cue sports players in their training and playing of the sport.
- I wrote 9 original C# scripts, amassing to approximately 1,200 lines of source code.
- I created an bespoke ball path estimation sub-system which accurately models the real world dynamics of ball-to-ball collisions.
- I performed structured user testing in order to assess the benefits the application provides to a user whilst they are playing a cue sport, as well as to gain critical feedback on the user interface design and possible improvements that could be made to the application.

Supporting Technologies

- I used a Microsoft HoloLens Gen. 1 head mounted display supplied by the Department to deploy my application to, for development and testing purposes.
  https://docs.microsoft.com/en-us/hololens/hololens1-hardware
- I used the Microsoft Mixed Reality Tool Kit to aid in development of my Microsoft HoloLens application (as recommended by Microsoft). Mainly, it aided my development by providing C# scripts allowing for user interaction within the application, as well as some pre-fabricated user interface objects such as buttons that are set up for HoloLens application use.
  https://github.com/microsoft/MixedRealityToolkit-Unity
- I used Unity Studios with Microsoft Visual Studio 2019 to develop and deploy applications to the Microsoft HoloLens. This is due to its simple user interface as well as integrated support and deployment pipeline for the Microsoft HoloLens.
  https://unity.com/
  https://visualstudio.microsoft.com/vs/
- Auto Desk Maya 2019 was used in order to create in 3D models used in the application.
  https://www.autodesk.co.uk/products/maya/overview
- The online application Chalky Sticks Pad was used to design an assortment of shots to be used when evaluating the finished application.
  https://pad.chalkysticks.com/

Notation and Acronyms

HoloLens	:	Microsoft HoloLens Generation 1
Holo-Object	:	A Holographic Object
SDK	:	Software Development Kit
AR	:	Augmented Reality
VR	:	Virtual Reality
MR	:	Mixed Reality
MRTK	:	Mixed Reality Tool Kit
UI	:	User Interface
HMD	:	Head Mounted Display
2D	:	Two Dimensional
3D	:	Three Dimensional
API	:	Application Programming Interface
UML	:	Unified Modeling Language
UWP	:	Universal Windows Platform

Acknowledgements

First and foremost I would like to thank my project supervisor, Dr Andrew Calway for his continual advice on project preparation, workflow methods, and general project guidance. I would also like to thank Joost Van Shaik for agreeing to meet with me and discuss projects he completed from his time in industry developing for the Microsoft HoloLens, as well as discussing and providing feedback on my project execution ideas and development strategies. Finally, I would like to thank everyone who gave up their time to complete a survey, give their feedback, or agree to partake in my user testing - all of which have been critical in making this project possible.

Chapter 1

Introduction and Motivation

1.1 Project Overview

Training methods for cue sports have more or less stagnated and have not embraced the technological advancements available in today's society. With the sport being steeped in tradition, this may not be surprising, however, snooker has been seen to add technology to enhance viewing experiences[16] as well as systems to help the referees re-setting balls in major tournaments[39]. This shows us that the game can be enhanced with technology and that people are willing to accept change, as long as its used in the right way. Despite technology being used in professional tournaments, utilising technology in cue sports training has not seen such advancements. Presently, the main way of improving at any cue sport involves repetition of pre-set drills concentrating on potting, positional play, and in-game tactics. An example of common drill that one may perform can be seen in Figure 1.1. Private lessons are also available, but these will consist of the same sort of drills being repeated again and again, and are often at an expensive rate. It seems logical that the time taken to improve one's accuracy in potting balls or completing difficult positional shots could be decreased, either by the use of visual aids, or real-time analysis and feedback.

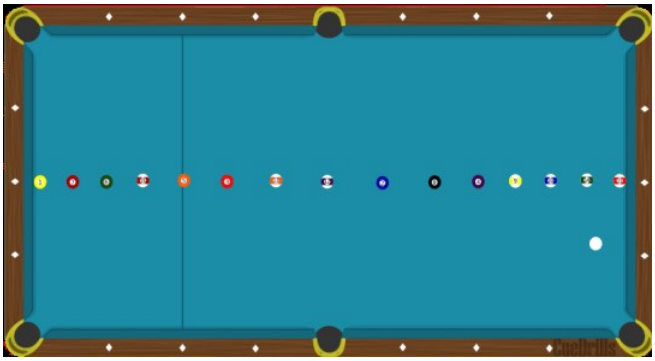


Figure 1.1: An example of a training drill performed to improve accuracy in pool or snooker. The aim is to pot every ball in the central line from where the cue ball ends up after the previous shot. Image from CueDrills.com[10].

One logical option to provide visual feedback to cue sports training seems to be Mixed Reality (MR) technology. Closely related to Augmented Reality (AR) and Virtual Reality (VR), mixed reality involves displaying augmented, or holographic, content into the world around us for a user to interact with using intuitive hand gestures. With the aid of devices such as Head Mounted Displays (HMDs), useful applications can be created for a wide array of activities, including visualising engineering and architectural designs, mapping interior design ideas onto a space in real time, and of course gaming. I intend to use one of the newest MR devices, the Microsoft HoloLens, in order to provide live shot assistance to the user in the hopes of improving their potting accuracy as a result. By showing the user exactly where to hit the cue ball, the path that the cue ball will travel along, and how the cue ball or any object balls will react to collisions, it should make it much easier for a user to pot balls and win games, especially beginners. With an active and large development community, inbuilt support to develop in Unity Studios, as well as being one of the first on the market, the Microsoft HoloLens seems like a natural choice for such an application.

Additionally, the Microsoft HoloLens has been released with an included Software Development Kit, the Mixed Reality Tool Kit, that allows developers to easily add recognised hand gesture controls into an application, removing the need for custom external controller support therefore making an application more natural to use.

Throughout the rest of this thesis, reference to *HoloLens* can be assumed to mean the *Microsoft HoloLens Generation 1*, and any reference to *Pool* or *Snooker* will be synonymous with cue sports in general.

1.2 Motivation

Although emerging over 40 years ago[3], mixed reality technology has come along way in the last decade. This is mainly due to the advancements in the computer vision, virtual reality, and hardware spaces that have allowed for portable devices capable of performing such complex tasks. In 2020 the MR market was valued at $553 million and is predicted to rise to over $5.81 billion by 2026[14], making it set to become one of the fastest growing markets over the next five years. A major reason for the market's rapid projected growth is down to the amount of competition present, with many large and small companies currently working on MR technology and no dominant players[14]. Despite some big names being in the MR space, such as Microsoft, Dell, and HP, there are also smaller companies and start-ups that have made considerable progress, such as Magic Leap who have released a direct competitor to Microsoft's HoloLens[15], and Amber Garage who have released an affordable mixed reality kit to use with a smartphone[11]. With so much competition and devices becoming smaller, more power efficient, and cheaper, it is no surprise that mixed reality is set to become one of the new major technological trends, both commercially and in industry.

Due to the naivety of MR technology, there is a serious lack of applications being developed within the commercial space. Some industrial companies have begun to use mixed reality to enhance their research and development, such as Airbus who have been using Microsoft HoloLens' since 2017 to allow their engineers to install aircraft parts quicker and easier[2]. This serious lack of any commercial applications is due in part to the current high cost of HMDs, but, over time as more advancements are made within the MR space, the prices will fall making them readily available to the public. As such, there isn't a comparable MR application currently available on the market which provides shot assistance to cue sports players, making my application one of the first of its kind.

Cue sports have always been a popular recreational activity, with many people playing in pubs, bars, and snooker halls. However, over recent years there has been a considerable increase in the popularity of professional cue sports - especially snooker. Beginning to regain the national popularity that it had back in the 1980s, the 2020 World Snooker Championship Final captivated over 25% of the UK population, roughly 17.1 million people[44]. This was up from 11.8 million viewers for the 2019 World Snooker Championship Final[44]. Additionally, over recent years snooker has soared in popularity in China, largely down to government funding and recent successes of professional players such as Li Hang, Marco Fu, and Ding Junhui[40][43]. With this rise in popularity will come more people wanting to enter into the sport and subsequently more people partaking in training, wanting to better their skills. This will directly increase demand for new and innovative ways to train, especially for beginners who will want to improve as quickly as possible. From interviewing experienced members of the University of Bristol Pool and Snooker Club, an application such as mine was seen to be most beneficial to these new players, helping them understand the basics of the game and how to aim different shots. Furthermore, it was also indicated that such an application could be useful for any player who wanted to be able to visualise the game's mechanics better or improve their more complex game skills. From this, we can see that an application that would aid players in their shooting abilities, and that lowers the learning curve into the sport, attracts the attention of already experienced players who have played competitively for their country, showcasing the desire and interest people have in the product.

1.3 Central Challenges

With any development project there are some key challenges which must be overcome, even more so for an emerging technology such as MR where such development could be the first of its kind. Human-computer interaction has always been one of the most important aspects of any software project, as without an intuitive and easy to use interface, the final product will be unusable. In order to interact with the HoloLens the user can perform two gestures, called Bloom and Air-Tap (seen in Figure 1.2), as well as move a cursor by physically moving their head which allows them to select objects, called Gaze. Due to the limited number of gestures available, it is critical that a clean and easy to use interface is created. This will include being able to line up a shot correctly, add and replace holographic objects as necessary, show and hide different application features, and being able to use every feature the application provides. Failure to do so will result in an application that provides little to no benefit to the user, as they wont be able to actually use it, despite how sophisticated the rest of the system is.

(a) Bloom Gesture. (b) Air-Tap Gesture.

Figure 1.2: Shown above are the two gestures that a user can use to interact with the Microsoft HoloLens. Panel (a) shows how the bloom gesture is performed, panel (b) shows how the air-tap gesture is performed. Images from Microsoft[23].

The other major challenge will be accurately estimating and displaying the paths of the cue ball and any object ball it hits, pre-collision and post-collision. Clearly, for an application that focuses on improving in the user's aim and potting skills, it is important to make this as accurate as possible. There are only two possible collision types, ball-to-ball and ball-to-cushion (or ball-to-rail), however each has its own set of assumptions that need to be made in order to calculate the correct post collision direction. Getting these assumptions correct will be a major part of the project's success, with both theoretical knowledge and physical testing required to best calculate them. Failure to do so will result in an application that does not provide any benefit whatsoever.

One final challenge for this project will be making the application efficient enough to run smoothly on the HoloLens. With only 900MB of RAM and 114MB of video memory available to the application, coupled with a low powered 1.04GHz Intel Atom processor [42], an effort needs to be made to reduce both the complexity of the 3D models displayed and make any algorithms as efficient as possible. As the displays on the HoloLens refresh 240 times a second (showing four separate color fields for each newly rendered image, resulting in a user experience of 60 frames per second)[22], not achieving 60 frames per second would result in a sluggish application that becomes either unusable or uncomfortable to use. Additionally, keeping the computational complexity of the application low will provide the benefit of increased battery life of the HoloLens, allowing it to be used for longer training sessions.

1.4 Project Aims and Objectives

1. Explore into the capabilities of mixed reality technology and head mounted displays such as the Microsoft HoloLens.
2. To be able to show the benefits of mixed reality technology when used in real world applications, such as pool or snooker.
3. To create an accurate shooting assistant for use in cue sports that helps to improve a user's potting success.
4. To provide a tool which reduces the initial learning curve of cue sports for beginners to the sport.
5. To create an intuitive and easy to use application for the Microsoft HoloLens.

Chapter 2

Background

Within this chapter, any relevant technical details required for the execution of the project has been described. This includes some of the major supporting technologies used, any problem specific equations or algorithms needed, as well as some examples of similar projects to provide some contextual background to the problem at hand.

2.1 Previous Works

With pool and snooker being such popular sports, it is no surprise that similar projects have been completed in the past. Although many use different techniques and technology to my project's execution, such as AR, they all try and achieve the same aim of visualising shot information to assist the user in their play. Similarly to my project, Medved[17] chose to use a Microsoft HoloLens in order to display ball trajectories onto the table. However, they chose to utilise a remote server to process uploaded photos from the HoloLens' photo-video camera rather than do this directly on the HoloLens itself. Said processing was either done to determine where the user is aiming to hit the ball, or to determine the accuracy of the shot after the cue ball has been struck and balls were in motion. Whilst this method of image processing has the added benefit of providing real time updates to the application features and an intuitive aiming system, as well as being able to perform more intensive processing algorithms with the increased resources the server has, it is dependant on a connection to the remote server which could be unstable and adds additional latency. In terms of calculating the cue balls trajectories, Medved[17] assumed a friction-less and spin-less system in order to simplify calculations. However, they do display an *area of error* around the predicted trajectory path to account for calculation errors, with this also being increased or decreased with the user's selected experience level to account for human error whilst playing the shot. Unfortunately, their project could not be completed nor evaluated due to the COVID-19 outbreak, and so it is unknown how successful the project implementation was.

For their project, Alves et al.[7] created PoolLiveAid which uses AR to assist inexperienced pool players. Similarly to Medved[17], they calculate and display the trajectories of the cue ball and any object balls it collides with. But, instead of a HoloLens, Alves et al.[7] use a camera mounted above the table to continually detect any motion of the balls, the position of the cue, as well as gathering ball positions when the table is in a stationary state. A mounted projector is also used to display the calculated trajectories onto the table's surface. Implementing the system in this way has benefits over a MR approach, as there is no chance for miss-alignment and the player is not obstructed by a head mounted display. However, a camera and projector based system is far less portable and less interactive than a solution involving a HoloLens or similar. In order to reduce the noise present in the camera images, Alves et al.[7] applied a Gaussian filter to the frames used for motion detection. When detecting the table boundaries and the positions of the balls once stationary, a 5-frame time average was used on the camera images instead. Again, like Medved[17], friction and spin were disregarded when calculating the ball trajectory paths. Ball-to-rail collisions were calculated as a reflection (the incoming angle of the cue ball is the same as its outgoing angle) and ball-to-ball collisions were calculated in a similar way to my project's implementation, described in the Section 2.5.1 - noting that Alves et al.[7] do not calculate the cue ball's trajectory after collision with another ball. An approach such as this reduces the complexity of the trajectory calculations whilst, in most cases, providing accurate approximations. By tracking the cue position from the mounted camera, Alves et al.[7] can update the projected cue ball

and object ball trajectories in real time, giving the user immediate shot feedback. This makes lining up a shot easy for a user, which is especially useful for an inexperienced player. Concluding, Alves et al.[7] found their calculated trajectories to be accurate 96% of the time for ball-to-ball collisions and 77% accurate for single ball-to-rail collisions. They also had minimal issues detecting the balls, table edges, or the cue. Disappointingly, they did not provide any user test results to see how effective the system was at improving someone's shot accuracy, but the accuracy results that were provided show project was successful.

2.2 The Microsoft HoloLens

Released in 2016, the Microsoft HoloLens Generation 1 was the world's first fully untethered holographic head mounted display[25]. As seen in Figure 2.1, it has two transparent displays, one in front of each eye, where the 3D holograms can be displayed whilst also allowing the user to see their surroundings, hence the name mixed reality - we are mixing together our reality with a virtual one and experiencing them concurrently. Included in the HoloLens is a photo-video camera which allows the user to take mixed reality photos and videos, allowing them to capture what they are seeing. The tracking system that the HoloLens uses allows the user Six Degrees of Freedom (6DOF), meaning that a user can rotate their head, walk around the room, and crouch whilst still having the HoloLens accurately track where it is in the room. In comparison, most VR devices allow for Three Degrees of Freedom (3DOF), limiting them to only look left/right, up/down, and tilt their head side to side - they cannot walk around the room[28]. This is one of the major benefits of using a HoloLens, allowing for more natural and free movements around a space.

Figure 2.1: An image of the Microsoft HoloLens Generation 1. Image from Microsoft[25].

By utilising seven different depth, reflectivity, and light sensors as well as an inertial measurement unit[25], the HoloLens scans the environment around it and creates a three dimensional point map that lets it know where each surface and object is in the room. This 3D point map then allows the HoloLens to create a 3-axis spacial coordinate system for the environment, which can then be used to display the 3D holograms[18]. Each point of the point map correlates to a coordinate in the spacial coordinate system, allowing the HoloLens to keep track of where it is currently positioned in the room. This also allows the user to place holograms into the space around them, including on top of real life objects such as a table. The spacial coordinate system ensures that when a user places a hologram, the HoloLens can keep it in the exact same position, even whilst the user walks around the room, by knowing where it is placed relative to different objects or point map nodes[28].When an application is running, objects from the application are positioned in the spacial coordinate system by mapping the its coordinates from the application's 3-axis coordinate system to the spacial coordinate system created from the 3D point map[18]. The spacial coordinate system expresses its coordinates in meters, meaning objects placed 2 units apart in an application's coordinate system will appear 2 meters apart when rendered in mixed reality[18]. This allows developers to easily position content relative to one another and create appropriately sized holographic objects.

2.3 The Mixed Reality Toolkit

Released along side the HoloLens, the Mixed Reality Toolkit (MRTK) is an open-source Software Development Kit (SDK) for Unity that makes it easier for developers to create mixed reality applications. It provides a cross-platform input system, foundational MR components, and common building blocks for spacial interactions[30]. Being open-sourced, the contents of the MRTK have been driven, improved, and added to by the mixed reality development community, allowing it to be as useful and focused as possible. Some of the most useful features included are the user interface controls, hand tracking (platform specific), boundary and spacial awareness systems, a diagnostic tool, and an input system to name a few[36]. Also included is a wide range of useful interaction based building blocks such as 3D buttons, a system keyboard, interactable and manipulator scripts allowing for any object to be picked up and moved, bound controls for resizing and rotating objects, movable sliders, voice commands, and eye tracking systems[36]. These building blocks are one of the main attractions of the MRTK as they enable a developer to quickly and easily add common elements to an application, meaning that they can spend more time on their actual application content rather than on the application's user interface.

Mentioned above, the cross-platform input system is the other main attraction and time saver for MR developers using the MRTK. Supporting a wide range of platforms including the Microsoft HoloLens, HTC Vive, and Oculus Rift, the input system allows you to define actions like select or menu, recognise speech, as well as recognise inputs from a wide range of input devices such as game controllers or even hand gestures for some platforms[27]. Additionally, pointers can be attached to input devices allowing them to query the scene and determine what object a user is aiming to interact with. An example of this would be a line pointer that casts a dotted line into the scene from the input device, allowing the user to easily aim at an object.[27]. These pointers also make it easy to create actions for the user to interact with objects in the scene by passing events from the controller through to the object via the pointer.

2.4 Unity Development Platform

Whilst a developer can use C++ and DirectX to create applications for the HoloLens, the recommended (and much easier) way to do so is to use the Unity development platform. With integrated support for HoloLens app development[52], as well as a wide array of inbuilt sub systems including a physics engine, Unity makes the development process more intuitive, interactive, and natural. Additionally, for the HoloLens there is the ability for developers to test and interact with their application in the Unity editor[54]. This reduces development time as any new features can be tested quickly rather than building the application and deploying it to the HoloLens every time. A preview of this emulation environment is seen in Figure 2.2

An application in Unity is composed of one or more scenes. Like a movie, multiple scenes can be used in larger applications to differentiate different aspects of the program, such as different sections of a map. However, it is also common to only use a single scene for applications with little background changes. Within each scene, a developer can easily add a new 3D object by dragging and dropping it into the Hierarchy panel, seen on the left hand side of Figure 2.2, which will then make it appear in the scene editor window. Each of these objects can also be assigned as a child of another object, grouping them together and applying the transformations applied to the parent onto the child as well. The editor window allows the developer to move, rotate, and scale objects into the desired position visually instead of by planning it out in their head or hard coding it, making for a more intuitive development process. Each object can also be assigned a number of components, which allows the developer to define different properties or actions with a variety of modules. Some of these include textures to define how the model looks, custom scripts to define how the object moves or changes with certain events, or physics modules like a rigidbody to make the object conform to gravity and other forces just to name a few. From this, we can see that Unity saves the developer many lines of code, as well as time, as much of the application development is done visually or by hand rather than by hard coding everything. This allows for easy testing, quick feature changes, and a more intuitive development style.

As mentioned above, the Unity development platform has an inbuilt physics engine which is the main attraction for many. With 3D and 2D options available, a developer can quickly create realistic reactions or object movement to happen at run time without having to hard code any events by applying momentum to an object[49]. The developer can also select which objects should interact with each other,

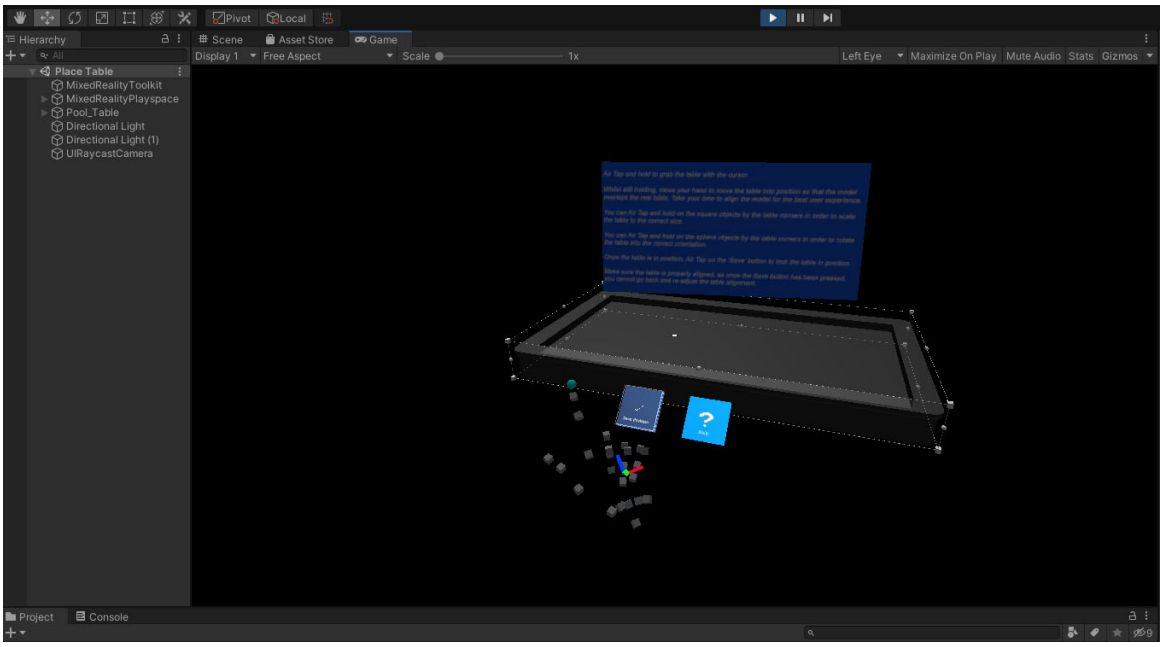


Figure 2.2: A preview of Unity's inbuilt HoloLens emulation environment allowing for application testing without deployment.

meaning that only certain objects can collide with each other - i.e. a cube can pass through water instead of bouncing off of it. The core components of the 3D physics engine are the following :

- **Rigidbodies -** Allows the attached game object to be subject to forces such as gravity, drag, or rebound forces from collisions with other game objects[50].
- **Colliders -** Defines the shape of a game object for the purposes of physical collisions. A collider is an invisible, rough approximation of the game object's shape. An inbuilt trigger system is constantly checking for any collisions, and if one occurs a method can be invoked in the game objects who have just collided[46].
- **Joints -** Connects a Rigidbody to another Rigidbody or a fixed point in space. They apply forces that move rigid bodies and limit said movement. Joint types include Fixed, Hinge, and Spring[47].

Using C#, scripting is another huge part of developing in Unity and is similar to object-oriented programming. Each script is its own class that allows a developer to assign attributes and define what a game object does. Whether that be how they respond to user inputs, arrange application events, create graphical effects, control physical behaviour of objects, implement a custom artificial intelligence for a game character, or even to listen for certain functions in another game object's script to be invoked[51]. There are two main functions in every Unity script - `Start()` and `Update()`, both of which are optional but almost always used. `Start()` is invoked once when a game object is first created or *activated*. Usually this function is used to define some private parameters, initialise listener functions, and set an object into motion. `Update()` is invoked once every frame, so is usually used to continually rotate or move objects, or to check for certain conditions like if the objects health parameter has fallen below a certain value. By combining this traditional way of development with the intuitiveness provided by the Unity editor, the developer has full control over anything they wish to create and has many different ways that they can achieve their goal.

2.5 Sphere Collision Mechanics

Within all cue sport variations, there are only two types of collisions that can occur. The specifics for each of these collision types is detailed below, with only the information relevant to the application included. For the purposes of the application, I will only consider 'standard' shots whereby the player strikes through the centre of the cue ball, not applying any side or bottom spin.

2.5.1 Ball-To-Ball Collisions

First, we have collisions between two balls. Although we are working in 3D when it comes to the graphics and implementation of the system, we can simplify all of our calculations by working in 2D. We can do this due to the fact that all the balls are identical in size and upon the same horizontal plane, meaning the y-axis components will all be the same regardless of where the ball is on said plane.

The collision between two billiard balls is almost perfectly elastic. This means that the kinetic energy of the system is conserved after impact. The two main properties which affect kinetic energy conservation between two colliding objects is the friction between the balls and the coefficient of restitution.

The Coefficient of Restitution

Also denoted by e, the coefficient of restitution is the ratio of the final to initial relative velocity between two objects after they collide. It normally ranges from 0 to 1, where 1 would be a perfectly elastic collision[56].

$$\text{Coefficient of Restitution } (e) = \frac{|V_2 - V_1|}{|U_2 - U_1|}$$

Where V_i is the velocity of Ball i after the collision, and U_i is the velocity of Ball i prior to the collision.

With an approximate coefficient of friction between the balls of $\mu = 0.05$, and an approximate coefficient of restitution between the balls of $e = 0.95$ [4], very little energy lost through friction or is absorbed or impact. We can therefore assume that these types of collisions are perfectly elastic[38] and still achieve accurate approximations of post collision trajectories. We will also safely assume that both cue balls and object balls have the same mass, meaning this too can be omitted from any calculations.

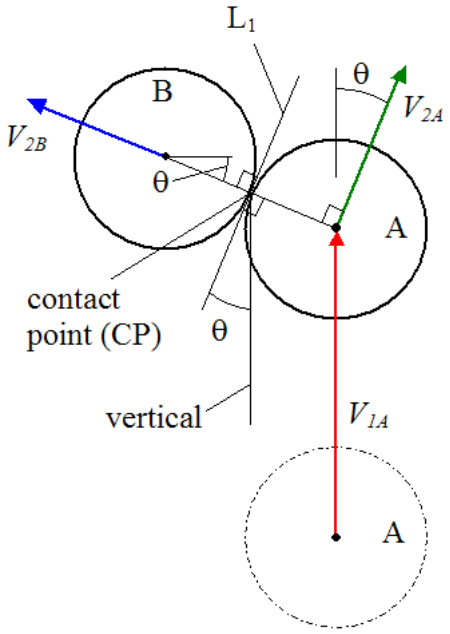


Figure 2.3: The direction vectors associated with each ball when a ball in motion, A, collides with a stationary ball, B. Image from Normani[38].

Figure 2.3 shows a ball in motion, A, with initial velocity V_{1A} colliding with a stationary ball, B. After the collision, ball B has velocity V_{2B}, whilst ball A has velocity V_{2A}. We can see that vector V_{2B} points in the same direction as a straight line drawn from the centre of ball A through to the centre of ball B at the exact time of collision. We can also see that vector V_{2A} is perpendicular to vector V_{2B} [38]. Due to the nature of my application, we do not need to worry about the speed of either ball post collision, as we are only interested in approximating their direction, not how far they will travel. Therefore, assuming

that the centre points of both balls are known at the time of collision we can calculate their post collision directions.

Despite the above being correct for calculating the object ball's post collision direction, it is only true for the cue ball when it is played as a *Stun Shot* - where a ball has no spin in any direction. In reality however, most non-technical shots will naturally apply top-spin onto the cue ball causing it not to travel in the direction of V_{2A}. Therefore, another way of estimating the angle for the cue ball post-collision will be required. Alciatore[6] has calculated some approximations that incorporate the cue ball's top-spin into the equation whilst still not needing to worry about friction or speed. Figure 2.4 shows Alciatore's 30° rule whereby a rolling cue ball will deflect from an object ball by approximately 30° from it previous direction if the collision is between a $\frac{1}{4}$-ball and a $\frac{3}{4}$-ball hit. Diagrams for $\frac{1}{4}$-ball and a $\frac{3}{4}$-ball hit are also found in Figure 2.4.

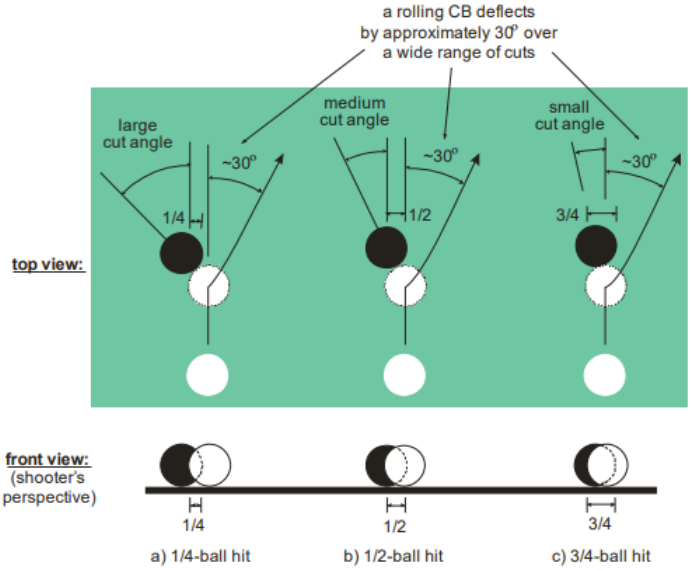


Figure 2.4: A visual representation of Alciatore's 30° rule, as well as diagrams of a $\frac{1}{4}$-ball, $\frac{1}{2}$-ball, and $\frac{3}{4}$-ball hit. Image from Alciatore[6].

For shots that are thinner (or less) than a $\frac{1}{4}$-ball hit there is Alciatore's 70% rule[6], shown in Figure 2.5. Here, a rolling cue ball will deflect about 70% of the angle between the cue ball's pre-collision direction and the tangent to the object ball's post-collision direction. Put more simply, from Figure 2.3 we would take 70% of angle θ as our cue ball deflection angle.

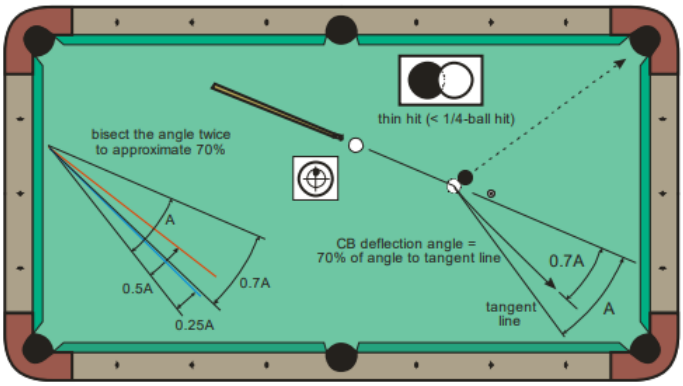


Figure 2.5: A visual representation of Alciatore's 70% rule, as well as a diagram of a less-than $\frac{1}{4}$-ball hit (or a thin-cut). Image from Alciatore[6].

Finally, for shots thicker (or greater) than a $\frac{3}{4}$-ball hit there is Alciatore's Full Hit 3x rule[6]. This is simply approximating a rolling cue ball's deflection angle off of an object ball to be 3 times the object ball's post collision angle. This is shown in Figure 2.6. Although these three approximations do not give us the cue ball's exact post collision trajectories for ball-to-ball collisions, it does allow for a much simpler algorithm that doesn't incorporate speed, spin, or friction. This will make it much easier to implement and less resource intensive to run, whilst giving us very close approximations to the cue ball's actual post collision direction.

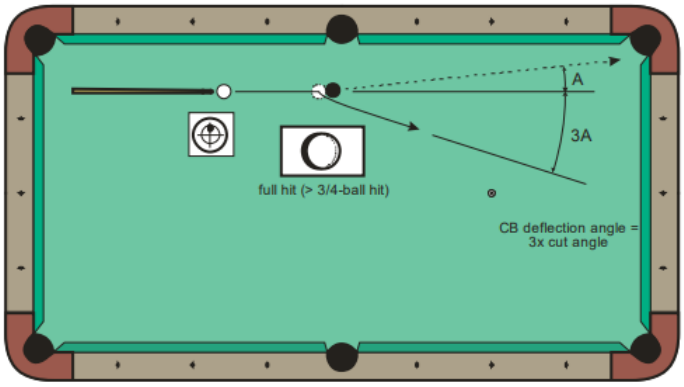


Figure 2.6: A visual representation of Alciatore's Full Hit 3x approximation, as well as a diagram of a greater-than $\frac{3}{4}$-ball hit (or a full-hit). Image from Alciatore[6].

2.5.2 Ball-To-Rail Collisions

The other type of collision we must consider is between a ball and the rail (or cushion) of the table. Unlike ball-to-ball collisions, ball-to-rail collisions are not (and cannot) be considered elastic. Typically, the coefficient of restitution between a pool ball and the rail of a table is between 0.6 and 0.9[4], making the collision very much not elastic. We can however assume both the ball and cloth to be smooth surfaces due to a very low level of rolling resistance between them - about 0.005 to 0.015[4]. Figure 2.7 shows a vector depiction of a ball-to-rail collision, with u being the cue ball's initial velocity and v being the cue ball's post collision velocity. Angles α and β are the incoming and exiting angles between the cue ball's incoming and exiting direction vectors and the rail respectively. For my applications purposes I am interested in finding out the angle β.

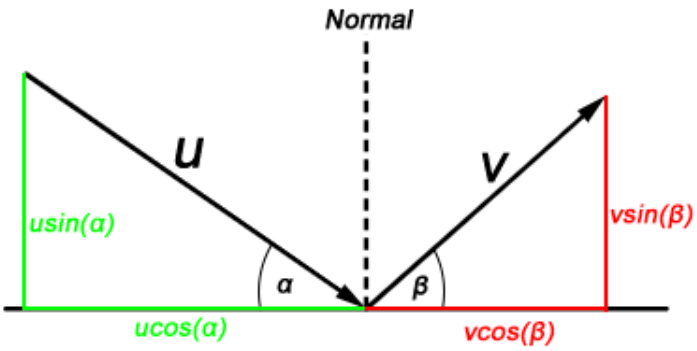


Figure 2.7: The direction vectors associated with a ball colliding with the rail of the table. The vector u depicts the ball's initial velocity, and vector v depicts the ball's velocity post-collision.

Because both the ball and rail are assumed smooth, the reaction between the two will act along the normal at the point of contact. This means that the impulse on the ball will act in the same direction -

perpendicular to the rail through the centre of the ball[12]. Consequently, the component of velocity of the ball parallel to the rail will be unchanged after the collision and so, from Figure 2.7, we can equate $v\cos\beta = u\cos\alpha$[12]. For the parallel component of v we can use Newton's Law of Restitution to equate $v\sin\beta = eu\sin\alpha$ where e is the coefficient of restitution between the rail and the ball[12]. Finally, to calculate angle β we can do the following :

$$\tan\beta = \frac{v\sin\beta}{v\cos\beta} \qquad (1)$$

$$\tan\beta = \frac{eu\sin\alpha}{u\cos\alpha} \qquad (2)$$

$$\beta = \arctan\frac{eu\sin\alpha}{u\cos\alpha} \qquad (3)$$

Given that we know e, u, and α, we can easily calculate β and therefore determine the direction of the cue ball after it collides with the table rail.

Chapter 3

Project Execution

3.1 Application Design

Before starting to create my shot assistant application, I first planned what features I wanted to include and how I was going to develop the application. In terms of development, this was an easy choice. Whilst I could have developed the application from the ground up in C++ and DirectX, I chose to use Unity given its inbuilt development support for the HoloLens, its advanced physics engine and scripting mechanism, as well as the extensive online tutorials and documentation available. It was the obvious choice which would save me a lot of time as I wouldn't have to worry about small details, or how things were going to be rendered - Unity provided this for me. Additionally, using Unity allowed me to periodically test what I had implemented in the Unity editor without having to deploy my application to the HoloLens, making the development cycle quicker and easier.

To be able to deploy an application built in Unity to the HoloLens, Microsoft Visual Studio is also required. I ended up using Unity version 2019.4.19f1 and Visual Studio 2019 as these were the latest versions at the time and were recommended by Microsoft on their get started page[29]. A GitHub repository was created for the project to provide proper version control. In terms of development style, I decided to follow the agile philosophy. Unlike a waterfall development style, an agile style will provide me with the flexibility to alter features more easily, test each new feature as it is implemented, and implement features in an iterative fashion.

In order to give an idea of what features could be useful for my application, I decided to interview some of the more experienced members and alumni of the University of Bristol's Pool and Snooker Club. The two members I interviewed have competed on a national level for the University of Bristol, having great individual and team success. Additionally, one has competed at university level for England, making them more than qualified to suggest features that would help a player's training. The questions I asked and the interviewees responses can be found in Appendix C. Both responders said that when they train they focus on potting, positional shots, break building and complex/spin shots. For the first three of these training areas, displaying the path the cue ball will take, along with the path of any ball it hits, would be a great benefit to players. When asked directly what features they thought would be useful in an application such as the one I was building, one said that being able to practice thin safety shots would be a good addition, which again is achievable using the displayed trajectory paths, and the other suggested a memory feature that allowed a player to re-set balls into the exact same position in order to practice the same shot repeatedly.

Using the ideas received from these interviews, I decided on creating a shot assistant that displays the path the cue ball will take when hit in an indicated position. Said position will be indicated by a *Cue Marker* object, shaped like the tip of a cue, and will also be used by the player to align their shot by dragging it around the cue ball. The displayed path will react if it collides with a table rail or another ball, with the subsequent rebounding lines also being displayed. By placing holographic ball objects over the top of real life balls, the user will be able to re-align the balls after their shot, so that if they wanted to repeatedly practice the same shot they can clearly see where to position the real balls. These two main features were chosen as they cover most of the main training techniques that the interviewees practice, as well as their suggested features, allowing both beginner and experienced players to utilise the application.

Shown in Figure 3.1 and Figure 3.2 are the UML diagrams describing the structure of the application, as well as how the user interacts with each object.

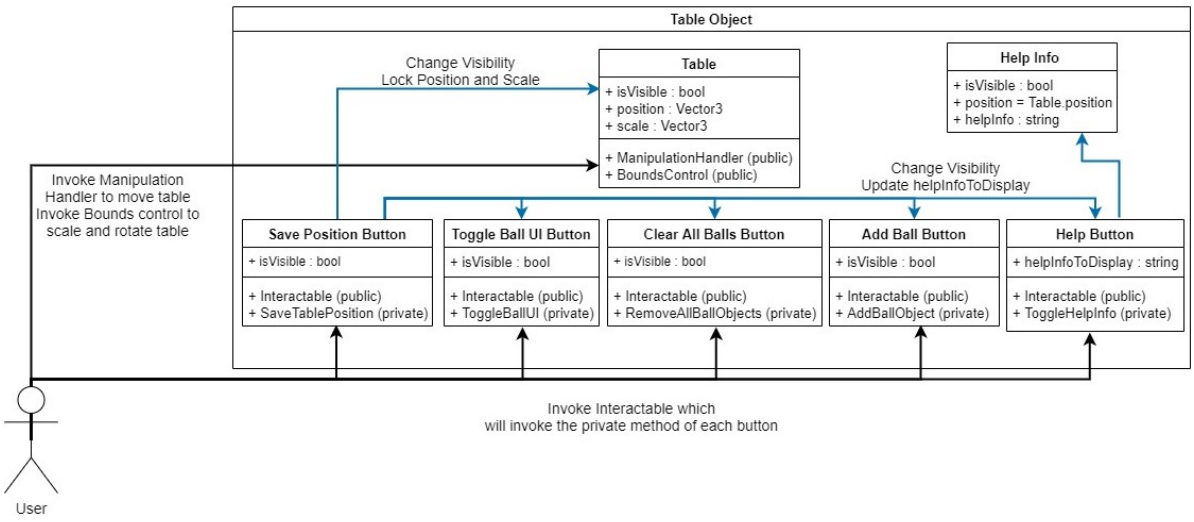


Figure 3.1: Detailed above is a UML diagram of the Table model, describing its structure and how the user interacts with it.

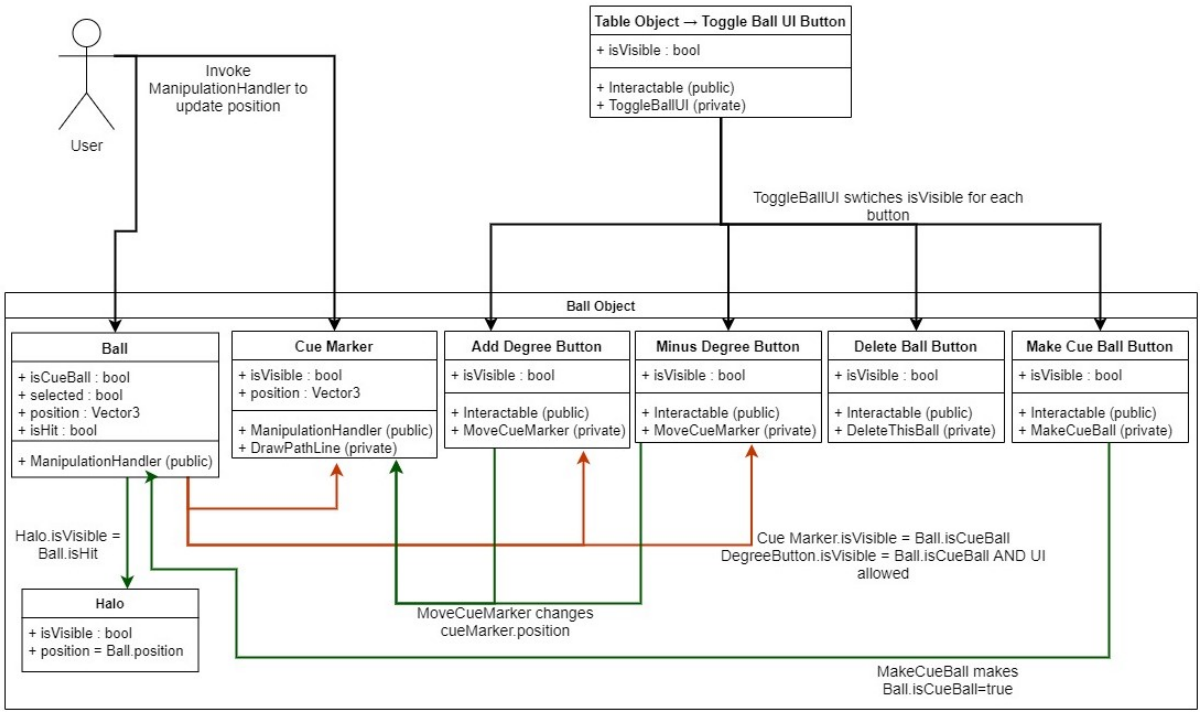


Figure 3.2: Detailed above is a UML diagram of the Ball model, describing its structure and how the user interacts with it.

3.2 Unity Set-Up and HoloLens Deployment

In order to successfully deploy an application to the HoloLens, some setup in Unity and Visual Studio is required. First, all the correct tools need to be downloaded and installed. These are listed at the start of Microsoft's development environment setup guide[29]. Development must happen on Windows 10 and, as previously mentioned, I will need Visual Studio 2019, ensuring that I install the *Desktop development*

with C++ and *Universal Windows Platform (UWP) development* workloads when doing so. Additionally, I require the Windows 10 SDK (build 18362 or later) which can also be installed through the Visual Studio Installer. The reason the C++ development package is required, despite Unity only allowing C# code, is because Unity compiles and builds a C++ based Visual Studio Project, meaning that without this package, I cannot build and deploy this Visual Studio solution to the HoloLens.

With all the needed tools downloaded I can now create a new Unity project, making sure that the *3D* toggle is selected. Once the project has been created and the Unity editor screen opens, I can install and add the MRTK to my project. For this I need to install the latest version of the Mixed Reality Feature Tool[35]. Once installed and launched, I follow the instructions displayed. For my project I only need the base MRTK packages, and so only need to select the packages seen in Figure 3.3. Once my project has been validated as compatible, and any dependencies are found and added, they can be imported into my new project.

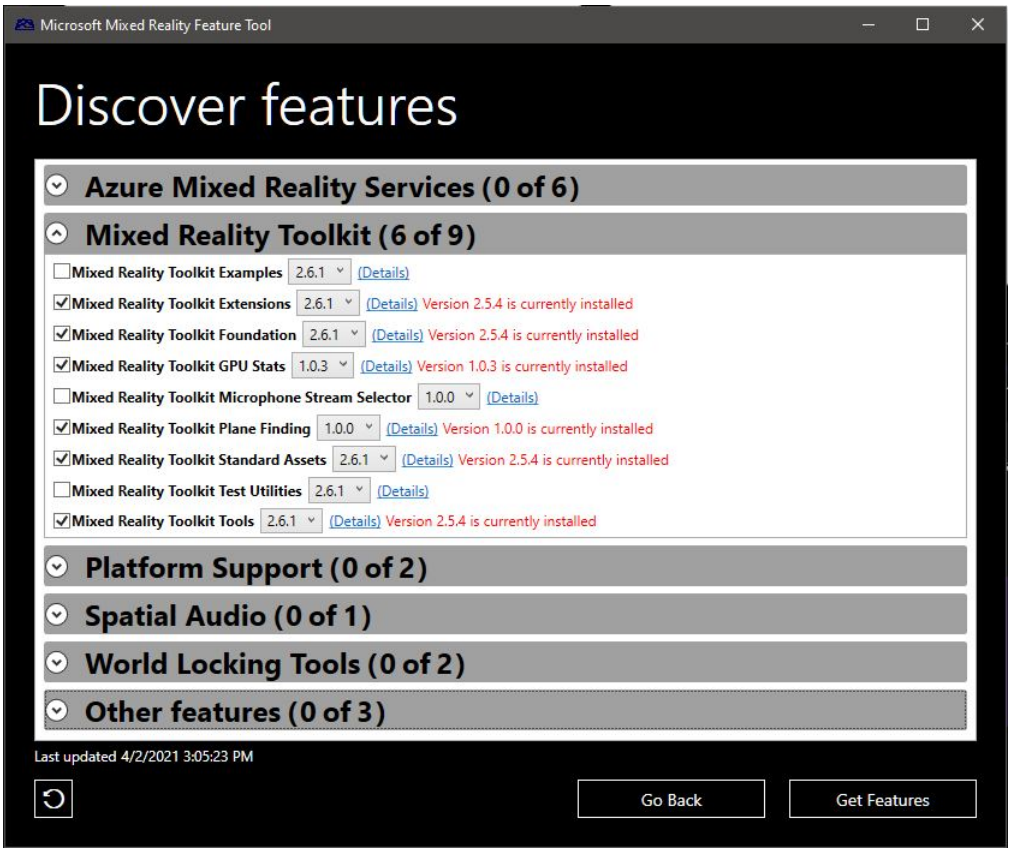


Figure 3.3: Shown are the packages that need to be selected in the Mixed Reality Feature Tool.

Now that everything needed is installed, I need to configure the Unity project for a HoloLens application. Some of the MRTK packages rely on another Unity asset package called *TextMeshPro*. This can be added to the project by clicking on *Window* → *TextMeshPro* → *Import TMP Essential Resources* at the top of the editor window. Next, I need to create a new Scene, which is where I will actually build my application. Then, I need to add the MRTK to the new scene, done by clicking *Mixed Reality Toolkit* → *Add to Scene and Configure* at the top of the Unity editor window, applying the default settings when the configuration pop-up appears.

Shown in Figure 3.4 is what will be in the Hierarchy section of the editor after I add MRTK to my scene. Selecting the *MixedRealityToolkit* object will display Figure 3.5 in the Inspector panel on the right side of the editor window. In order to configure the MRTK to my application type, I need to change some of the settings in the *MixedRealityToolkit* tab, seen in Figure 3.5. Firstly, I want to create a custom profile so that I can customise the camera and render settings. By using the top drop-down box and selecting *DefaultHoloLens1ConfigurationProfile*, I can clone this profile and make changes as

needed. One important change is to the *Target Scale*, where I want to select *World*. This lets the HoloLens know that the user will be walking around the room and will also allow me to use *Spacial Anchors* - something used to stabilise the holograms, covered in depth in Section 3.4. Next, in the profile settings I need to ensure that the *Camera* tab matches the settings shown in Figure 3.5 so that the holograms are rendered correctly. Lastly, in the *Diagnostics* tab I deselect both options, as this prevents a diagnostic box appearing when the application is deployed to the HoloLens[32]. To be able to use the MRTK, the library dependencies seen in Listing B.1 need to be added to the beginning of any C# scripts.

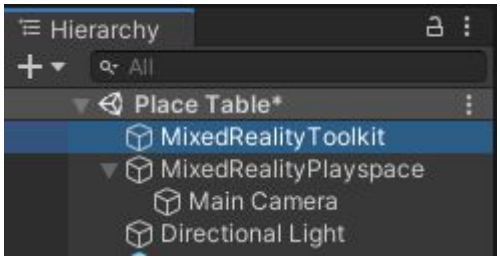


Figure 3.4: Shown is the Hierarchy panel in the Unity editor after adding the Mixed Reality Toolkit to the scene.

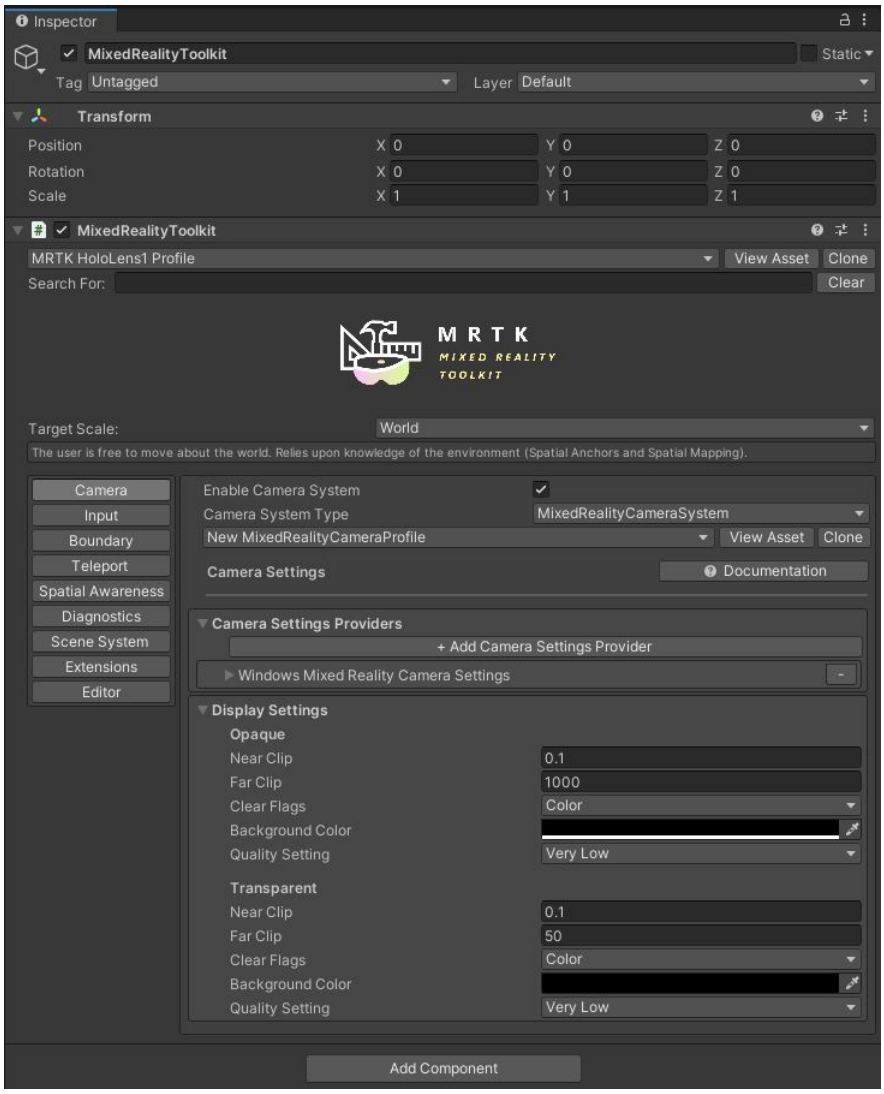


Figure 3.5: Shown is the Inspector Panel in the Unity editor for the MixedRealityToolkit object.

The last project settings I need to change are in *Edit* → *Project Settings* at the top of the Unity editor window. First, in the *Quality* tab I want to change the application quality to very low. As mentioned in Section 1.3, the HoloLens has limited hardware resources, so by selecting the lowest quality level I can attain the best performance possible. Due to the simple shapes used for my models, a high frame rate is more desirable than a high quality sphere. Next, in the *Player* tab I first go to the *Publishing Settings* section and locate the *Capabilities* sub-section. Here I want to enable the *InternetClient, PicturesLibrary, VideosLibrary, WebCam, Proximity, Microphone, SpacialPerception, GazeInput* options so that my application has the correct hardware permissions to operate. Next, in the *XR Settings* sub-section I make sure that *Virtual Reality Supported* is checked and that *Windows Mixed Reality* is present in the *Virtual Reality SDKs* section, ensuring that *Depth Buffer Sharing* is enabled and *Depth Format* is set to 16-bit, again to get the best performance possible. The final Unity configuration settings are located in *File* → *Build Settings*. At the top of the pop-up I add my new scene and select it, then change the rest of the settings to match those seen in Figure 3.6. Once the settings are changed, clicking *Switch Platform* at the bottom of the pop-up will confirm them.

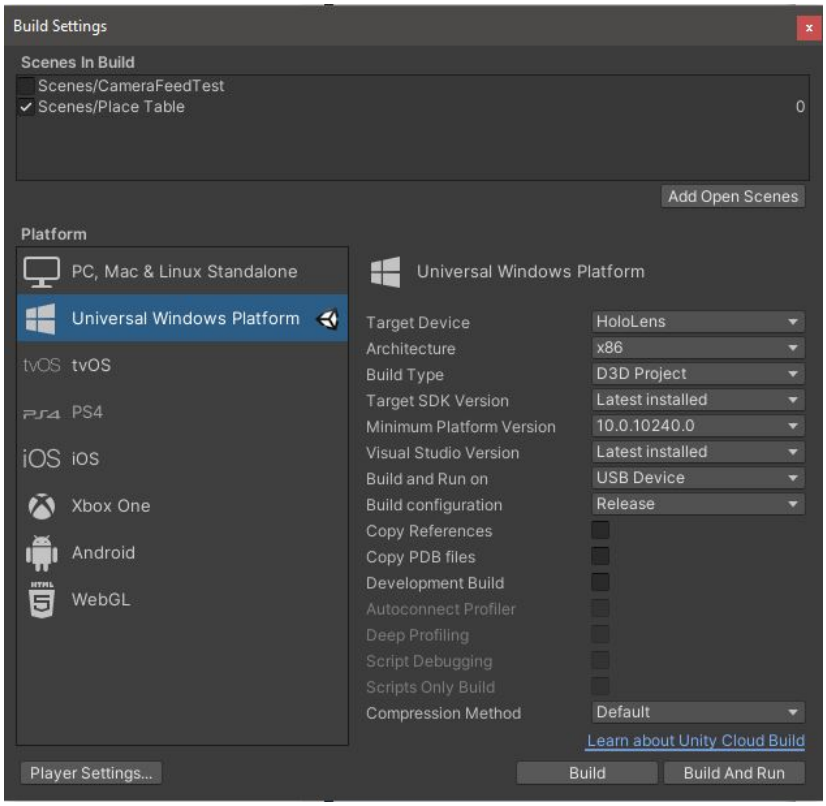


Figure 3.6: Shown are the Unity build settings required for a HoloLens application.

In order to actually deploy my application to the HoloLens, I chose to do this over USB as it was the easiest to do and didn't rely on my home network, which is prone to going down on occasion. I first go to *File* → *Build Settings* → *Build* and wait for Unity to build the application. Once this has been completed, a new Visual Studio solution will have been created. Opening this, I first need to change some build settings in Visual Studio so that it is compiled and deployed correctly. At the top of the Visual Studio window there are two drop down boxes followed by another with a green arrow, as shown in Figure 3.7. The leftmost drop down should be set to *Release*, the middle to *x86* (matching the HoloLens' processor architecture), and the right most with the green arrow to *Device* (telling Visual Studio to deploy to a USB device). Once these are set, clicking *Debug* → *Start Without Debugging* will build and deploy the application to the HoloLens.

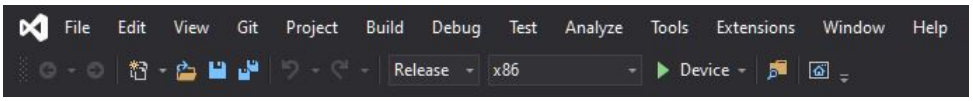
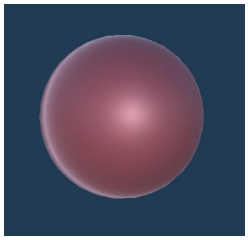


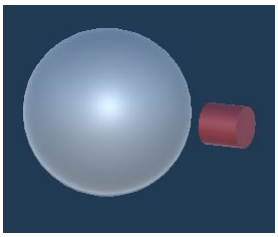
Figure 3.7: Shown are the build and deployment settings required in Visual Studio to deploy an application to the HoloLens over USB.

3.3 Application Models

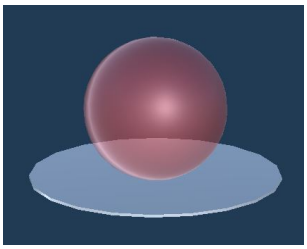
For the entire project only a few 3D models were required. Most of these were simple shapes and could be created in Unity with its included basic object models. The models of the Ball and Cue Marker were simply a sphere and cylinder Unity object respectively, with the ball's radius matching that of a real pool ball - 25mm. The cue marker is only visible on the cue ball, but is present on every ball model as to reduce the number of models in the application. Additionally, if the cue ball's path intersects with another ball, a *Halo* object appears beneath the object ball it will collides with, clearly indicating which ball is being hit. Again, the halo object is present on every ball model but only visible on an object ball that will be hit. Figure 3.8 shows the ball model in each of its different states.



(a) Object Ball Model.



(b) Cue Ball Model with Cue Marker.



(c) Hit Object Ball Model.

Figure 3.8: Shown above are the three different states a ball model can be in - an object ball, a cue ball, or an object ball that will be hit.

In terms of the materials for these objects, a semi-transparent solid colour was chosen for both ball and cue marker objects in order to allow the application's user to see the real-world item behind the holograms more easily. For the Halo, a solid white, non-transparent material was used as this object will appear on the table bed beneath both the holographic and real balls. Non-cue balls and the cue marker were coloured red, with the cue ball coloured white.

In some interim application testing, it was suggested that having the cue marker change colour slightly when the user was to Gaze at it would make it easier to know when they were looking directly at it. This is important as the user needs to Gaze at the cue marker in order to Air-Tap and move it. As such, a second orange material (also semi-transparent) is used for the cue marker and is applied when the user is looking directly at it. To achieve this effect, I added a listener function to invoke the `markerHoverEntered` function when user's pointer is hovering over the Cue Marker object. Additionally, another listener function was added to invoke the `markerHoverExit` function when the user stops looking at the cue marker. This is to reset the material back to the standard red. Attached to the cue marker object is the `ObjectManipulator` script from the MRTK. Its primary functionality is allowing the user to grab and move the attached object with the Air-Tap gesture when their Gaze pointer is hovering on top of it. The script will recognise when the user's pointer is hovering over a game object as both of the object's colliders (detailed in Section 2.4) will be in contact with one another. When this happens, the

`OnHoverEntered` function in the `ObjectManipulator` script is invoked, and this is what I have added a listener to. Listing 3.1 details the code to make the colour change effect happen. The `data` parameter that is passed to the `markerHoverEntered` and `markerHoverExit` functions is the event data, such as the pointer object and it's velocity values, that is returned from the `OnHoverEntered` and `OnHoverExited` functions. For this use case, the parameter can be ignored as I only need to know when `OnHoverEntered` and `OnHoverExited` are invoked.

```
void Start() {
    ...
    cueMarker.GetComponent<ObjectManipulator>().OnHoverEntered.
        AddListener((data) => markerHoverEntered(data));

    cueMarker.GetComponent<ObjectManipulator>().OnHoverExited.
        AddListener((data) => markerHoverExit(data));
    ...
}

ManipulationEventData markerHoverEntered(ManipulationEventData data) {
    cueMarker.GetComponent<MeshRenderer>().material = cueMarkerHovered;
    return data;
}

ManipulationEventData markerHoverExit(ManipulationEventData data) {
    cueMarker.GetComponent<MeshRenderer>().material = cueMarkerDefault;
    return data;
}
```

Listing 3.1: The listener and material updater code needed to make the cue marker change colour when looked at.

The only other model needed for the application is one of a pool table, which the user will align over the top of the real life table so that the shot assistant knows where the table rails and table bed is. In order to test and build my application around the COVID-19 lockdown measures, I borrowed a home pool table that is slightly smaller than a standard sized one. With this, I measured out every dimension and built an accurate 3D model of the table at 1:1 scale using Maya 2019, chosen due to my previous experience using it in a Character and Set Design Module. The final table model can be seen in Figure 3.9. With the model complete, I easily imported it into Unity by dragging it into the `Asset` folder and could immediately add the model to my scene. Similarly to the balls, the table material is semi-transparent to make it easier to see the real table behind the hologram whilst aligning them. As the table model is not seen after its position is confirmed by the user, the table's material is the Unity default gray in colour.

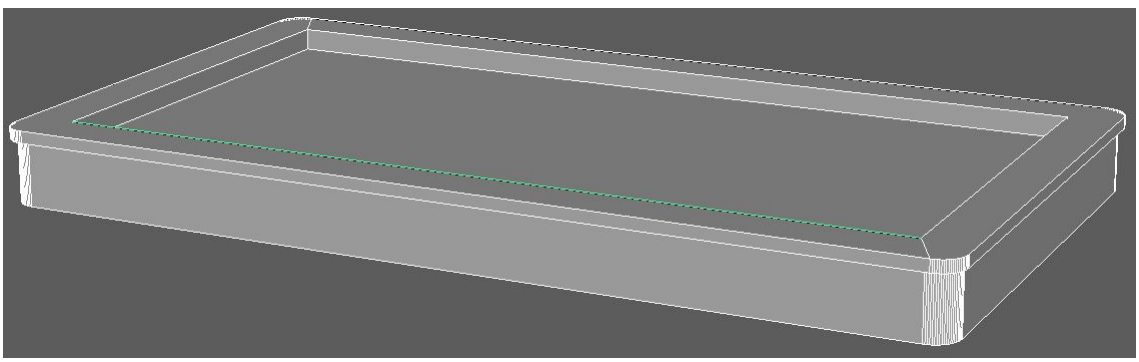


Figure 3.9: The Maya 2019 model of my home pool table, to scale.

3.4 Hologram Stabilisation

For an application such as mine where the holograms will be placed over the top of real world objects, ensuring that the holograms are stable and stationary is vital. As I made the application *World Scale*, it means I can utilise something called a *Spacial Anchor*. Simply, a spacial anchor represents an important world point that the HoloLens tracks over time. Each one has its own coordinate system based on other anchors or reference points, such as a real life object. By rendering a hologram in the anchor's coordinate system, a more precise position is achieved at any given time[34] and the hologram will remain in the exact place the user left it. This physical alignment with the real world is achieved by adjusting the position of the anchor in the application coordinate system based on any other anchors and objects in the room. This ensures each anchor stays precisely where it was placed[18]. For my application, when the user has aligned the table and clicked the *Save* button, I add a spacial anchor to the table object which will automatically place the table and all of its child objects into this anchors coordinate space[53]. Listing 3.2 shows how to add a spacial anchor to an object via a script, where `this` represents the object that the script is attached to - in this case, the table model.

```
void savePos() {
    this.gameObject.AddComponent<WorldAnchor>();
    ...
}
```

Listing 3.2: Shown is the code required to add a Spacial Anchor to an object, and subsequently the scene.

As mentioned above, any child object of the table object will also be in the spacial anchor's coordinate space. Therefore, I made all the table UI elements, such as buttons, and any balls added to the scene children of the main table object, ensuring that the entire application stays as stable as possible.

3.5 User-Object Interactions

For the application to work correctly, the table model needs to be aligned with the real world table and a ball model needs to be aligned with each real world ball. In order to do this, the user needs to be able to Air-Tap and hold to grab a hologram, then move their hand to move the object into place. For the table model, the user also needs to be able to rotate it to the correct orientation. Additionally, the cue marker also needs to be able to be grabbed and moved around the cue ball to allow the user to aim their shot. To achieve this functionality I can utilise some of the scripts included in the MRTK, namely the `ObjectManipulator` and `ManipulationHandler` scripts.

Table Manipulation

For the table model, the user needs to be able to move it freely in 3D space to properly align it with the real world table. Additionally, they may need to rotate the table to align the sides correctly, as well as scale the table up or down if they are using a different table to the one that I have modelled. Adding the `ManipulationHandler` script from the MRTK to the table model allows the user to grab and move the table in 3D space using the Air-Tap gesture. Subsequently, the `BoundsControl` script is automatically added to the table model with the `ManipulationHandler` script due to the inter-dependency between them. This `BoundsControl` script creates a bounding box around the table, seen in Figure 3.10 Image (a), which allows the user to either scale or rotate the table about a different axis. The cubes seen in each corner allow the user to scale the model, where as the spheres allow for rotation of the model about a different axes.

Now that the functionality I wanted is added, there are some limits and conditions that need to be implemented. These are mainly to keep any buttons or UI elements with the table as it is moved or rotated, as well as scale these up or down with the table model. The easiest way to do this was adding listeners to certain functions within the aforementioned scripts that are invoked when the table is moved, rotated, or scaled. Listing B.2 shows the six listeners required, two for each manipulation type (one for `OnManipulationStarted` one for `OnManipulationEnded`). These are located in the `TablePlacement` class that is attached to the table model's parent game object - an object with no physical rendering,

used only to hold scripts and group table items (such as buttons and UI) together.

The four different functions called by the listeners seen in Listing B.2 do almost exactly the same thing. `ManipStart` and `RotateORScaleStart` call the `SetParentTable` function which makes every UI button a child of the table game object rather than its parent game object. This allows for these objects to move with the table if it is rotated or moved in 3D space, as well as stay the same relative size as the table when it is scaled up or down. Listing 3.3 shows the `SetParentTable` function.

```
void SetParentTable() {
    saveBtn.transform.parent = table.transform;
    helpBtn1.transform.parent = table.transform;
    addBallBtn.transform.parent = table.transform;
    clearBallsBtn.transform.parent = table.transform;
    toggleBallUIBtn.transform.parent = table.transform;
    helpBtn2.transform.parent = table.transform;

    helpBox1.transform.parent = table.transform;
    helpBox2.transform.parent = table.transform;
}
```

Listing 3.3: The `SetParentTable` function used to keep UI elements relative to the table as it is manipulated.

Conversely, the `RotateORScaleEnd` and `ManipEnd` functions call a similar function - `SetParentThis`. This function is called after manipulation of the table has stopped, and resets all the button and UI game object's to be the children of the table's parent game object again. The code is identical to Listing 3.3, except `table.transform` for each object is replaced with `this.transform`. The reason for not leaving the buttons or UI elements as children of the table game object throughout the application is to keep an easily readable hierarchy in the Unity editor, as well as prevent any conflicts with other functionality, such as when the table is made invisible once its position is saved (further detailed in Section 3.6).

Ball Manipulation

Like the table model, the user is required to move the ball holograms over the top of the real balls on the table using the Air-Tap gesture and then dragging them into place. However, unlike the table model, the user will not need to rotate or scale the ball model as they are perfectly symmetrical and a standard size. To allow the user to move the ball holograms, I used the `ObjectManipulator` script from the MRTK which allows a user to grab and move an object automatically. Whilst I want the user to be able to move the ball, I also want to restrict this to within the bounds of the surface of the table. When the ball is added to the scene via the *Add Ball* button (detailed in Section 4.4), it is placed at the exact centre of the table and on top of the table bed. Therefore, in the `BallProperties` class, which is attached to the ball game object, I can retrieve the table's current height value in the `Start` function, and then in the `update` function, reset the ball's height to this value every frame. This prevents the user from being able to lift the ball off of the table surface, or pull the ball through the table bed. My other option to prevent the ball changing height would have been to give it an extremely large mass or drag coefficient inside the Unity physics engine. However, the problem with this is that the user wouldn't be able to move the ball at all. In order to prevent the user from moving the ball through the table rails, I attached a collider component to the ball and to each of the table edges. Now, the Unity physics engine will prevent the ball object from being able to pass through the table rails automatically, limiting ball movement to only within the bounds of the table. This is discussed in more detail in Section 3.7.

Cue Marker Manipulation

The final object that the user has to move is the cue marker. By Air-Tapping and dragging, the user rotates the cue marker to aim up their shot, before then using it as a guide as to where to strike the cue ball. Similarly to the ball object, the `ObjectManipulator` script from the MRTK was used to allow the user to grab and move the cue marker. In order to limit the cue marker's movement so that it could

only be rotated around the ball object required the use of a hinge joint. A hinge joint acts just like a door hinge, where the object rotates about a single point. Attaching a hinge joint component to the cue marker, I set the rotation point to be the centre of the cue ball and did not set any limits so that the cue marker can be rotated 360^{o} around the ball. One problem I initially faced from doing this was that when the ball was moved, the cue marker rotated about the ball's initial position, not its new one. There was however a simple fix for this. I added a listener function to the `OnManipulationStarted` function from the `ObjectManipulator` script that invokes the `cueMarkerManipulationStarted` function. In this function, I can update the rotation point of the anchor just before the user starts moving the cue marker, meaning it rotates about the ball's new centre point. The code for this can be seen in Listing 3.4.

```
ManipulationEventData cueMarkerManipulationStarted(ManipulationEventData
    data) {
    ...
    cueMarker.GetComponent<HingeJoint>().connectedAnchor = ball.
        transform.position;
    return data;
}
```

Listing 3.4: The `cueMarkerManipulationStarted` function used to update the rotation point of the cue marker.

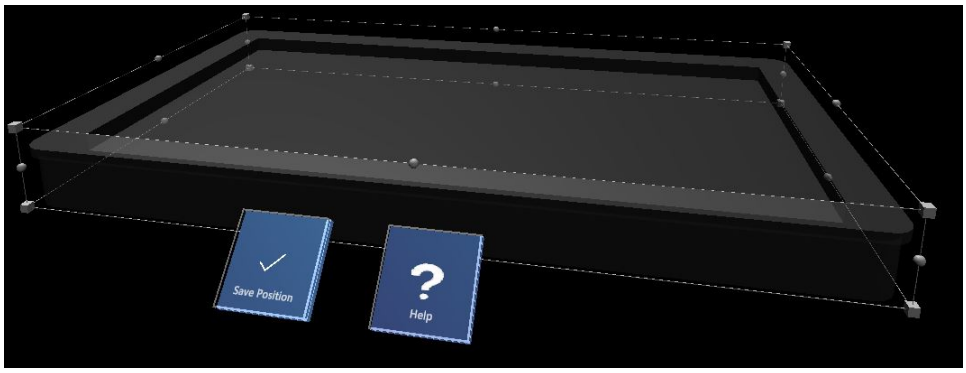
From some interim user feedback, many found the cue marker extremely sensitive to move. To combat this, an extremely high drag coefficient was applied to the cue marker to make it harder to move. Whilst this did work to some degree, some still found it too sensitive. The decision was made to add some fine adjustment buttons to the cue ball that incrementally move the cue marker in each direction. This received better feedback as it allowed a user to line up the shot roughly, then make any final adjustments with these buttons. Details of how these buttons work are found in Section 3.6.

3.6 User Interface

Much of the core program interaction is done through clickable buttons. The MRTK comes with some button models that are ready to use which recognise the Air-Tap gesture as button click. They are configurable to be either a toggle switch (has an off and on state) or a one function button (click to make an action happen). Figure 3.10 and Figure 3.11 show all of the buttons present in the application. As we can see, the table has two sets of buttons, one that is displayed before the user has aligned the table, and another which is displayed after the table has been aligned. Similarly, the ball object has state dependant UI. All balls, object or cue ball, will have the buttons seen in Figure 3.11 Image (a), however, the buttons seen in Figure 3.11 Image (b) are only shown on the cue ball. Detailed in the sub-sections below is what each buttons does and how it achieves this action.

Save Table Position

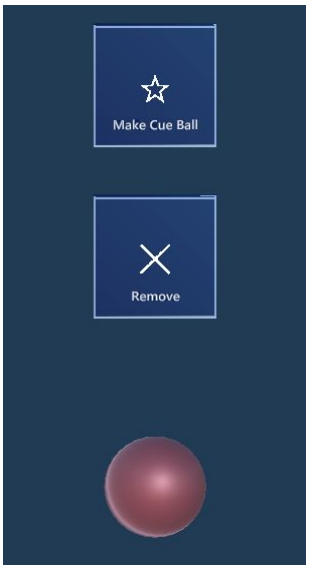
Once the user has aligned the holographic table with the real world one, they need to confirm this position so that they cannot move it out of place accidentally. Additionally, at this point the table model becomes invisible to avoid distracting the user whilst they are using the app. A spacial anchor is also placed when the Save Button is clicked, as discussed in Section 3.4. Once again, using a function listener on the buttons `OnClick` function from the `Interactable` component, I can invoke my own function, `SavePos`. Within this function, first the tables Manipulation based scripts are disabled, followed by the Save Position and the first Help Information buttons being disabled to hide them. The buttons seen in Figure 3.10 Image (b) are then enabled to make then visible and clickable by the user. Lastly, each of the table parts (each rail, the table bed, as well as the rest of the shroud) is hidden. Note that in Figure 3.10 Image (b), this part of the code was disabled in order to create the screen grab of the UI next to the table. Listing 3.5 details the code required to achieve the aforementioned functionality in the respective order.



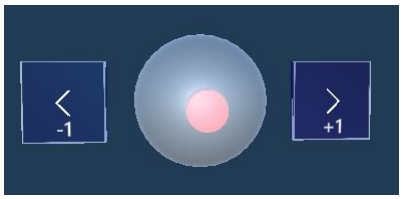
(a) Table with UI before position has been saved.

(b) Table with UI after position has been saved.

Figure 3.10: Shown are the buttons present next to the table model.



(a) Object ball with UI elements.



(b) Cue ball with cue marker directional buttons.

Figure 3.11: Image (a) shows the UI that both object and cue ball's have. Image (b) shows the UI that only the cue ball has.

Display Help Information

In order to inform the user of how to use my application, I included some help billboards that are displayed above the table. The information on the two billboards, one for pre-placement and one for post-placement, are displayed using a custom material of the text instructions on flat cubes, whose positions remains above the table as shown from Listing 3.3 (`helpBox1` and `helpBox2`). From Listing 3.5 it is apparent that two help buttons were used in the application, one for pre-placement and one for

```
void SavePos() {
    ...
    //de-activate scripts responsible for movement
    table.GetComponent<BoxCollider>().enabled = false;
    table.GetComponent<NearInteractionGrabbable>().enabled = false;
    table.GetComponent<ManipulationHandler>().enabled = false;
    table.GetComponent<ConstraintManager>().enabled = false;
    table.GetComponent<BoundsControl>().enabled = false;

    // Dissable save btn and help btn 1, enable add ball btn, clear
       balls btn, toggle UI btn, help btn 2
    saveBtn.SetActive(false);
    helpBtn1.SetActive(false);
    addBallBtn.SetActive(true);
    clearBallsBtn.SetActive(true);
    toggleBallUIBtn.SetActive(true);
    helpBtn2.SetActive(true);

    //make table invisible
    foreach(Transform child in table.transform)
    {
        if(child.parent.name == table.name && child.name != "rigRoot" &&
            child.tag != "Pocket") child.GetComponent<MeshRenderer>().
           enabled = false;
    }
}
```

Listing 3.5: The code required to change the table UI, hide the table model, and lock the table into position.

post-placement. The decision to use two buttons was made to distinguish the information that they would show, as well as removing the need to update the position of a single button when the table position was saved. Despite there being two help buttons, they both have identical functionality - showing or hiding a game object. Again, a listener was used to wait for the invocation of the `OnClick` function, at which point my own function was called that either performed `helpBox.SetActive(false)` or `helpBox.SetActive(true)`, depending on if the button was toggled on or off.

To ensure that the user could always read the help information displayed, the Unity `LookAt` function was used. Invoked in `Update`, the `LookAt` function rotates a game object so that the front of it (defined by its surface normal) points towards another game object - in this case the *Camera* (dictates what the user sees and is where the application is rendered from). Listing 3.6 shows how the `LookAt` function was called to ensure both help billboards were always readable.

```
void Update() {
    ...
    // Keep help text boxes looking at user
    helpBox1.transform.LookAt(Camera.main.transform);
    helpBox2.transform.LookAt(Camera.main.transform);
}
```

Listing 3.6: Utilisation of the `LookAt` function to make the help information always point towards the user.

Add Ball to Scene

Quite simply, the Add Ball button adds a new ball object to the middle of the table so that the user can align it with a real life ball. Again, a listener is used to detect when the button is clicked by the user, at which point my `addBall` function is called. Listing 3.7 shows the code used to create a new ball object and place in the centre of the table. Here, `ball_prefab` is a blueprint that allows the script to add an exact copy of a ball object to the scene. The `ball_pos` variable is the center of the table model at exactly the height of the table bed, accounting for the table being scaled.

```
void addBall() {
    float scale = table.transform.localScale.y;
    Vector3 ball_Pos = table.transform.position + new Vector3(0, (0.25f
        * scale), 0);
    Instantiate(ball_Prefab, ball_Pos, Quaternion.identity, this.
        transform);
}
```

Listing 3.7: Shown is the `addBall` function which is invoked when the Add Ball button is selected by the user.

Remove All Balls From Scene

Similar to the previous buttons descriptions, a custom function is called via a listener when the user selects the Clear Balls button. Listing 3.8 shows said function, which searches through the application hierarchy and finds any game object with the `"Ball_Marker"` tag, then removing it. Tags are used in Unity to describe a game object and allow then to be found easily in C# scripts. Multiple game objects can have the same tag, usually if they are the same type of object (i.e. both are car objects but are different models), but this does not link them in any way.

```
void ClearAllCurrentBalls() {
    var listBalls = GameObject.FindGameObjectsWithTag("Ball_Marker");
    foreach(GameObject ball in listBalls)
    {
        Destroy(ball);
    }
}
```

Listing 3.8: Shown is the `ClearAllCurrentBalls` function which is invoked when the Clear Balls button is selected by the user.

Toggle Ball UI

The final of the table buttons, Toggle Ball UI, allows the user to make the buttons that show next to any of the balls disappear, allowing them to take the shot without distraction. For this buttons functionality, it first retrieves whether or not the button is toggled on (hide UI) or off (show UI). If it is toggled on, then similar to Listing 3.8, all ball objects are found and all buttons are disabled by switching the `allowUI` parameter in the script attached to the ball to `false`. If the button is toggled off, then every ball object is found and the `allowUI` parameter is set to `true`, along with enabling the cue marker buttons on the cue ball. These buttons are enabled manually as they are always shown (when ball UI is allowed) compared to the other two buttons (Make Cue Ball and Remove Ball) which are only shown above the ball that the user has most recently looked at. This functionality is implemented by invoking a function when the user looks directly at a ball object - similar to changing the cue marker's colour as described in Section 3.3.

Make Cue Ball

Within the application, and any cue sport, there can only ever be a single cue ball. In the script attached to each ball object, there is an `isCueBall` boolean parameter that indicates if a ball is a cue ball. When the Make Cue Ball button is selected, all ball objects are found (similar to Listing 3.8) and this `isCueBall` parameter is set to `false`, along with changing each ball's material to red and disabling its cue marker and buttons. Then, before the function returns, `this.isCueBall = true` is performed in order to make the ball that the Make Cue Ball button is attached to the new cue ball. Additionally, the ball's material is changed to white and the cue marker, as well as the cue marker buttons, are enabled.

Remove Ball

When selected, the listener function invokes a single lined function. By running `Destroy(this.gameObject)`, Unity will remove the game object that the script is attached to - in this case the ball object the Remove Ball button was attached to.

Move Cue Marker Buttons

The final buttons to cover are the ones which move the cue marker either clockwise or anti-clockwise around the cue ball. Listing 3.9 shows the code for the clockwise movement, with the only difference in the anti-clockwise function being in line 10 where the target position equals `hinge.angle - 1.0f` instead. Initially, the cue marker is made *non-kinematic*, meaning that it can be moved by external forces such as gravity or colliding with another object. Then, the pivot point of the cue marker's anchor is set to the centre of the ball for the same reason described in Section 3.5. On a hinge joint, there is an attribute called a *HingeSpring* which can be used to make the hinge joint return (or *spring*) back to the same point. This is mainly used for swinging doors, but I can update this return point to 1-degree clockwise or anti-clockwise of the current cue marker's position. As I previously made the cue marker react to external forces, the *HingeSpring* will make it move around the cue ball. It is important that the cue marker becomes *kinematic* again, meaning it does not react to external forces, so that it will move with the cue ball around the table. Additionally, being kinematic will ensure the cue marker doesn't behave abnormally when the user tries to move it without the buttons - i.e. by being continually affected by the *HingeSpring* force and springing back to a single location. This is done in the functions that are invoked when the cue marker or ball itself are being grabbed and moved by the user. Listing 3.9 shows the `addDegree` function that is responsible for moving the cue marker around the ball.

```
void addDegree() {
        cueMarker.GetComponent<Rigidbody>().isKinematic = false;
        cueMarker.GetComponent<Rigidbody>().WakeUp();

        cueMarker.GetComponent<HingeJoint>().connectedAnchor = ball.
           transform.position;

        HingeJoint hinge = cueMarker.GetComponent<HingeJoint>();
        JointSpring hingeSpring = hinge.spring;
        hingeSpring.spring = 1000;
        hingeSpring.damper = 0;
        hingeSpring.targetPosition = hinge.angle + 1.0f;
        hinge.spring = hingeSpring;
        hinge.useSpring = true;
    }
```

Listing 3.9: Shown is the `addDegree` function that moves the cue marker around the cue ball (clockwise) by 1-degree.

3.7 Inter-Object Collisions

As mentioned in Section 3.5, a collider component is added to the ball so that the user cannot move it through the table model and into an invalid position. Colliders are also used for the objects that the user can grab and move, and are one of the components that the scripts in the MRTK use to know when the user is looking at a hologram - as the pointer's collider and objects collider will be touching. When two colliders come into contact, Unity automatically calculates the force that should be applied to both objects by using their velocity and mass, resulting in a rebound effect. Information about an object's velocity and mass, as well as its drag, whether it is affected by gravity, or whether it is *kinematic*, is stored in another component called a *Rigidbody* (previously described in Section 2.4). If I want two objects to collide, and therefore not be able to pass through one another, but not want the objects to bounce off of one another, then I can make them both *kinematic*. By being *kinematic*, an object will not react to any forces applied to it, such as gravity or those applied to it when a collision occurs. This is especially useful for my application as I want the balls to be able to collide with and be stopped by the table rails, but I don't want the balls to bounce around the table when they are put against said rail. Being kinematic also means that the user cannot throw the cue ball or cue marker, especially useful for objects that need to be placed precisely.

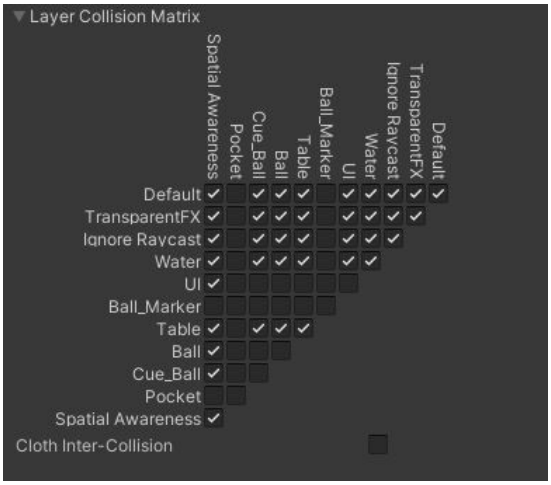
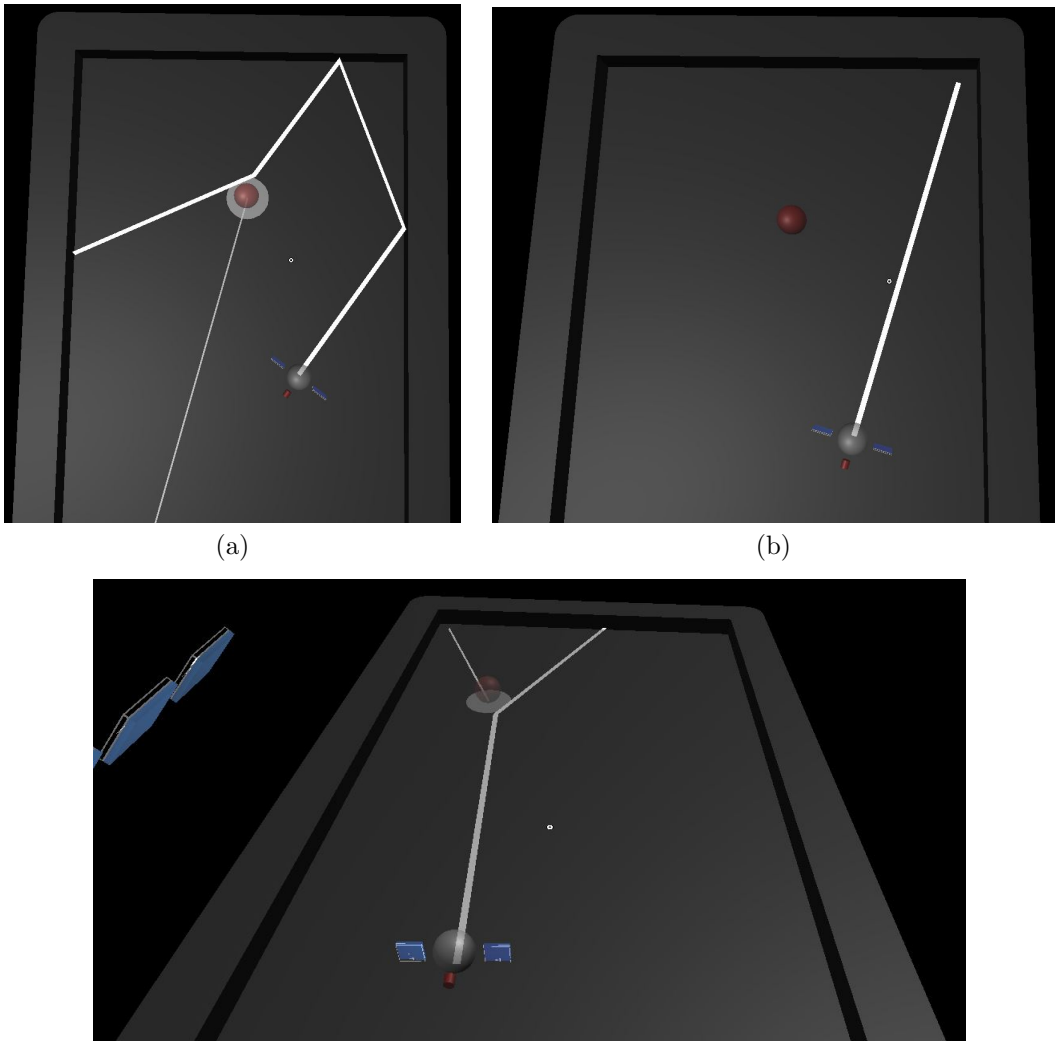


Figure 3.12: The Layer Collision Matrix used in my application.

There are some instances where two objects require a collider components but I don't want them to interact or collide at all - say two ball models, or the cue marker and the table model. This is where *Layers* come in. A Layer defines which game objects can interact with one another and are commonly used by lights to illuminate only parts of the scene, or used to selectively ignore certain colliders[48]. Each game object must be assigned a layer and there is the option to create a new layer if appropriate. For my application, it was useful to have separate layers for object balls, cue balls, the table, the cue marker, and the pockets of the table. By assigning the game objects relevant layers, I can then alter the *Layer Collision Matrix* in the Physics settings of Unity to dictate which layers should interact with each other. This can be seen in Figure 3.12, where it shows that balls and cue balls will react, and therefore collide, with the table layer. However, the table model will not collide with the cue marker (layer labelled *Ball_Marker* in Figure 3.12) or the pocket layers. This is especially important as I want the player to be able to move a ball or cue ball right up against a rail of the table, but if the cue marker were to interact with the table, it would simply bounce off preventing this from being possible. As for the pocket colliders, these are static and overlap with the table's collider component, so by not allowing these layers to interact I prevent the table from flying across the scene. Additionally, notice that neither cue balls nor object balls will collide with pockets. This is due to the pocket colliders being used solely for the trajectory estimations and detecting if a ball's path will intersect with a pocket - meaning that in real life the ball would fall into the pocket. More detail is on this is provided in Section 3.8, along with the reason for using a different layer for cue balls and object balls.

3.8 Post Collision Direction Estimation and Path Drawing

The final, and most important, part of the implementation was calculating the cue ball's trajectory and plotting this path onto the table for the user to see. Figure 3.13 shows three examples of this system in action. Image (a) shows the cue ball rebounds off of two table rails, then colliding with an object ball. Image (c) shows a direct collision with an object ball, potting it into a corner pocket. Seen in both images is the fact that after a collision with an object ball, both object ball and the cue ball have their path drawn up to the next collision. The decision to not do any post-collision rebounds was made as to keep it clear which direction the balls would travel in and to reduce the computational load on the system.



(a) (b)

(c)

Figure 3.13: Shown are three examples of a cue ball's shot path. One having two rebounds, one having a direct hit with an object ball, and one having a direct path into a pocket.

Image (b) from Figure 3.13 shows the cue ball being aimed directly at a pocket. Notice that no rebound lines are calculated when this happens. This is down to the path estimation system interacting with the pocket collider component and knowing not to calculate any more of the trajectory. Just how the estimation system detects if the cue ball's path will be intercepted by a collider - whether it be another ball, the table rail, or a pocket collider - is down to the use of three *Raycasts*. Part of the Unity physics engine, a raycast fires a ray into the scene from a given point at a given direction and returns the information about the first collider component that it hits. This information crucially includes what game object was hit (including the game objects layer and tag) and the global position at which the collision occurred. If this is repeated a number of times, updating the firing point and direction each time, a series of points can be collected to represent the path of every collision that will occur. This

information can then be sent to another Unity component called a *LineRenderer*, which takes a series of points in 3D space and connects them with with a singular line. These two components enable the path estimation feature found in my shot assistant application.

There are two collision scenarios, ball-to-ball and ball-to-rail. The algorithm for calculating the post collision trajectory is very different for these two collision types, but the pre-collision part of the system is identical. To setup at the beginning of the path estimation function, the current position of the cue ball is retrieved, `cuePos`, as well as the direction vector pointing from the cue marker through the centre of the cue ball, `dir`. Using the cross product of the direction vector and the y=axis vector $(0, 1, 0)$, the normal to the direction vector is found, `dirNorm`. We need the normal so that we can find the points either side of the cue ball's centre that intersects with the cue ball's circumference, and fire a raycast from these two points in the direction of `dir` too. The following lists are then initialised to keep track of important values.

- `currentPositions` - keeps track of the current ray firing locations (initially: `cuePos`, `cuePos` + ball radius along the normal, `cuePos` - ball radius along the normal).
- `rayHitPoints` - keeps track of the information received from the three raycasts.
- `pathPoints` - keeps track of the collision points to draw the trajectory path between.

After this setup, a for loop is then started which will perform up to a maximum of 5 repetitions, meaning that the cue ball's path will be drawn up to 4 bounces off of rails. At the start of the for loop, three raycasts are fired, one from each of the positions in `currentPositions`, with the return information stored in `rayHitPoints`. From the information returned from these raycasts, I can check if any of the raycasts hit a ball by querying the hit game object's tag. If one or more raycast hits a ball then I initiate the ball-to-ball collision code, if not then I initiate the ball-to-rail collision code. The code for the beginning of this for loop is seen in Listing 3.10, with a visual representation of the rays being fired found in Figure 3.14. The raycasts are all within `if` statements as occasionally the raycast will misfire and not collide with anything. If this happens then the application crashes as later in the function I would be referencing values that do not exist.

```
for (int j = 0; j < 5; j++) {
    if (Physics.Raycast(currentPositions[0], direction, out rayHitPoints
        [0], Mathf.Infinity, layerMask)) {
        if (Physics.Raycast(currentPositions[1], direction, out
            rayHitPoints[1], Mathf.Infinity, layerMask)) {
            leftSideHit = true;
        }
        if (Physics.Raycast(currentPositions[2], direction, out
            rayHitPoints[2], Mathf.Infinity, layerMask)) {
            rightSideHit = true;
        }
        // Check which rays hit balls
        if (rayHitPoints[0].collider.tag == "Ball") ballHitC = true;
        if (leftSideHit == true && rayHitPoints[1].collider.tag == "Ball
            ") ballHitL = true;
        if (rightSideHit == true && rayHitPoints[2].collider.tag == "
            Ball") ballHitR = true;
        ...
```

Listing 3.10: Shown is the beginning of the for loop which fires the raycasts into the scene.

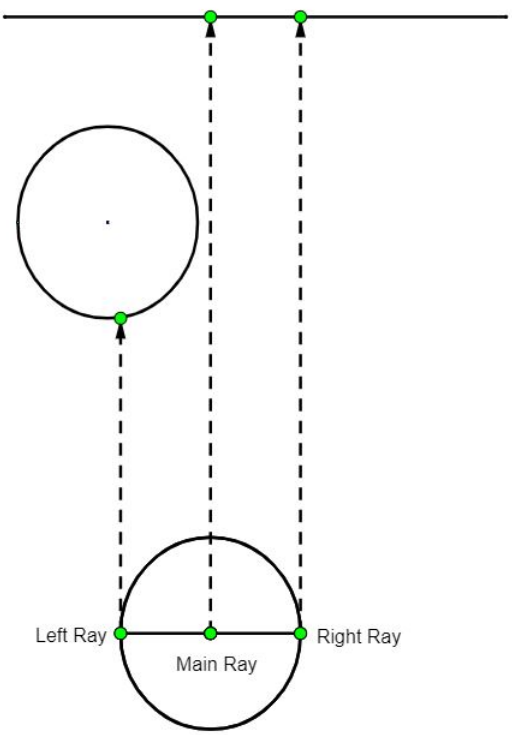


Figure 3.14: Depicted is a visual representation of all three raycasts being fired into the scene, with the green nodes at the end of the dotted vectors being the collision points.

Before getting into the details of each collision type, it may be clear that I want the raycast to ignore certain game objects such as the cue ball itself, the cue marker, as well as any other UI elements (such as the cue marker buttons). To do this, a layer mask is used. Represented as an integer, a layer mask can be used to tell the raycast which physics layers to ignore. With a size of 32-bits, and a maximum of 32 layers, every bit of the layer mask that equals `0` will tell the raycast to ignore that layer number. For example, if I want to ignore all game objects on the cue ball layer, which has layer number 8, then I can turn bit number 8 in the layer mast to a 0. To do this, I used bit shifting as follows : `layerMask = unchecked((int)0xFFFFFFFF - (1 << 8));`. This was also done for the cue marker and UI layers to exclude them from any raycast collisions.

Ball-to-Rail Collisions

After doing the check shown at the end of Listing 3.10, if none of these Booleans are `true` then the raycasts must have hit either a pocket or a table rail. Seeing as I know none of the raycasts hit a ball, I can just focus on the centre raycast from now on, as if only the left or right ray hit a pocket collider then the ball will not drop into a real life pocket. To check if the centre raycast hit a pocket I simply query its return information. If the game object's tag is `Pocket`, then I store the global position of where the raycast collided with the pocket by appending it to the `pathPoints` list, and break out of the for loop. This will then invoke the cue ball's Line Renderer component to draw the estimated path.

If no pocket was hit, then I now need to calculate the angle the cue ball will travel after it bounces off of the table rail. By following the assumptions in Section 2.5.2, I first retrieve the surface normal of the rail the centre raycast collided with, before calculating the angle between the cue ball's direction vector and the table rail, α from Figure 2.7. To calculate α, I know that the angle between the table rail and its normal is 90^o, and Unity includes a function to find the angle between two 3D vectors. Therefore, `alpha = 90 - Vector3.SignedAngle(dir, surfaceNormal)`.

From Section 2.5.2, it was said that given the coefficient of restitution and the ball's initial velocity, I can calculate the exit angle β. For the initial velocity, I always assumed this as $3.0ms^{-1}$, using the typical

fast ball speed provided by Alciatore[5]. As for the coefficient of restitution, I calculated this myself by using a bird's eye view recording of myself performing a rebound shot. By selecting three video frames, one for the starting ball position, one for the position of the ball when in contact with the rail, and one for the end position, I could measure the distances traveled in pixels using Adobe Photoshop and calculate the pre-collision and post-collision speeds. With these velocities, I can plug them into the coefficient of restitution equation seen in Section 2.5.1 - with V_1 and U_1 being 0 as the table has no velocity. For my table, this came out to be $e = 0.54$. With these values, I can now calculate β as described in Section 2.5.2.

To calculate the next section of the path, I need to fire new raycasts from where the cue ball has just collided with the rail. This requires a new direction vector and starting point. Now, the starting point, `cuePos`, is updated to where the cue ball has just collided with the rail (the global position returned from the previous raycast). To update the direction vector, `dir`, so that the angle between it and the table rail that was just collided with is β, I reverse the current direction vector (u from Figure 2.7) and rotate it about the updated firing point by $180^o - \alpha - \beta$. The resulting vector is one that points away from the table rail at an angle of β (v from Figure 2.7). Before progressing to the next loop repetition and finding the next section of the collision path, I append the updated `cuePos` to `pathPoints` so it is drawn later on. I then find the updated direction vector's normal and use this to update the values in `currentPositions` to the new firing points. The logic calculating the cue ball's post rebound trajectory vector can be seen in Listing B.3.

Ball-to-Ball Collisions

If one of the raycasts did hit an object ball, then first I need to find out which ball hit is the closest to the cue ball, in the event where multiple balls were hit. Once this has been determined using the Pythagorean theorem ($c = \sqrt{a^2 + b^2}$), I then need to calculate two equations - the equation of a straight line and the equation of a circle. By equating the centre raycast's line equation with the equation of a circle, whose centre is the hit ball's centre and whose radius is twice that of an object ball, I will find two points. The closest of these two points to the cue ball will be the cue ball's centre position at the exact time it collides with the object ball. This can be seen in Figure 3.15. The cue ball's centre position at the exact time it collides with the object ball is used to find the post collision directions of the object ball and the cue ball. The reason I am using double the radius of the object ball for the circle equation is because both cue ball and object ball have the same radius, and so at the time of collision the distance between the cue ball's centre and the object ball's centre will be exactly 2 times their radius.

To calculate the line of the raycast, I can use $z = mx + c$, the equation of a straight line. Here, z is being used instead of y as I know the y coordinate will not change due to the raycast being fired perpendicular to the y-axis. This also means that I can work in 2D geometry, simplifying everything greatly. Listing 3.11 shows how the gradient (m) and the z-intercept (c) are found.

```
float gradient = (cueBallCentre.z - rayHitPoints[0].point.z) / (
    cueBallCentre.x - rayHitPoints[0].point.x);
float intercept = cueBallCentre.z - gradient * cueBallCentre.x;
```

Listing 3.11: Finding the gradient and z-intercept of the central raycast's line equation.

As for the circle equation, I can simply plug the x and z values of the hit object ball's centre coordinates (x_b and z_b), as well as 2 times its radius (r) into the standard circle equation ($(x - x_b)^2 + (y - y_b)^2 = r^2$). In order to find the x coordinates of the intersect points, I can substitute the z value in the circle equation for the raycast's straight line equation and simplify. The resulting equation is :

$$x^2(1 + m^2) - 2x(x_b - m(c - z_b)) = 4r^2 - x_b^2 - (c - z_b)^2$$

The full simplification to get to this equation is detailed in Appendix A. As this is a quadratic equation, I can use the quadratic formula ($x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$) to calculate my two x values, and plug these into the $z = mx + c$ equation to get the matching z coordinates. Then, again using the Pythagorean theorem, I can calculate which of these two points is closer to the cue ball and is therefore the cue ball's centre point at the time of collision. Listing B.4 shows the code that achieves this.

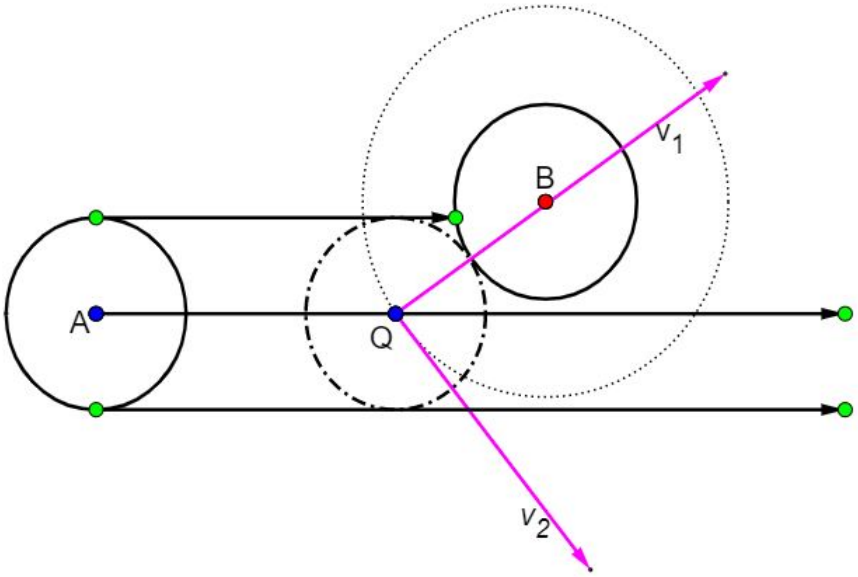


Figure 3.15: Shown is a visualisation of how to calculate the centre point of the cue ball (A) at the exact time when it collides with an object ball (B). This is done by finding the intersection points of the centre vector coming from point (A) and the dotted circle with centre point (B). One of the two found points will be point (Q).

With point Q from Figure 3.15 now found, I then append this centre point to `pathPoints`. To calculate the object ball's post collision direction vector I find the vector that passes through the cue ball's centre point at the time of collision and the object ball's centre point. Using the approximations in Section 2.5.1, I calculate the rebound angle of the cue ball and form its post-collision direction vector by rotating the object ball's post collision direction vector accordingly about Q. Listing B.5 shows the code that retrieves both of these direction vectors.

Finally, for both the cue ball and object ball, one last raycast is performed to collect the last collision points needed to draw the full trajectory path line. After these points are collected and appended to `pathPoints`, the outer for loop is broken to invoke the Line Renderer and the estimated path is drawn for the user to see.

Chapter 4

Critical Evaluation

Throughout this chapter, each major part of the shot assistant application was tested and evaluated in order to gauge the success of the project, as well as its usefulness and usability. Within each section, details of the experiments performed and discussion of the subsequent results are present in order to give a contextual, in-depth overview of the findings.

4.1 Hologram Stability

A major part of the application working well, and being effective in improving the user's potting accuracy, is for the holographic content to stay in place whilst the user moves around the room. As most of the holographic content is placed over the top of real life objects, i.e. the balls and table, if they were to move out of alignment then the user would not be able to line up a shot accurately, making the application less effective. Some mitigations were taken in order to try and reduce the amount holograms move as the user moves about the room, as detailed in Section 3.4, and it is important to see how effective these were in stabilising the holograms the application displays. In order to determine how drastic this movement (or *drift*) is, and to what extent it interferes with someone using the system, the following experiments detailed in the sub-sections below were performed. Their main goal was to try and imitate what a user would typically do whilst planning and taking a shot, as well as some other common movements or environment changes that any HoloLens wearer could experience regardless of the application that they were using.

Whilst the HoloLens is capable of taking photos and videos capturing what the user sees, there is an apparent offset whilst doing so [37]. Although the user will see that the content is aligned, in said MR photo/video content this is often not the case, with the holograms appearing positioned up and to the left of where they actually are at distances of greater than approximately one meter. As such, most captured content cannot be used to accurately evaluate any holographic drifting that may occur. An effort was made to try and reduce or eliminate this offset effect using the techniques advised by Microsoft [19][31], however the issue still persisted after the fact. It is apparent that for Microsoft's newest HoloLens device, the HoloLens 2, there is a way for developers to enable rendering of the holographic material from the point of view of the photo-video camera, which ensures MR captured content is aligned at any distance[33]. However, I did not have a HoloLens 2 available to me and so this could not be tested nor verified.

To get around not being able to capture content from the user's perspective, an alternative evaluation method has been devised which was used for all experiments detailed in the sub-sections below. This method is as follows:

1. A camera was placed in a single position throughout testing such that it had a clear view of the entire table.

2. After starting my application and placing the table model into place, I place a holographic ball onto the table in any location.

3. I then place a real life ball in the exact position of the holographic ball, leave it for a few seconds so that the camera can record it in position, then remove it.

4. From here I perform the necessary actions that I am evaluating before returning to the table.

5. Then I again place the real ball in the exact position that I see the holographic ball and leave it for a few seconds so that the camera can record it in position.

6. With these recorded positions, I can extract single frames from the video of each time I placed the real ball and compare its position between frames.

Using this method, I can compare the ball positions between the frames either visually, or by aligning and superimposing them onto each other. If there is little to no movement of the ball between the aligned frames then the holograms are stable. However, if there is drastic differences between the ball's position in the frames then I will know that the holograms are not stable under the tested conditions. To maintain accuracy, the camera position was fixed throughout each test and I myself carried out each test as to not introduce any variation between ball placement or judgement of where the ball is.

4.1.1 Movement Around the Table

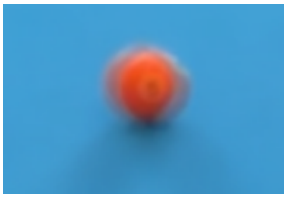
In order to test how stable the holograms would be whilst a user was walking around the table, like they would do in a game or when practicing, I lined up the real ball with where I saw the holographic one from each side of the table. Figure 4.1 shows the ball's apparent position from the left, top, right, and bottom edges of the table from one of the five test repetitions, each of which yielded consistent results.

(a) Position 1 (LHS).

(b) Position 2 (TOP).

(c) Position 3 (RHS).

(d) Position 4 (BOT).



(e) All balls from frames superimposed.

Figure 4.1: The above images show where the holographic ball was seen in each position. These have been superimposed in image (e) to allow for easier recognition of any drift that has occurred.

From panel (e) in Figure 4.1, we can see that there was some minor movement between each of the four positions of up to approximately a quarter of the ball's diameter, or 12.5mm. This is clearly not ideal, especially for thin cut shots, as the user's shot that they have lined up will not be accurate due to the drift that has occurred. The user can correct this by moving the ball's back into place before they take their shot, but this is not ideal and is likely to frustrate them over time. Although I am unable to

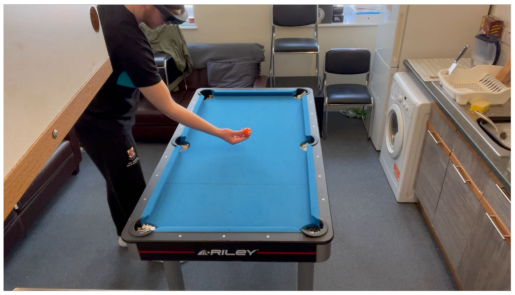
test or evaluate if the table hologram has also drifted, I can assume that if the balls have experienced drift then so has the table. This is because all ball objects are children of the table object, meaning that they are subject to the *spacial anchor* attached to the table[53]. Whilst in theory the *spacial anchor* should keep the holograms in the exact place they were left, it is clear that this has not happened exactly from the drift apparent in panel (e). It will become clear in Section 4.2 whether or not this minor drift caused by moving around the table has any effect on the accuracy of the path prediction system.

4.1.2 Major Environmental Changes

For this test, I wanted to see how the HoloLens and the holograms it displays reacted to environmental changes, such as leaving the room briefly and returning. Microsoft states that tracking can be effected due to dynamic changes in the environment, such as people walking around the room[21] or significant movement of known objects such as furniture[20]. What isn't clear is how an addition to the known environment, like walking into another room, will effect the HoloLens' tracking and subsequently the position of any holograms. In order to give the best chances if success, an ideal testing condition environment was used as detailed by Microsoft[26]. Therefore, adequate lighting was used, no objects within either room were moved, no other people were moving around the rooms, and both rooms had a large number of objects to help improve tracking.

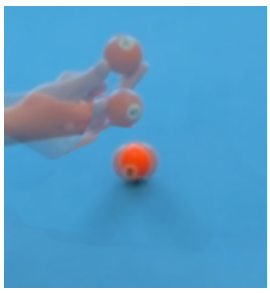
(a) Initial position.

(b) Position after leaving the room once.



(c) Position after leaving the room twice.

(d) Position after leaving the room thrice.



(e) All balls from the above images superimposed.

Figure 4.2: The images above show the position of the holographic ball after leaving and re-entering the room a number of times. These have been superimposed in image (e) to allow for easier recognition of any drift that has occurred.

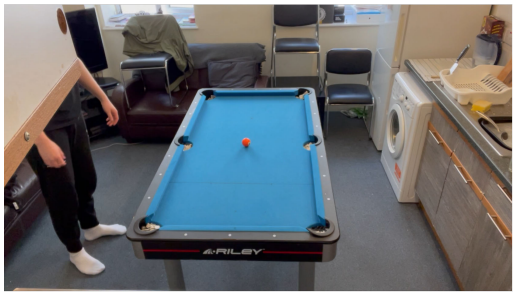
Figure 4.2 shows the position of the holographic ball, represented to the camera by placing the real ball in the same location as the holographic one, after leaving the room and returning up to three times. Just from comparing panels (a) through (d), it is evident that this had a major effect to the tracking of

the HoloLens, causing substantial holographic drift. This is visualised more clearly in panel (e), where it can be seen that vertical drift is most apparent, with horizontal drift kept fairly minimal. Interestingly, it seems that only leaving the room once had a similar impact to the hologram positions as moving around the table did, but on repeated exit and entry to the room there was more of an effect. Despite this, it becomes clear that once a user begins to use the shooting assistant that they should not leave the room whilst wearing the headset, else the application becomes unusable without restarting it.

4.1.3 Removal of the Headset

As with many head mounted displays, there are a proportion of users who experience discomfort, eye strain, and neck fatigue from extended use. This has been reported in previous studies where a HoloLens was used[55], and was also mentioned in the feedback from user testing that I carried out by at least one of the participants. As such, it may be common for a user to take breaks and remove the headset whilst they are using it. I wish to see what effect removing the HoloLens would have on the hologram stability and tracking if it were removed and placed upside-down onto a surface whilst my shot assistant application was running.

(a) Initial position.



(b) Position after removing HoloLens once.

(c) Position after removing HoloLens twice.

(d) Position after removing HoloLens thrice.

(e) All balls from the above images superimposed.

Figure 4.3: Shown above are the positions of the holographic ball after removing the HoloLens multiple times. These positions have been superimposed in image (e) to allow for easier recognition of any drift that has occurred.

Figure 4.3 shows the perceived position of the holographic ball from the left hand side of the table after removal of the HoloLens up to three times, each time placing the headset down in a different location in the room. Panel (e) depicts these positions superimposed on top of one another and as we can clearly see, removing and replacing the HoloLens causes little-to-no holographic drift. The only assumption made with this is that the user removed and replaced the HoloLens from approximately the same place. Whether moving about the room and replacing the headset in a different location would effect drift is unknown. However, as the room itself has not changed, and the headset is moving whilst being

placed down, it is unlikely that replacing the headset in a different location would cause any additional holographic drift than perceived when simply moving around the table.

4.1.4 Change in Headset Height

The final stability test focuses on another common movement a player has when playing pool or snooker - leaning over to take the shot. When doing this, the user's head height changes, and as such, so will the height of the HoloLens whilst the user is using my application. Any holographic drift that occurs under this action could have a large effect on the user's shot accuracy, as a change in the cue ball's position will also change the position of the cue marker, causing the user to strike the cue ball in the wrong place.

(a) Superimposition of ball positions when standing and crouched - Position 1.

(b) Superimposition of ball positions when standing and crouched - Position 4.

Figure 4.4: Shown above are two repetitions of marking the holographic ball when standing and when crouched over (replicating the user taking a shot). The two stances have been superimposed in each repetition to allow for easier recognition of any drift that has occurred.

To evaluate this drift, I stood in 6 locations around the table and marked with a real ball where I perceived the holographic ball to be on the table. I then bent over into the shot-taking position and placed the ball again to mark the holograms position. Results from two of these tests can be seen in Figure 4.4, with each panel showing the standing and crouched ball positions superimposed on top of one another. Interestingly, when performing the task in position 1, shown by panel (a), there is some visible drift occurring similar to the amount seen when walking around the table. In comparison, whilst in position 4, shown by panel(b), there is no visible drift present. One explanation for this is that as more positions were tested out, the HoloLens was able to track more of the room and so the stability of

the holograms was greater as time went on. Therefore, there could be a substantial benefit for the user to walk around and explore the room before interacting with the application in order to increase tracking data and improve hologram stability.

Overall we can see that in certain conditions the holographic material displayed will drift, despite the efforts to reduce this in Section 3.4. There are some things that a user can do to reduce this though, such as remaining in a single room whilst using the application, exploring and looking around the room before starting the application, and aiming the shot from the location they wish to play it from.

4.2 Post Collision Path Estimation Accuracy

Being the major feature of the application, it is essential to ensure that the path drawing and path estimation system is accurate and consistent. Otherwise, the application would not provide any benefit to a user and would be rather useless. To verify its accuracy, a similar method was used to the one described in Section 4.1, again due to the offset present when recording holographic content. In total, five shots were performed to test different aspects of the path estimation system such as path accuracy over distance, path accuracy at a variety of contact angles, and path accuracy after rebounding off of a rail. The shots taken were a cut-back shot, a cross table long shot, a rebound shot, a double rebound shot, and a thin cut shot. The method for evaluating the path accuracy is as follows:

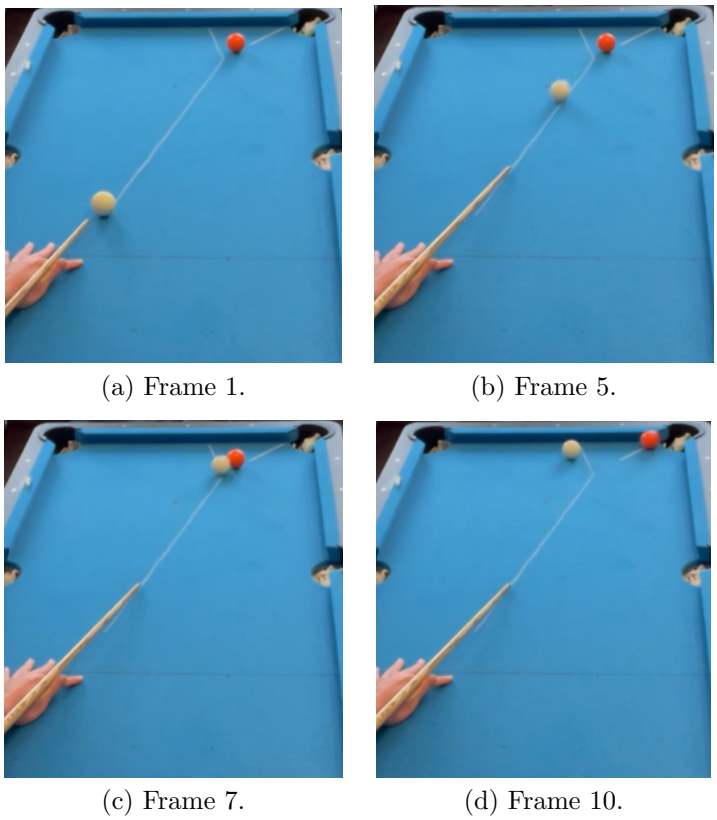
1. A camera was placed in a single position throughout testing such that it had a clear view of the entire table.
2. After starting my application and aligning the table model, I placed the required holographic balls onto the table in position for the shot that was being tested.
3. The cue marker was then moved around the cue ball to line up the shot and display the path lines on top of the table.
4. In order for the camera to see where the estimated path lines were drawn, chalk was used to mark the lines onto the real table.
5. Real ball(s) were then placed into the position of their holographic counter parts, and the shot was taken.
6. The shot was then re-set and performed another two times to ensure consistent results.

From the subsequent camera recordings, the series of frames from the shot could be analysed in order to see where the balls travelled, and whether or not they followed the chalk lines drawn onto the table.

Figures 4.5 and 4.6 show frames from the cross table shot and cut-back shot respectively. From these we can analyse whether the path estimation system is accurate over long distances as well as thin contact angles. As we can see from both sets of frames, prior to collision with the orange object ball the cue ball moves along the chalk line exactly, illustrating good accuracy in the path estimation from where the application tells the user to strike the cue ball. Once the orange object ball is struck, the cue ball seems to run slightly off from where it is estimated to go. Whilst this is not as accurate as it could be, I do not see this as detrimental to the application as it is only marginally off the predicted path and will not hinder the user from potting their targeted object ball. Additionally, it is not too surprising due to the approximations that are used to estimate it's post collision angle, as discussed in Section 3.8.

Despite the slight deviation of the cue ball compared to its estimated path post collision, the orange object ball follows its estimated post collision path exactly over a range of contact angles. This can be seen in both Figure 4.5 and Figure 4.6, and gives us conclusive evidence that the object ball's post collision path estimation is extremely accurate. For an application that is helping people to improve their potting accuracy, this is extremely important to the application's success.

Assessing the accuracy of rebound path estimations, Figure 4.7 shows frames from a recording of single rebound shot being taken. Similarly to the cut-back and cross table shots, prior to any collision we see the cue ball following the predicted path exactly. However, unlike what was seen on previous shots, after colliding with the rail the cue ball is not sticking to the predicted path. Whilst this is not



(a) Frame 1. (b) Frame 5.

(c) Frame 7. (d) Frame 10.

Figure 4.5: Shown above are selected frames from the video taken of a cross table shot to show the path taken by the cue ball and orange object ball.

ideal as it limits the types of shots that can be executed accurately, there is a simple explanation to the result. As discussed in Section 3.8, the post rail collision path estimation relies heavily on two values, the coefficient of restitution of the table rail and the speed of the cue ball before impact. The former value was calculated through analysing a rebound shot, but the latter was set to a fixed constant. This results in a situation where if the pre-collision speed of the cue ball isn't exactly $3.0ms^{-1}$ (the value it was fixed to in the collision algorithm) then the path estimation will be incorrect. Furthermore, for multiple rail collision shots (i.e. a double or triple rebound) the path estimation will be increasingly incorrect for every subsequent rebound as again, the pre-collision speed will be incorrect - reducing its accuracy with every collision. With each of the repetitions of the rebound shot, a different striking forced was used in order to try and get the cue ball to follow the chalk line. Whilst this did work in reducing how far away from the chalk line the cue ball was, the closest effort achieved was that seen in Figure 4.7, which is clearly still not accurate. As such, we can conclude that the rail rebound collision algorithm needs to be revised to account for varying cue ball speeds, as well as accounting for reduced ball speed after every rail collision.

(a) Frame 1.

(b) Frame 3.

(c) Frame 5.

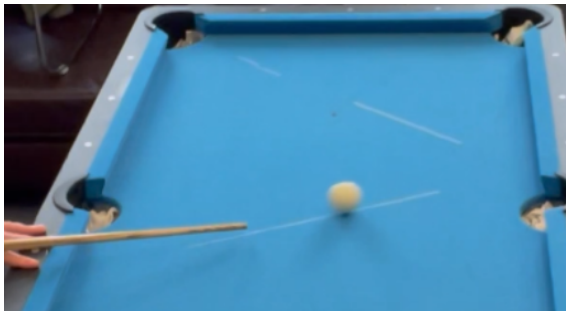
(d) Frame 6.

(e) Frame 8.

(f) Superimposition of all frames from shot video.

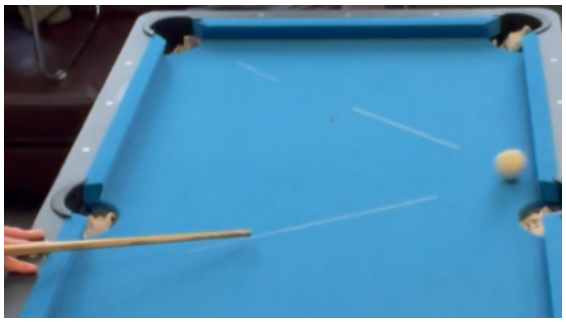
Figure 4.6: Shown above are selected frames from the video taken of a cut-back shot to show the path taken by both balls. Image (f) shows all 9 video frames superimposed to show the full trajectory of the ball.

(a) Frame 1.



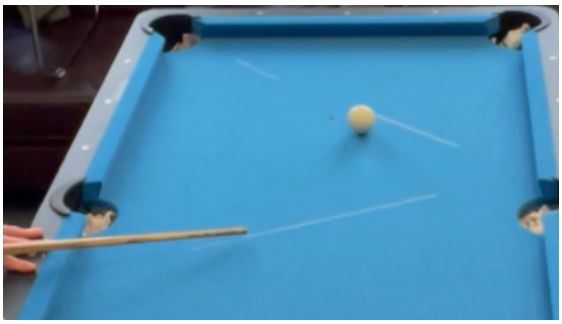
(b) Frame 4.

(c) Frame 6.



(d) Frame 7.

(e) Frame 11.



(f) Frame 12.

Figure 4.7: Shown above are selected frames from the video taken of a rebound shot to show the path taken by the cue ball.

4.3 User Shot Accuracy Improvement

Addressing arguably the most important project objective, in person user testing was performed in order to assess if the application actually improves a user's potting accuracy, and to what degree[1]. Each test user took a total of 7 pre-determined shots, such as the one shown in Figure 4.8, both with the HoloLens headset using my application and without any additional aid. The shots were designed to replicate some of the most common shots someone would play in an ordinary game of pool or snooker whilst also varying in difficulty level. The full list of the shots performed is as follows, with visual representations found in Appendix D.

- **Shot 1 -** A cross table long shot.
- **Shot 2 -** A half table thin cut shot into the centre pocket.
- **Shot 3 -** A single rebound off of a cushion into a ball to pot it.
- **Shot 4 -** A double rebound off of two cushions into a ball to pot it.
- **Shot 5 -** A cross table long shot where other balls must be avoided.
- **Shot 6 -** A cross table long shot, but with the object ball placed just in-front of the cue ball.
- **Shot 7 -** A cut back shot.

In order to try and mitigate any learning effect the user could experience just from playing pool (i.e. their accuracy improves throughout the test only due to them playing), half of the test users took the shots with my application first, then without, while the other half did the opposite.

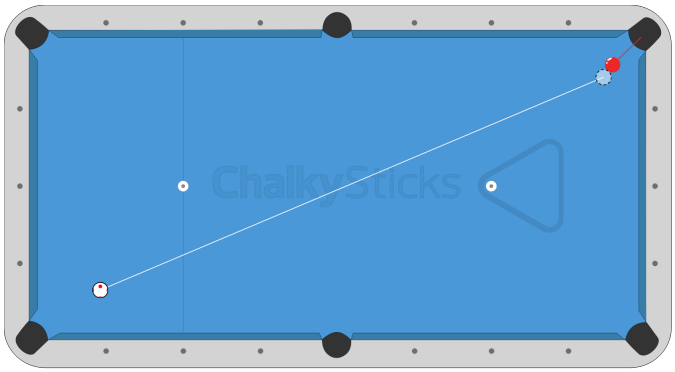


Figure 4.8: Shot 1 - An across table long shot. Graphic created using Chalky Sticks [1].

In total, four people tested my application, of whom two thought of their skill level as beginner (Users 3 and 4) and two thought of their skill level as intermediate (Users 1 and 2). It was ensured that one player from each skill level played the seven shots with my shooting assistant first, and one player from each skill level played the seven shots without my shooting assistant first. For each of the seven shots, the user was told which ball to target, which pocket to aim for, and which cushions to hit (if appropriate to the shot). A successful pot was recorded if all of the aforementioned shot conditions were met, otherwise, the shot was considered unsuccessful. Prior to taking the shots with the HoloLens on, each test user completed the *Learn Gestures* application that comes pre-loaded on the HoloLens[24] to ensure that they were able to get to grips with how to correctly interact with the device. A full breakdown of all of the test user's results can be found below in Table 4.1 and Table 4.2.

It is worth analysing a few of the unsuccessful shots the intermediate users had whilst using my shot assistant from Table 4.1. With User 1, they failed to pot Shot D.1 with the headset on, but successfully potted it without using my shot assistant. At the time they explained *"The cue ball seemed to follow the*

[1]It is worth noting that due to the COVID-19 outbreak and subsequent social distancing precautions present at the time, only a few people were able to test the application safely whilst abiding to the government restrictions. Because of this, no substantial statistical analysis could be performed due to the small sample size, but, we can still evaluate any results that were collected and determine trends within the data.

	User 1		User 2	
	With Headset	Without Headset	With Headset	Without Headset
Shot D.1	X	✓	✓	✓
Shot D.2	✓	✓	X	✓
Shot D.3	X	X	X	✓
Shot D.4	X	X	X	X
Shot D.5	✓	✓	✓	✓
Shot D.6	X	X	✓	✓
Shot D.7	X	X	X	X

Table 4.1: Results from the user accuracy test for both intermediate skill level users. A check mark indicates the shot was potted successfully, a cross indicates the shot was not potted successfully. Results from the user using my shot assistant application (indicated by the *With Headset* column) and not using my application (indicated by the *Without Headset* column) are presented.

	User 3		User 4	
	With Headset	Without Headset	With Headset	Without Headset
Shot D.1	✓	X	✓	X
Shot D.2	✓	X	X	X
Shot D.3	X	X	X	X
Shot D.4	X	X	X	X
Shot D.5	✓	X	X	X
Shot D.6	✓	✓	✓	✓
Shot D.7	X	X	✓	X

Table 4.2: Results from the user accuracy test for both beginner skill level users. A check mark indicates the shot was potted successfully, a cross indicates the shot was not potted successfully. Results from the user using my shot assistant application (indicated by the *With Headset* column) and not using my application (indicated by the *Without Headset* column) are presented.

line exactly." which could point us towards the assumption that the user miss-aligned one of the balls, the table, or simply aimed incorrectly. Additionally, this was the first shot that they played with my shot assistant and so they could have still been getting used to using it. However, given that the other three test users all potted Shot D.1 whilst using my shot assistant, I think this is unlikely. Given a larger sample size, it would be interesting to see if this was a common theme throughout.

The two shots that User 2 failed to pot whilst using my shot assistant, but succeeded in potting when not using my application, were Shot D.2 and Shot D.3 - a very thin cut shot and a rebound shot respectively. Concerning Shot D.2, very precise aim is required in order to successfully make this shot. Given that all users commented on it being somewhat difficult to place the balls and table, as well as how sensitive the cue marker was to move, I could attribute this unsuccessful pot to some minor mis-alignment. Further reinforcing this is the fact that overall, only two out of the four test users successfully potted Shot D.2 whilst using my shot assistant, one from each skill level. The same conclusion of object misalignment could be made for Shot D.7, another difficult thin cut shot, with only one out of the four test users successfully potting this shot with or without the use of my shot assistant - noting that this single successful pot for Shot D.7 was completed whilst using my shot assistant, showing that when set-up correctly, it can help a user make very difficult shots.

Moving onto Shot D.3. Being a rebound shot, it is notoriously difficult for even highly skill players to pot successfully. This is reflected in my results with only a single successful pot for either of the two rebound shots (Shot D.3 and Shot D.4) attained from all test users. Given that from Section 4.2 it was discussed that the calculation of post cushion collisions was inaccurate, and relies on assumptions about the cue ball's speed, this result is not too surprising. However, being able to hit rebound shots successfully is a highly important skill in getting out of snookered[2] positions, and so the fact that my shot assistant cannot currently provide accurate help in these situations is somewhat disappointing.

[2] A situation where the cue ball position is such that one cannot directly hit the required object ball.

After all shots were taken, both intermediate players saw a drop in how many shots they potted when using the shot assistant compared to not using it. User 1 saw a 33% drop, going from 3 pots to 2, whilst User 2 saw a 40% drop, going from 5 pots to 3. All shots that were potted whilst using the headset were also potted without for both users. Conversely, for both beginner players an increase in potting success was seen when using my shot assistant compared to not using it. User 3 had an increase of 300%, going from 1 successful pot without the headset to 4 successful pots with the headset, and User 4 had a 200% increase in potting success, going from 1 successful pot without the headset to 3 successful pots with the headset. From these results we can see that my shot assistant application seems to benefits less skillful players much more than someone with previous playing experience.

One explanation of why the system benefits less skillful players more could be attributed to more skillful players sub-consciously not following the shot assistant exactly. Perhaps applying their own adjustment to the shot, higher skilled players would have their own judgement of where to aim which could effect where they hit the cue ball, resulting in an unsuccessful pot. On the other hand, someone who doesn't play cue sports often will not have such pre-conceptions and so could be more trusting of where the shot assistant is showing them to hit.

However, perhaps a more likely explanation as to why less skillful players performed better with my shot assistant is to do with the shots themselves. The shots that saw a 75% or greater pot success when using my application were Shots D.1, D.5, and D.6. All three of these shots do not require a thin cut or a rebound in order to successfully pot the target ball and so are the simplest of the seven. Because of this, a majority of the users were able to line up these shots correctly and successfully make the pot with the use of my shot assistant. However, when it came to potting the same shots without the shot assistant, only the skillful players tended to make the shots due to their experience. Whilst this still shows the usefulness of my application to beginner or less skillful players, it again highlights the major issues with the current version of the shot assistant which is hindering it from being useful to a wider range of users.

4.4 User Interface and Ease of Use

After each user had taken the seven shots, once with my shot assistant and once without, I asked them a series of questions in order to gauge how easy they found the application to use, how useful they thought it was, as well as gaining any feedback and general comments they had. To do this, I came up with a set of questions myself (See Appendix E) which mainly focuses on intuitiveness and general feedback, as well as using the NASA Task Load Index (TLX) survey in order to assess the user workload when interfacing with my application in six different areas[45]. These areas are Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration. For all of these areas except performance, a lower score is better as it indicates less effort needed. For performance, a higher score is better as it indicates how successful they were in the task they were asked to do. Table 4.3 shows the results of the NASA TLX, while the transcripts associated with my own questionnaire can be found in Appendix F.

	User 1	User 2	User 3	User 4	Mean Score
Mental Demand	2	5	4	17	7
Physical Demand	8	5	1	3	4.25
Temporal Demand	0	0	0	2	0.5
Performance	15	13	15	10	13.25
Effort	12	12	12	15	12.75
Frustration	2	5	2	15	6

Table 4.3: Test user's scores for each of the fields of the NASA-TLX survey. Each result has a value between 0 (Very Low) and 20 (Very High).

First looking at the results from Table 4.3, we can see that on average, users thought that there wasn't too much mental or physical demand whilst using my shot assistant. This denotes that my shot assistant was intuitive to interact with, requiring very little thought or exertion to operate. Looking at User 4 however, they seemed to find the application very mentally demanding to use compared to the other three test users, which is also the case with their score for frustration - much higher than the other test user scores. From their transcript it is evident that this frustration and mental exertion came from trying to line up the shot exactly with the cue marker and side buttons. Whilst this was also noted by

the other three test users in their interviews, it seemed to not frustrate them as much. The difficulty that all users faced in properly aiming the shot is reflected in the effort metric, with all four users scoring 12 or higher, indicating that a decent amount of work was required to complete their shots. As such, it is evident that the way in which users line up a shot needs to be revised in order to make the application easier and less frustrating to use.

As there was no time limit on the user's taking their shots, it is unsurprising that the lowest scoring metric was temporal demand. All users seemed to think that the application aided their performance in potting the balls across the shots, as seen by the performance metric - the highest scoring on average. This is solidified from comments made in all of the user interviews, where all said that they found my application useful and helped them to perform the activity better. Despite User's 1 and 2 performing worse with the use of the application, they still thought that the shot assistant application allowed them to successfully accomplish what was asked of them and gave them a better idea of exactly where to aim. Overall, the scores from the NASA-TLX survey indicate that my application is intuitive to use and is found useful to some degree by all test users. It has however highlighted that it is not the easiest to use, with all users requiring some degree of effort in order to accomplish their task of potting the balls.

Looking through the transcripts of the questions asked to each user after they had used my application sheds some light into what aspects exactly were more difficult to use. Table 4.4 shows each of the test user's scores for the numerical based questions they were asked (Q1-Q5 in Appendix E), along with the mean average of those scores. Here it is evident that most found the table model difficult to align properly, unlike placing the holographic balls which most found easy to do. Although not a low score, the intuitivity and navigation of the application could have been improved by providing a more useful help information panel, which received the second lowest score. From the transcripts to later questions, it is evident that some found the help information difficult to read with the headset on because of the text size and the way in which is it presented. Despite the user's score suggesting that they didn't have too much difficulty in lining up shots, some of the other feedback gained suggests otherwise. All of the users commented on the cue marker being very sensitive to move by grabbing it, but appreciated the plus and minus degree buttons for easier aiming.

	User 1	User 2	User 3	User 4	Mean
Q1. Help information usefulness	5	6	8	3	5.5
Q2. Intuitivity and Navigation	6	6	8	7	6.75
Q3. Table placement	4	4	7	1	4
Q4. Ball placement	7	8	9	5	7.25
Q5. Lining up a shot	7	5	6	6	6

Table 4.4: Each user's rating for the first five questions of the post testing questionnaire. The final column shows the mean average of the scores.

Questions 6 through 9 asked each user if they found the application useful, what they liked and disliked the most, as well as providing an opportunity for any general comments or feedback. As reflected from the scores to questions 1 through 5, the aspects that user's disliked the most were positioning the table and using the cue marker to line up the shots. Additionally, from the general feedback, half of the users commented on the HoloLens being uncomfortable to wear, being in the way, as well as having a very narrow field of view, meaning that they couldn't see all the holograms that were in their eye's peripheral vision at once. Whilst this isn't feedback against my application itself, it does showcase some of the current limitations and problems with mixed reality technology that will hopefully be improved upon in the future. Moving onto what the users liked the most about the shot assistant, all said that they really liked the path being displayed of where the balls will go as it helped them to visualise what was going on. Additionally, most said that they enjoyed using the application, despite its frustrations, and said that they would use it again.

Overall, the user responses to the post task questions suggested that the application was useful, enjoyable, and fairly intuitive to use, but did have some pitfalls. It is clear that work needs to be done on making the table easier to place, making shots easier to line up, as well as displaying the help information in a more user friendly format. With these changes, as well as some progress made in mixed reality headset development making them more comfortable to wear and use, my shot assistant application would become both usable and useful for everyone.

Chapter 5

Conclusion

The main aim of this project was to create an application that could be used to help improve shooting accuracy in pool or snooker. Throughout this thesis I have detailed the process that has been carried out in order to create such a system. After looking at why mixed reality technology was chosen as the basis of this application in Section 1.2, research was done into the Microsoft HoloLens, how it worked, and how to develop for it seen, in Section 2. After collecting the material needed to estimate post collision directions, seen in Section 2.5, as well as looking at previous similar works for inspiration in Section 2.1, the application was successfully created and deployed using Unity Studios as outlined in Section 3. Here it was detailed how the application operates, how a user interacts with the system, as well as how various algorithms work, such as the post collision path estimation in Section 3.8. Extensive testing of the HoloLens and my application provided me with the knowledge needed to assess the project's success. This consisted of testing the hologram stability in Section 4.1, testing how accurate our post collision estimation system is in Section 4.2, as well as having four users of varying cue sports experience test out the application in Section 4.3 and Section 4.4. Here it was outlined that the application was successful in predicting the main post collision paths, that users found the application intuitive to use, and that it was useful to cue sport beginners. But, it was also outlined that there were some areas for improvement, such as the help screen, table placement, and ball-to-rail collisions.

Listed below is a summary of my main achievements throughout the project :

- After researching the most useful features to include, I successfully created one of the first cue sports shot assistant applications for the Microsoft HoloLens with an intuitive user interface.
- I successfully implemented an accurate and efficient post ball-to-ball trajectory estimation system.
- My application substantially increased the shot accuracy of beginner pool players when compared to not using my application, successfully lowering the entry curve into the sport.
- I was able to demonstrate the usefulness that mixed reality technology can provide to cue sports training by collecting primary user test data and user feedback from my own experiments, as well as a widely used questionnaire - the NASA-TLX.
- I thoroughly evaluated the success of my project by proving the reliability and stability of my application and the Microsoft HoloLens.

5.1 Project Status

The current state of the project can reviewed and pulled from GitHub at the following URL:

https://github.com/FinnWilkinson/Individual-Project-and-Innovation-Case/tree/main/HoloLensPoolTool

A short video demonstration is also available to view from the following URL. It should be noted that the video offset mentioned in Section 4.1 is present in this recording:

https://youtu.be/DDLZG0abcgQ

Currently, the application is able to provide shot assistance to players when taking non-rebound shots and is therefore most useful to beginners, helping them to learn about shot dynamics and how to play shots. This was shown in Section 4.3 where both of the beginner users saw a considerable increase in pots made when using my shot assistant. Therefore, I believe that objective 4 was successfully achieved as my application reduces the learning curve present when first starting out in the sport by making it easier to pot balls. Whilst useful to beginners, my shot assistant was not as useful to people of higher skill levels, mainly due to the inaccuracy of the ball-to-rail post-collision path estimation, as demonstrated and described in Section 4.2. With objective 3 being to create an accurate shot assistant to improve a user's potting success, this lack of a complete path estimation system doesn't fully satisfy the objective. However, I believe that I was partially successful in achieving objective 3. This is due to the improved accuracy of the beginner players who used my shot assistant, and with only the cue ball slightly deviating from the post-collision estimated path as evidenced in Section 4.2, the ball-to-ball post-collision path estimation algorithm yields very good predictions.

As discussed in Section 4.4, my application was found to be intuitive to navigate and operate but not necessarily easy to use to complete the task at hand - potting balls. The major complaints from Section 4.4 were about using the system. Specifically, when aligning and placing the table model into place over the top of the real table, as well as lining up shots by dragging the cue marker around the cue ball, which users found very sensitive to move. Although both features of the application do work without error, they take too much effort to accomplish for what should be simple tasks. Therefore, I believe that objective 5 is also partially complete, as the application is intuitive, but not as easy to use as it should be. Perhaps these issues are partially caused by the limitations of the Microsoft HoloLens Generation 1, as with inbuilt object recognition and more sophisticated hand tracking, operation of these parts of the application could be natively easier.

Addressing objective 2, I believe I was successful in showing the benefits that MR brings to cue sports. By creating an intuitive and useful tool for beginners to the sport, with a clear potential to help more advanced players too, it demonstrates ways in which mixed reality technology can be utilised to help with specific tasks. I think that as the technology continues to advance, more sophisticated tools can be created by incorporating computer vision, machine learning, and deep learning, and not just for cue sports. This project has shown that the sky is the limit with mixed reality, and that I have hardly scratched the surface of what is possible given enough time. My application has shown that even with a small amount of information displayed to the user an improvement can be made to a persons skill. Something that is far more sophisticated has the potential to revolutionise the field in which it is used, possibly changing how many jobs or tasks are performed. Objective 1 was about exploring what MR is truly capable of, and I believe that my project, whilst not using the technology to its fullest, is a successful example of what is truly possible.

5.2 Future Plans

After evaluating the project, and given more time, I believe that there is much more work to be done on this project, including the aspects of the application that are not completely working or still require further improvement. One of the major parts of the application that does not work as desired is the ball-to-rail collision estimation algorithm. As such, the application is only useful for non-rebound shots. As first stated in Section 2.5, some assumptions are required in order to calculate the rebound path of the cue ball. The most crucial of these assumptions was the initial speed of the cue ball, which was fixed to a constant value. This caused the estimations to be consistently wrong, as seen in Section 4.2, and clearly further work is required to improve this. One solution could be to measure the user's shot speed, perhaps akin to Medved's[17] application, and average this out over the course of them playing many shots in order to improve the accuracy over their playing time. Another solution could be to utilise a slider to indicate how hard the user intend to hit the shot, which would subsequently change the post rebound path. If I were to do the project again, I would be inclined to initially implement the latter of these solutions first, as having a static value for the shot speed clearly hindered the accuracy of the ball-to-rail collision estimations.

Another area for improvement is aiming the shot using the cue marker. All test users commented on this being very sensitive to move in Section 4.4, despite efforts made reduce this as detailed in Section 3.5. Experimenting with different cue markers and further evaluation could help to resolve this issue. As

discussed in Section 4.4 and Section 4.2, improving how the user lines up a shot, and improving the post-rebound collision estimations, would vastly improve the current version of my shot assistant, allowing it to be used on more complex shots such as rebounds. Subsequently, this would make the application not only useful to beginners, but to more experienced players who want to practice safety, positional, and rebounds shots too.

The last clear area that was not finalised in the application, and that received poor feedback in Section 4.4, is placing the table model. Whilst this does currently work, all user's found it difficult to place the table model correctly and accurately, often finding themselves fighting over two different corners being misaligned. Work had been started to improve this by using ArUco markers, an extension to the OpenCV package[9], in order to automatically place the table model into position. I was successful in being able to detect the markers from a live video feed of the HoloLens' photo-video camera, and transform a cube object to follow the detected marker as I moved. However, I was not able to get the cube hologram to properly align with the marker in time for this project deadline. Given more time, this feature could have been completed and evaluated to determine its usefulness and improvement over the current solution.

Continuing down the computer vision route, having the balls on the table automatically detected and placed would remove a task that the user currently has to perform. By using a You-Only-Look-Once(YOLO) real-time object detection algorithm, such as the one proposed by Huang et al.[13], could allow the HoloLens to detect every ball on the table in real time. Coupled with Canny edge detection from OpenCV to detect the playing surface of the table [8], a system for automating ball placement could be fabricated and added to the application. Additionally to automated table and ball placement, expanding the application to be used on, and recognize, any sized table would increase its usability. Currently, although the table model can be resized in application, it is set to the dimensions of the home pool table I was using for testing and development. By again utilising the Canny edge detection from OpenCV, the application could be made to recognise any sized table, allowing someone to use it on any table and for any cue sport.

Another set of useful features that would improve the applications usability are concerned with way that the user aims their shot. First, the application could be made to line up a shot automatically by the user selecting the ball they wish to pot and the pocket they wish to hit it into. This would educate beginner players by allowing them to visualise where the ideal place to aim certain shots is, as well as allowing them to concentrate on potting rather than aiming. Lastly, similar to many of the projects reviewed in Section 2.1, a more intuitive aiming system could be developed where the users cue tip is tracked, perhaps using ArUco markers as demonstrated by De Paolis et al.[41]. With this, the user could simply line up the cue to the cue ball and the application would draw on the estimated path if the user where to strike there. Both of these proposed future additions would help to combat the current user frustration when aiming shots, as discussed in Section 4.4, as well as showcasing further what mixed reality technology and the Microsoft HoloLens are truly capable of.

Bibliography

[1] Chalky sticks pad. https://pad.chalkysticks.com/. Accessed 10th April 2021.

[2] Airbus. Virtual reality with real benefits. https://www.airbus.com/newsroom/news/en/2017/09/virtual-reality-with-benefits.html. Accessed 10th May 2021.

[3] David Alayón. The age of mixed reality. https://medium.com/future-today/the-age-of-mixed-reality-7689fa403a90. Published 21st February 2018.

[4] Dave Alciatore. Pool physics property constants. https://billiards.colostate.edu/faq/physics/physical-properties/. Accessed 6th April 2021.

[5] Dave Alciatore. Typical pool ball speeds. https://billiards.colostate.edu/faq/speed/typical/. Accessed 8th April 2021.

[6] David Alciatore. Rolling cue ball deflection angle approximations. https://billiards.colostate.edu/bd_articles/2011/nov11.pdf. Released November 2011.

[7] Ricardo Alves, Luís Sousa, and J.M.F Rodrigues. Poolliveaid: Augmented reality pool table to assist inexperienced players. *21st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS Association*, pages 184–193, 2013.

[8] Anon. Canny edge detection. https://docs.opencv.org/master/da/d22/tutorial_py_canny.html. Accessed 10th May 2021.

[9] Anon. Detection of aruco markers. https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html. Accessed 11th May 2021.

[10] Tyler Craig. The line. https://www.cuedrills.com/drills/pool-drills/the-line/. Accessed 9th May 2021.

[11] Amber Garage. Holokit. mixed reality for everyone. https://holokit.io/. Accessed 10th May 2021.

[12] Susan Hooker, Mick Jennings, Jean Littlewood, Bronwen Moran, and Laurence Pateman. *Edexcel AS and A Level Modular Mathematics - Mechanics 4*. Pearson, 2009.

[13] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510, 2018.

[14] Mordor Intelligence. Mixed reality market - growth, trends, covid-19 impact, and forecasts (2021 - 2026). https://www.mordorintelligence.com/industry-reports/mixed-reality-market. Accessed 10th May 2021.

[15] Magic Leap. Magic leap 1. https://www.magicleap.com/en-us/magic-leap-1. Accessed 10th May 2021.

[16] Hawke-Eye Innovations Ltd. Hawk-eye in snooker. https://www.hawkeyeinnovations.com/sports/snooker. Accessed 12th March 2021.

[17] Sarah Medved. Augmented reality billiards assistant. 2020. *Williams Honors College, Honors Research Projects*. 1104.

[18] Microsoft. Coordinate systems. https://docs.microsoft.com/en-us/windows/mixed-reality/design/coordinate-systems#spatial-coordinate-systems. Accessed 13th May 2021.

[19] Microsoft. Focus point in unity. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/focus-point-in-unity. Accessed 30th April 2021.

[20] Microsoft. Headset tracks incorrectly because the environment has changed significantly over time. https://docs.microsoft.com/en-us/windows/mixed-reality/design/coordinate-systems#headset-tracks-incorrectly-because-the-environment-has-changed-significantly-over-time. Accessed 7th May 2021.

[21] Microsoft. Headset tracks incorrectly due to dynamic changes in the environment. https://docs.microsoft.com/en-us/windows/mixed-reality/design/coordinate-systems#headset-tracks-incorrectly-due-to-dynamic-changes-in-the-environment. Accessed 7th May 2021.

[22] Microsoft. Hologram stability - frame rate. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/hologram-stability#frame-rate. Accessed 20th May 2021.

[23] Microsoft. Hololens 1 gestures for authoring and navigating in dynamics 365 guides. https://docs.microsoft.com/en-us/dynamics365/mixed-reality/guides/authoring-gestures. Accessed 9th May 2021.

[24] Microsoft. Hololens 1 gestures for authoring and navigating in dynamics 365 guides. https://docs.microsoft.com/en-us/dynamics365/mixed-reality/guides/authoring-gestures#need-a-tutorial-on-gestures. Accessed 4th May 2021.

[25] Microsoft. Hololens (1st gen) hardware. https://docs.microsoft.com/en-us/hololens/hololens1-hardware. Accessed 13th May 2021.

[26] Microsoft. Hololens environment considerations. https://docs.microsoft.com/en-us/hololens/hololens-environment-considerations. Accessed 7th May 2021.

[27] Microsoft. Input overview. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/features/input/overview. Accessed 14th May 2021.

[28] Microsoft. Inside-out tracking. https://docs.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/tracking-system. Accessed 13th May 2021.

[29] Microsoft. Install the tools. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/install-the-tools?tabs=unity. Accessed 16th May 2021.

[30] Microsoft. Introducing mrtk for unity. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/mrtk-getting-started. Accessed 14th May 2021.

[31] Microsoft. Mixed reality capture for developers. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/mixed-reality-capture-for-developers. Accessed 30th April 2021.

[32] Microsoft. Mixed reality toolkit profile configuration guide. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/configuration/mixed-reality-configuration-guide. Accessed 16th May 2021.

[33] Microsoft. Render from the pv camera (opt-in). https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/mixed-reality-capture-for-developers#render-from-the-pv-camera-opt-in. Accessed 5th May 2021.

[34] Microsoft. Spatial anchors. https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-anchors. Accessed 17th May 2021.

[35] Microsoft. Welcome to the mixed reality feature tool. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/welcome-to-mr-feature-tool. Accessed 16th May 2021.

[36] Microsoft. What is the mixed reality toolkit. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/. Accessed 14th May 2021.

[37] Microsoft. What to expect when mrc is enabled on hololens. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/mixed-reality-capture-for-developers#what-to-expect-when-mrc-is-enabled-on-hololens. Accessed 30th April 2021.

[38] Franco Normani. The physics of billiards. https://www.real-world-physics-problems.com/physics-of-billiards.html. Accessed 6th April 2021.

[39] Nicholas Richard Jame Opperman. Snooker ball setting aided by image correlation. https://patents.google.com/patent/GB2440573A/en. Patent Number GB2440573A, Accessed 12th March 2021.

[40] Lily Ordineav. Why snooker is so popular in china. https://www.shockpedia.com/snooker-popular-china/. Accessed 10th May 2021.

[41] Lucio T. De Paolis, Marco Pulimeno, and Giovanni Aloisio. Different simulations of a billiards game. *Journal of Applied Quantative Methods*, 4(4):440–448, 2009.

[42] Damiel Rubino. Microsoft hololens - here are the full processor, storage and ram specs. https://www.windowscentral.com/microsoft-hololens-processor-storage-and-ram. Published 2nd May 2016.

[43] Kenzie Shofner. From the uk to china: The global transformation of snooker. https://www.unitedlanguagegroup.com/blog/from-the-uk-to-china-the-global-transformation-of-snooker. Accessed 10th May 2021.

[44] World Snooker. World snooker championship pockets record viewing figures. https://wst.tv/world-snooker-championship-pockets-record-viewing-figures/#:~:text=The%20World%20Championship%20achieved%20strong,reporting%20on%20Radio. Accessed 10th May 2021.

[45] Phil So. Nasa tlx: Task load index. https://humansystems.arc.nasa.gov/groups/tlx/. Accessed 5th May 2021.

[46] Unity Technologies. Colliders. https://docs.unity3d.com/Manual/CollidersOverview.html. Accessed 14th May 2021.

[47] Unity Technologies. Joints. https://docs.unity3d.com/Manual/Joints.html. Accessed 14th May 2021.

[48] Unity Technologies. Layers. https://docs.unity3d.com/Manual/Layers.html. Accessed 19th May 2021.

[49] Unity Technologies. Physics. https://docs.unity3d.com/Manual/PhysicsSection.html. Accessed 14th May 2021.

[50] Unity Technologies. Rigidbody overview. https://docs.unity3d.com/Manual/RigidbodiesOverview.html. Accessed 14th May 2021.

[51] Unity Technologies. Scripting. https://docs.unity3d.com/Manual/ScriptingSection.html. Accessed 14th May 2021.

[52] Unity Technologies. Unity. https://unity.com/. Accessed 14th May 2021.

[53] Unity Technologies. Wmr input and interaction concepts - world anchors. https://docs.unity3d.com/2018.2/Documentation/Manual/wmr_input_types.html. Accessed 7th May 2021.

[54] Unity Technologies. Wmr testing during development. https://docs.unity3d.com/2017.4/Documentation/Manual/wmr_testing.html. Accessed 14th May 2021.

[55] Eswara Rao Velamkayala, Manuel V. Zambrano, and Huiyang Li. Effects of hololens in collaboration: A case in navigation tasks. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 61(1):2110–2114, 2017.

[56] Wikipedia. Coefficient of restitution. https://en.wikipedia.org/wiki/Coefficient_of_restitution#:~:text=The%20coefficient%20of%20restitution%20(COR,be%20a%20perfectly%20elastic%20collision. Accessed 8th April 2021.

Appendix A

Simplification of Circle Equation

Detailed below is the mathematics behind the simplification of plugging the equation of a straight line into the equation of a circle.

Equation of the straight line : $z = mx + c$
Equation of the circle : $(x - x_b)^2 + (z - z_b)^2 = (2r)^2$

Plugging $z = mx + c$ into $(x - x_b)^2 + (z - z_b)^2 = (2r)^2$:

$$\begin{aligned}
(x - x_b)^2 + (mx + c - z_b)^2 &= (2r)^2 \\
x^2 - 2x_b x + x_b^2 + m^2x^2 + 2mx(c - z_b) + (c - z_b)^2 &= 4r^2 \\
x^2(1 + m^2) - 2x(x_b - m(c - z_b)) &= 4r^2 - x_b^2 - (c - z_b)^2
\end{aligned}$$

$$\text{let } t = (1 + m^2),\ u = 2(x_b - m(c - z_b)),\ v = 4r^2 - x_b^2 - (c - z_b)^2$$

$$tx^2 - ux - v = 0$$

This gives us the polynomial needed to solve for x using the quadratic formula.

Appendix B

Listings

Within this Appendix there are some of the longer listings referenced in the main body of the thesis.

```
using Microsoft.MixedReality.Toolkit;
using Microsoft.MixedReality.Toolkit.Boundary;
using Microsoft.MixedReality.Toolkit.Input;
using Microsoft.MixedReality.Toolkit.Utilities;
using Microsoft.MixedReality.Toolkit.UI;
using Microsoft.MixedReality.Toolkit.UI.BoundsControl;
```

Listing B.1: Useful dependencies to include in any script when developing for the HoloLens using the MRTK.

```
void Start() {
    ...
    table.GetComponent<ManipulationHandler>().OnManipulationStarted.
        AddListener((input) => ManipStart(input));
    table.GetComponent<ManipulationHandler>().OnManipulationEnded.
        AddListener((data) => ManipEnd(data));

    table.GetComponent<BoundsControl>().RotateStarted.AddListener(() =>
        RotateORScaleStart());
    table.GetComponent<BoundsControl>().RotateStopped.AddListener(() =>
        RotateORScaleEnd());

    table.GetComponent<BoundsControl>().ScaleStarted.AddListener(() =>
        RotateORScaleStart());
    table.GetComponent<BoundsControl>().ScaleStopped.AddListener(() =>
        RotateORScaleEnd());
    ...
}
```

Listing B.2: The listeners required to ensure objects attached to the table model are manipulated appropriately along with the table.

```
// Find reverse of inbound direction vector
Vector3 reverseVector = new Vector3(cuePos.x - rayHitPoints[0].point.x,
    0.0f, cuePos.z - rayHitPoints[0].point.z);

// Calculate Rotation
float newX, newZ;
float rotationAngle = -(Mathf.PI - alpha - beta);
// Check if need to rotate clockwise or anti-clockwise
if (normalDirAngle < 0) rotationAngle = -rotationAngle;
newX = (reverseVector.x) * Mathf.Cos(rotationAngle) - (reverseVector.z)
    * Mathf.Sin(rotationAngle);
newZ = (reverseVector.x) * Mathf.Sin(rotationAngle) + (reverseVector.z)
    * Mathf.Cos(rotationAngle);

// Update Direction Vector and its normal, y stays the same as balls all
     at the exact same height
dir = new Vector3(newX, dir.y, newZ);
dirNorm = Vector3.Cross(dir, Vector3.up);

// Store hit point
pathPoints[hitsMade] = rayHitPoints[0].point;
hitsMade++;
// Update raycast firing positions
currentPositions[0] = rayHitPoints[0].point;
currentPositions[1] = rayHitPoints[0].point - (dirNorm * radius);
currentPositions[2] = rayHitPoints[0].point + (dirNorm * radius);
// Continue round loop to calculate next collision/bounce
```

Listing B.3: The logic required to rotate the direction vector about the collision point to find the cue ball's post collision direction for ball-to-rail collisions. Other key values are also updated after the angle β has been found.

```
float[] potentialXcoords = new float[2], potentialZcoords = new float
    [2];
float t = 1 + (gradient * gradient);
float u = 2 * (hitBallCentre.x - (gradient * (intercept - hitBallCentre.
    z)));
float v = (4 * (radius * radius)) - Mathf.Pow(hitBallCentre.x, 2) -
    Mathf.Pow(intercept - hitBallCentre.z, 2);

// Use quadratic formula to get both possible x solutions
potentialXcoords[0] = -(((-u) + Mathf.Sqrt((u * u) - (4 * t * (-v)))) /
    (2 * t));
potentialXcoords[1] = -(((-u) - Mathf.Sqrt((u * u) - (4 * t * (-v)))) /
    (2 * t));

// Plug both x coordinates we found into z=mx+c to get both possible z
    coordinates
potentialZcoords[0] = gradient * potentialXcoords[0] + intercept;
potentialZcoords[1] = gradient * potentialXcoords[1] + intercept;

// Find which of the 2 possible points is closest to cue ball, and will
    therefore be the correct coordinates
if (Mathf.Sqrt(Mathf.Pow(potentialXcoords[0] - cueBallCentre.x, 2) +
    Mathf.Pow(potentialZcoords[0] - cueBallCentre.z, 2)) <
     Mathf.Sqrt(Mathf.Pow(potentialXcoords[1] - cueBallCentre.x, 2) +
         Mathf.Pow(potentialZcoords[1] - cueBallCentre.z, 2)))
     cueBallCollCentre = new Vector3(potentialXcoords[0], cueBallCentre.y
         , potentialZcoords[0]);
else cueBallCollCentre = new Vector3(potentialXcoords[1], cueBallCentre.
    y, potentialZcoords[1]);
```

Listing B.4: Finding the centre of the cue ball at the exact time it would collide with the object ball.

```
// Calculate direction of hit balls trajectory post collision (vector
   going from centre of cue ball position at collision through centre of
    ball we hit)
Vector3 hitBallDirection = hitBallCentre - cueBallCollCentre;
// Invokle object ball's line drawing function to draw correct post
   collision path (0 bounces, 1 collision)
closestBall.GetComponent<BallProperties>().DrawCollisionPath(
   hitBallDirection);

float angleBetween = Vector3.Angle(direction, hitBallDirection);
float rotationAngle = Mathf.PI / 2; // (90 degrees)
// If angle between cue ball direction pre collision and
   hitBallDirection < 15 degrees, cue ball will travel about 3x angle of
    hit ball to original vector
if (angleBetween < 15) rotationAngle = (angleBetween*3 * (Mathf.PI /
   180));

// If angle between cue ball direction pre collision and
   hitBallDirection is between 15 and 55 degrees, then cue ball will
   travel about 30 degrees from initial trajectory line
if (angleBetween >= 15 && angleBetween <= 55) rotationAngle = ((
   angleBetween + 30.0f) * (Mathf.PI/180));

// If angle between cue ball direction pre collision and
   hitBallDirection is > 55 degrees, cue ball will travel 70% of angle
   to tangent
if (angleBetween > 55) rotationAngle = (angleBetween + ((90 -
   angleBetween) * 0.7f)) * (Mathf.PI / 180);


// Rotate hit ball's post collision vector 90 degrees to get cue ball's
   post collision direction vector
float newX, newZ;
// If right side of cue ball hits, rotate anti-clockwise
if (ballHitR && closestBall == initialHits[2].collider.gameObject) {
    newX = hitBallDirection.x * Mathf.Cos(-rotationAngle) -
       hitBallDirection.z * Mathf.Sin(-rotationAngle);
    newZ = hitBallDirection.x * Mathf.Sin(-rotationAngle) +
       hitBallDirection.z * Mathf.Cos(-rotationAngle);
}
// If left side of cue ball hits (or centre hit), rotate clockwise
else {
    newX = hitBallDirection.x * Mathf.Cos(rotationAngle) -
       hitBallDirection.z * Mathf.Sin(rotationAngle);
    newZ = hitBallDirection.x * Mathf.Sin(rotationAngle) +
       hitBallDirection.z * Mathf.Cos(rotationAngle);
}

// Update cue ball's direction vector and firing position
dir = new Vector3(newX, dir.y, newZ);
```

Listing B.5: Finding the post-collision vectors for the cue ball and the object ball that intersects the cue ball's predicted path.

Appendix C

Experienced Player's Perspective Questionnaire

C.1 Questions

Below is the questionnaire sent out to selected members of the University of Bristol Pool and Snooker Club. The members were selected based on how long they have been playing cue sports for, their technical ability, their success in competitions, as well as any society positions undertaken that would have present them with a wider exposure to the sport.

1. Do you perform training drills either alone or with others?
2. Do you think training drills are an important part of cue sports development
3. What technique(s) do you believe are the most important to focus on when training?
4. Which one skill do you believe is the most important to train and develop in order to see the most improvement in a player?
5. Would you be interested in using a head mounted display when training that provides feedback and guidance on your shots whilst you play?
6. For such a device, what features or visual guidance would you find useful for it to have?
7. Who do you think would benefit most from such a system?

C.2 Responses

Below are the responses for the questionnaire shown above.

C.2.1 Response 1

1. Yes.
2. Yes.
3. Potting the target ball, positioning the cue ball optimally for the next shot, break Building, and complex / spin shots.
4. Hitting the target ball and potting, target ball, break building.
5. Yes.
6. I think it would be very useful to have some sort of memory function - take a snapshot of the table before taking a shot, so that if you want to replay the shot for some reason, you can set it up again exactly. I think another good feature would be some ability to record statistics, i.e. etc.
7. New players or people who play as part of a club regularly / compete regularly.

C.2.2 Response 2

1. Yes.
2. Yes.
3. Potting, positioning the cue ball, break Building, safety shots, spin shots.
4. Break building.
5. Possibly, it may be distracting whilst down on a shot.
6. I think it would be useful to practise thin safeties as it's very tricky to know which angle to play.
7. Anybody if the system was good. Would be very helpful for new players to first understand shots.

Appendix D

Pre-Defined Shots Used in Testing

Seen below are graphical representations of the shots each test user had to perform, both using the HoloLens application and without using it. In each of the shots, the user had to pot the red ball and bounce the cue ball (white ball) off of certain cushions (if required by the shot) whilst avoiding any yellow balls (if present in the shot).

All of the shot diagrams were made with the use of the Chalky Sticks Pad online application [1].

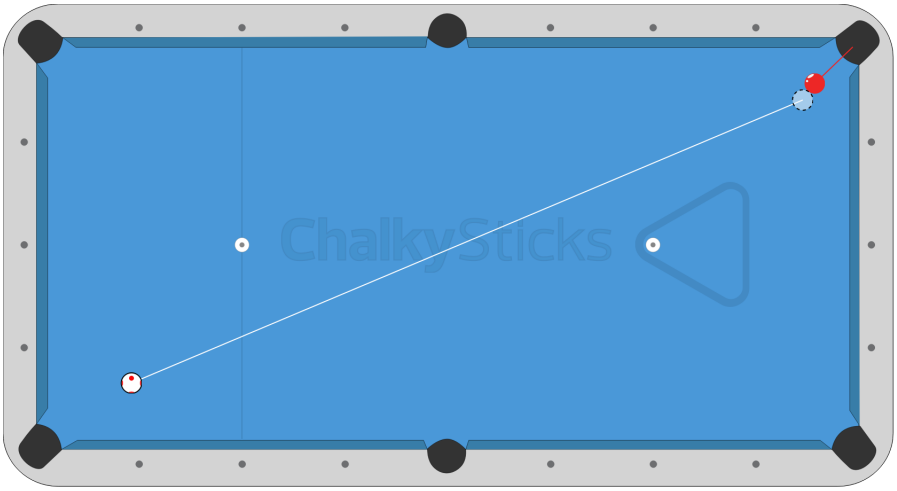


Figure D.1: Shot 1 - An across table long shot.

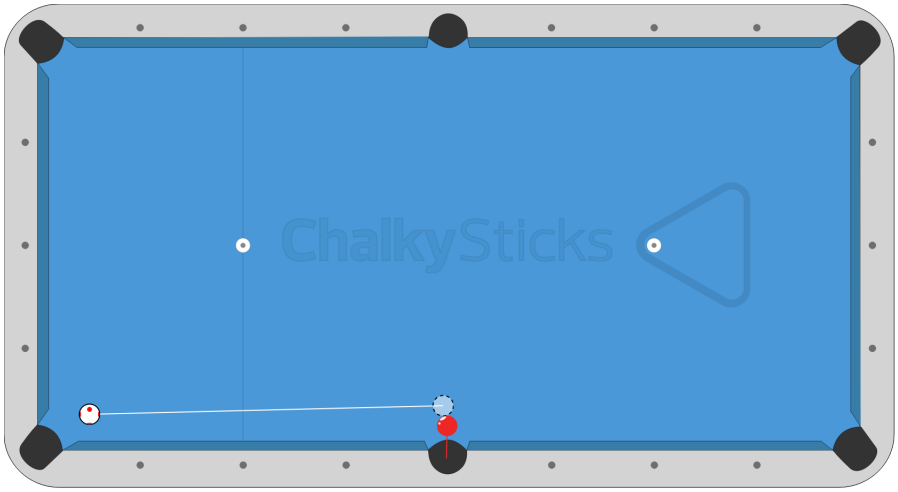


Figure D.2: Shot 2 - A tight clip shot.

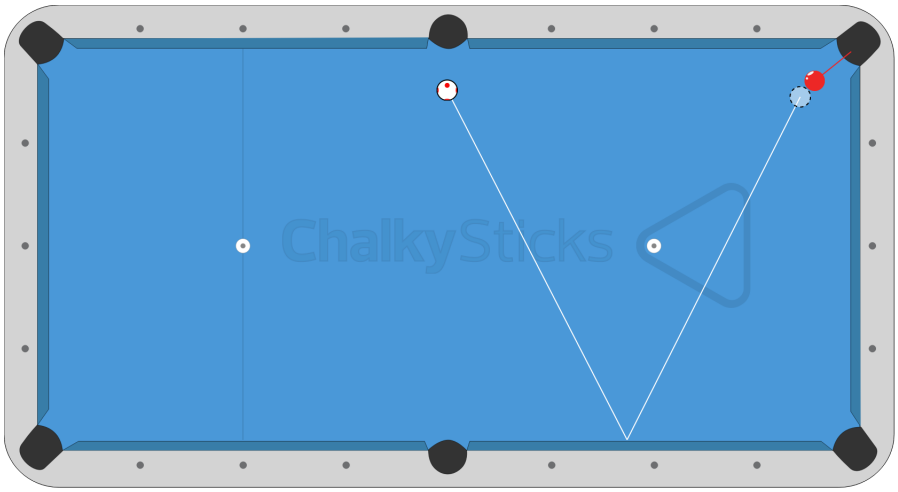


Figure D.3: Shot 3 - A single rebound shot.

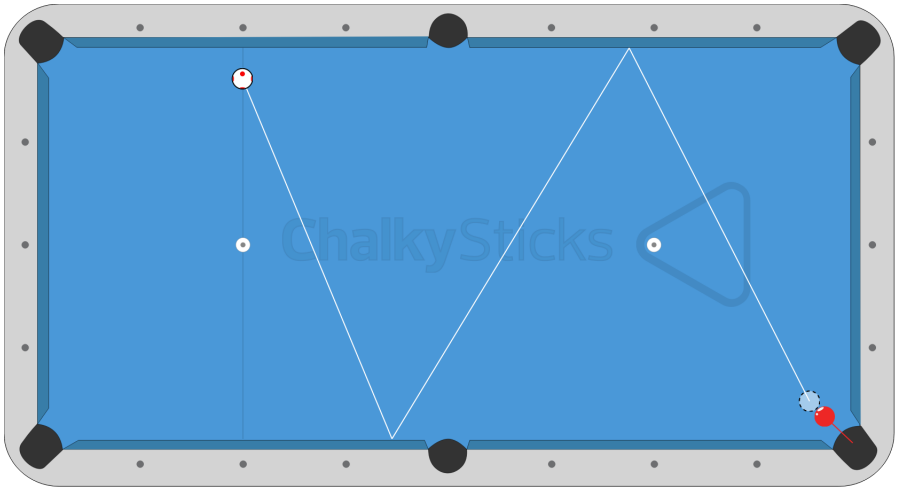


Figure D.4: Shot 4 - A double rebound shot.

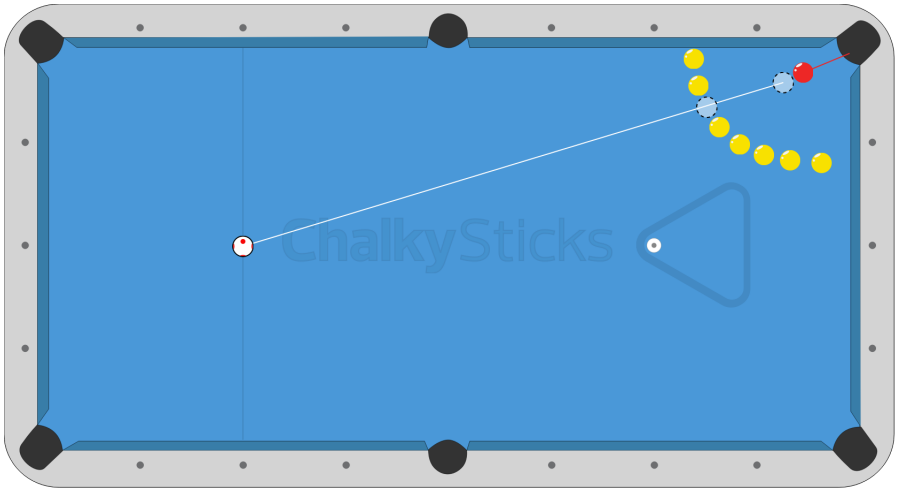


Figure D.5: Shot 5 - A long shot requiring precise accuracy.

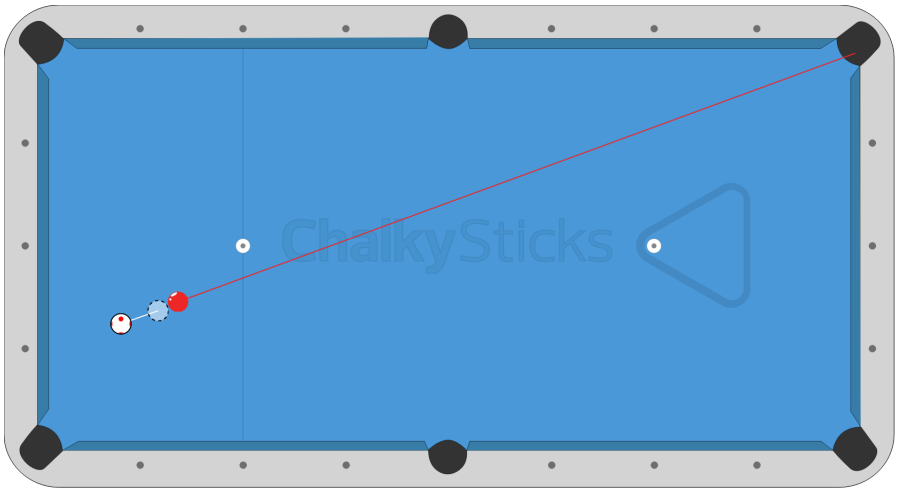


Figure D.6: Shot 6 - A long shot with the target ball placed close to the cue ball.

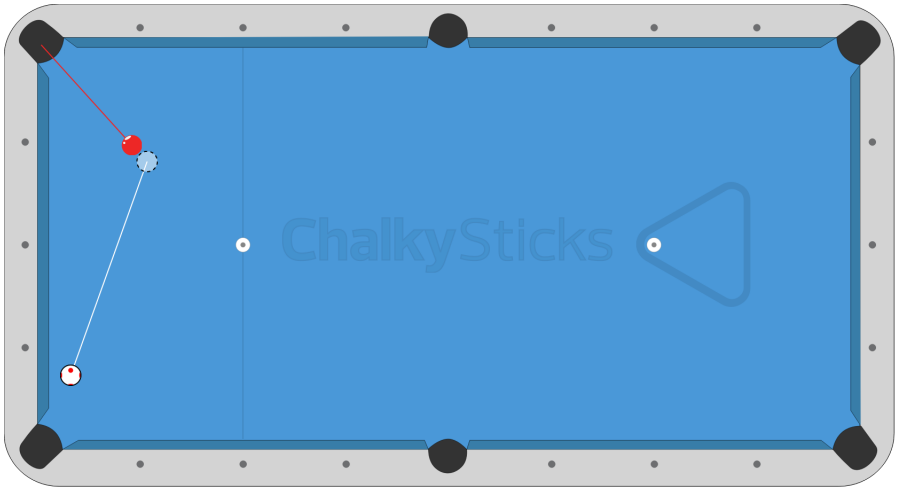


Figure D.7: Shot 7 - A difficult cut back shot.

Appendix E

Post User Testing Questions

Detailed below are the questions I asked to each test user after they had completed all seven shots with and without my shot assistant application.

1. From a scale of 1 to 10, how helpful did you find the Help Information inside the application? With 1 being not helpful at all, 10 being very helpful.
2. From a scale of 1 to 10, how intuitive was the application to use and navigate? With 1 being not intuitive at all, 10 being very intuitive.
3. From a scale of 1 to 10, how easy was it to place the table model into place? With 1 being not easy at all, 10 being very easy.
4. From a scale of 1 to 10, how easy was it to place the ball models into place? With 1 being not easy at all, 10 being very easy.
5. From a scale of 1 to 10, how easy did you find it to move the cue marker and line up a shot? With 1 being not easy at all, 10 being very easy.
6. Did you feel as though the application helped you to perform the activity better and was useful?
7. What did you dislike the most about the application?
8. What did you like the most about the application?
9. Any additional comments or feedback?

Appendix F

Post User Testing Transcripts

Below are the transcripts between myself and each of the test users after they had used the system and completed various different shots.

F.1 User 1

Myself :	Thanks for that. Can I ask you a few questions?
User 1 :	Yeah sure.
Myself :	Great, thanks. OK, from a scale of 1 to 10, how helpful did you find the Help Information inside the application? With 1 being not helpful at all, 10 being very helpful.
User 1 :	Uhh a five.
Myself :	Thank you. Next, from a scale of 1 to 10, how intuitive was the application to use and navigate? With 1 being not intuitive at all, 10 being very intuitive.
User 1 :	Six.
Myself :	OK. From a scale of 1 to 10, how easy was it to place the table model into place? With 1 being not easy at all, 10 being very easy.
User 1 :	A four, it was hard to line it up exactly.
Myself :	Thank you. From a scale of 1 to 10, how easy was it to place the ball models into place? With 1 being not easy at all, 10 being very easy.
User 1 :	Seven.
Myself :	Thank you. From a scale of 1 to 10, how easy did you find it to move the cue marker and line up a shot? With 1 being not easy at all, 10 being very easy.
User 1 :	Seven again.
Myself :	Great. Next, did you feel as though the application helped you to perform the activity better and was useful?
User 1 :	Yes, for more difficult shots I had a better idea of where I had to aim. For the easier shots I didn't really notice a difference.
Myself :	OK. Now, what did you dislike the most about the application?
User 1 :	Positioning the table is quite difficult as the table doesn't fit on the screen all at the same time, so you have to keep walking round adjusting each corner which messes up other bits.
Myself :	OK, and what did you like the most about the application?
User 1 :	The aim assist thing was good.
Myself :	Nice. And finally, do you have any additional comments or feedback?
User 1 :	I think the headset is uncomfortable and a bit off-putting but the system it self is intuitive and very helpful. Because you cant see everything at once you have to keep looking away from the ball to see where the target is. When you compare it to VR everything you see is hologram but having half hologram half normal world is a bit weird.
Myself :	Brill, that's it. Thanks again for helping me out and testing my app.
User 1 :	No problem.

F.2 User 2

Myself : OK, we are done testing wise. Can I ask you a few questions?
User 2 : Yeah go ahead.
Myself : Great. First off, from a scale of 1 to 10, how helpful did you find the Help Information inside the application? With 1 being not helpful at all, 10 being very helpful.
User 2 : A six.
Myself : Thank you. Next, from a scale of 1 to 10, how intuitive was the application to use and navigate? With 1 being not intuitive at all, 10 being very intuitive.
User 2 : Again a six.
Myself : From a scale of 1 to 10, how easy was it to place the table model into place? With 1 being not easy at all, 10 being very easy.
User 2 : Four.
Myself : From a scale of 1 to 10, how easy was it to place the ball models into place? With 1 being not easy at all, 10 being very easy.
User 2 : Eight.
Myself : OK, from a scale of 1 to 10, how easy did you find it to move the cue marker and line up a shot? With 1 being not easy at all, 10 being very easy.
User 2 : A five.
Myself : Next, did you feel as though the application helped you to perform the activity better and was useful?
User 2 : I thought knowing where the white ball will go was useful to know, but, I mean, I don't think it made me any better as I got more in without it aha.
Myself : Aha right, thanks. What did you dislike the most about the application?
User 2 : Moving the balls and cue marker was very sensitive and made it hard sometimes to place things accurately.
Myself : What did you like the most about the application?
User 2 : The white line showing where the balls will go, I thought it was really good!
Myself : OK and finally, do you have any additional comments or feedback?
User 2 : The headset feels like its in the way sometimes and it would be nicer if there was a bigger screen so you could see everything at the same time. I thought the app was good but a few things could be done to improve it.
Myself : What exactly do you think could be done to improve it?
User 2 : Maybe something to place the table and balls automatically? I think that would be the biggest thing.
Myself : Great that's it!. Thanks again for helping me out.
User 2 : My pleasure.

F.3 User 3

Myself : Thank you for testing out my application. Would I be able to ask you a few questions?
User 3 : Yes of course.
Myself : Great. Firstly, from a scale of 1 to 10, how helpful did you find the Help Information inside the application? With 1 being not helpful at all, 10 being very helpful.
User 3 : I'd say an eight. It helped me figure out what to do but could have been presented in a better way.
Myself : OK cool. Next, from a scale of 1 to 10, how intuitive was the application to use and navigate? With 1 being not intuitive at all, 10 being very intuitive.
User 3 : Again I'd say an eight.
Myself : OK thanks. From a scale of 1 to 10, how easy was it to place the table model into place? With 1 being not easy at all, 10 being very easy.
User 3 : A seven. It wasn't that hard but took a little while to line it up properly.
Myself : Thank you. From a scale of 1 to 10, how easy was it to place the ball models into place? With 1 being not easy at all, 10 being very easy.
User 3 : A nine, it was very easy to place them into position.
Myself : Thank you. From a scale of 1 to 10, how easy did you find it to move the cue marker and line up a shot? With 1 being not easy at all, 10 being very easy.
User 3 : Nine for sure. I found the plus 1 degree and minus 1 degree buttons really useful to finalise it's position after I got it nearly into place.
Myself : That's great! Next, did you feel as though the application helped you to perform the activity better and was useful?
User 3 : Yes it was very helpful, especially on shot 5.
Myself : OK, now, what did you dislike the most about the application?
User 3 : Moving the hit marker into position as it was very sensitive to any movement, even very small ones, and it was quite hard to grab onto sometimes.
Myself : OK thank you. What did you like the most about the application?
User 3 : The trajectory shown is quite accurate actually. It defiantly helped me to pot the balls better! The headset itself and making gestures like you're actually grabbing something is really cool too!
Myself : It's great isn't it! And finally, do you have any additional comments or feedback?
User 3 : I think making the hit marker less sensitive to movement and when you were taking a shot would be good. You also cant really see the rest of the table or where the ball will go when you're trying to take a shot as the screen is not aligned with your line of sight. But it make me want to play pool more in real life! I really enjoyed it!
Myself : That's great. Thank you again for testing out the application!
User 3 : You're welcome. It was great fun!

F.4 User 4

Myself : OK, we're all done. Can I ask you a few questions?
User 4 : Of course!
Myself : Great. From a scale of 1 to 10, how helpful did you find the Help Information inside the application? With 1 being not helpful at all, 10 being very helpful.
User 4 : A three, it was kinda hard to read.
Myself : Thank you. From a scale of 1 to 10, how intuitive was the application to use and navigate? With 1 being not intuitive at all, 10 being very intuitive.
User 4 : Seven.
Myself : From a scale of 1 to 10, how easy was it to place the table model into place? With 1 being not easy at all, 10 being very easy.
User 4 : A one, I found it very hard!
Myself : Thank you. From a scale of 1 to 10, how easy was it to place the ball models into place? With 1 being not easy at all, 10 being very easy.
User 4 : A five, much easier than the table!
Myself : From a scale of 1 to 10, how easy did you find it to move the cue marker and line up a shot? With 1 being not easy at all, 10 being very easy.
User 4 : A six, it was a bit sensitive but the buttons helped a lot.
Myself : Next, did you feel as though the application helped you to perform the activity better and was useful?
User 4 : Yes it was helpful as long I was able to line up the shot properly.
Myself : What did you dislike the most about the application?
User 4 : Having to move the objects, but most of all line up the shot by dragging the cue marker. The help text was also kinda hard to read.
Myself : And what did you like the most about the application?
User 4 : That it explains it very well and that I hit more balls with it on!
Myself : OK and finally, do you have any additional comments or feedback?
User 4 : I think that it is very smart application and if it was easier to use it would be more welcomed, but I wouldn't use it myself as I found it too hard to move things into place and line up the shots. I was wondering whether I was placing the ball correctly or not as there is a lot of human error so I put a lot of focus on moving the ball maybe more than I should have just so it lines up correctly. The lines did help to visualise the shot a lot. When I usually play I do imaginary lines to line up my shots but with the HoloLens it displays them for me which I found useful. It would be useful if it put everything in place for me! I also Started to get a headache.
Myself : Great. Thank you very much for your help and feedback.
User 4 : No worries! It was fun to use but a little frustrating aha.