Simo Ollinen

HoloLens
Development of mixed reality game

Bachelor's thesis
Information Technology / Game Programming

2019

Kaakkois-Suomen ammattikorkeakoulu

Tekijä	**Tutkinto**	**Aika**
Simo Ollinen	Insinööri (AMK)	Kesäkuu 2019

Opinnäytetyön nimi	
HoloLens Mixed reality -pelin kehittäminen	74 sivua 1 liitesivu

Toimeksiantaja

Kaakkois-Suomen ammattikorkeakoulu Oy

Ohjaaja

Lehtori Marko Oras

Tiivistelmä

Tämän opinnäytetyön päämäärä oli esitellä mixed realityn ja Microsoft HoloLens mixed reality -laitteen aihepiirejä. Lisäksi tavoitteena oli kehittää mixed reality -peli, jotta mixed realityn mahdollisuudet pelikehityksen kannalta saisi selvitettyä.

Opinnäytetyö kuvailee oleellisimpia mixed reality- ja HoloLens-aihepiireihin liittyviä kohteita pohjustaakseen vaatimukset toteutusta varten. Vaatimuksissa keskitytään erityisesti spatiaaliseen kartoittamiseen, pelaajan syötteen tunnistamiseen ja vuorovaikutukseen mixed reality -ympäristöissä. Lisäksi tämä opinnäytetyö tarjoaa vaiheistetun opastuksen mixed reality -pelin suunnitteluun ja toteutukseen.

Mixed reality -pelin kehitys toteutettiin Unity-pelimoottorin avulla. Pelin kehityksen prosessiin sisältyi myös useita lisätyökaluja. Lisätyökaluilla avustettiin tarvittavien peli-assettien tuottamista ja nopeutettiin pelin testausta. Tiettyjen logiikkalohkojen toteuttamista varten verrattiin tarkasti erilaisia menetelmiä, joista valittiin kaikkein oivaltavimmat.

Onnistuneen opinnäytetyön tuloksena kehittyi toiminnallinen mixed reality -peli, joka otettiin käyttöön HoloLens -laitteella. Peli tarjoaa oivallisen esimerkin spatiaalisen kartoituksen ja kädenliikkeen tunnistamisen hyödyntämisestä HoloLensillä. Lisäksi peli voi toimia mallina vaihtoehtoisille lähestymistavoille ja tarjota perustan tulevaisuuden kehitykselle.

Asiasanat

hololens, mixed reality, pelikehitys, spatiaalisuus, vuorovaikutus

XAMK
South-Eastern Finland University of Applied Sciences

Author	Degree	Time
Simo Ollinen	Bachelor of Engineering	June 2019

Thesis title

HoloLens
Development of mixed reality game

74 pages
1 page of appendices

Commissioned by

South-Eastern Finland University of Applied Sciences Ltd.

Supervisor

Marko Oras

Abstract

The objective of this thesis was to introduce the topics of mixed reality and Microsoft HoloLens mixed reality device, as well as to develop a mixed reality game in pursuit of studying the potentiality for game development purposes.

This thesis describes the essential subjects related to mixed reality and HoloLens topics in order to layout fundamental requirements for the implementation, focusing especially on spatial mapping, recognition of player input, and interaction in mixed reality environments. Additionally, this thesis provides an explanation of the design and implementation processes of the mixed reality game in a thorough step by step guide.

Development of the mixed reality game was the primary purpose of this thesis and was conducted by utilizing a game engine known as Unity. The development cycle included various additional tools to aid producing necessary assets and to accelerate the testing process of the game. Several types of methods for implementing certain blocks of logic were closely compared, of which the most insightful were chosen.

As a result, a functional mixed reality game was developed and deployed on the HoloLens device, marking the thesis project a success. The game provides a fine example of utilization of spatial mapping and gesture recognition with HoloLens. Additionally, the game can serve as a template for alternate approaches and provide basis for future development.

Keywords

game development, hololens, interaction, mixed reality, spatiality

CONTENTS

ABBREVIATIONS ... 6
1 INTRODUCTION ... 7
1.1 Objective of this thesis ... 7
1.2 Initial approach ... 8
1.3 About the commissioner ... 8
2 MIXED REALITY ... 8
2.1 Virtuality Continuum ... 10
2.2 Main display types ... 12
2.3 Free from the frame ... 14
3 HOLOLENS ... 17
3.1 The device ... 18
3.2 Input ... 20
3.2.1 Gaze ... 20
3.2.2 Gestures ... 21
3.2.3 Voice ... 25
3.3 Spatial mapping ... 26
4 DARTS MR ... 30
4.1 Game analysis ... 31
4.2 Story and characters ... 31
4.3 Gameplay overview ... 32
4.4 Control scheme ... 33
4.5 Game aesthetics and user interface ... 34
5 IMPLEMENTATION ... 35
5.1 Prerequisite tools ... 36
5.1.1 Visual Studio 2017 ... 37
5.1.2 Unity ... 38
5.1.3 Blender ... 39

5.1.4 HoloLens emulator 40
5.2 Setting up the project 41
5.3 Creation of fundamental game objects 43
5.3.1 Camera 44
5.3.2 GazeManager 45
5.3.3 SpatialMappingManager 46
5.3.4 GameManager 51
5.3.5 Cursor 55
5.3.6 Dart 56
5.3.7 Board 60
5.3.8 UI 63
5.4 Building and deploying the project 65
6 CONCLUSIONS 66
7 FURTHER DEVELOPMENT 67
REFERENCES 69
FIGURES 72
APPENDICES
Appendix 1. Flowchart of the proposed game logic

ABBREVIATIONS

AV	Augmented Virtuality, inclusion of real objects into virtual environment.
AR	Augmented Reality, virtual objects overlaid on top of real world.
FPS	First-Person Shooter, a game genre.
GDD	Game Design Document, a comprehensive evolving document describing design of a game.
HUD	Heads-Up Display, a status bar visualizing information.
HMD	Head Mounted Display, a computer display worn on the head.
MR	Mixed Reality, blending of reality and virtuality.
MRTK	Mixed Reality Toolkit, a toolkit consisting of preset functionalities to accelerate development of mixed reality applications.
UI	User Interface, space for human-computer interaction.
UWP	Universal Windows Application, a set of definitions, protocols and tools for creating Windows applications.
UX	User Experience, a discipline of game design focused on psychology of the player.
VR	Virtual Reality, replacement of reality through simulation of artificial environment
WMR	Windows Mixed Reality, a platform for MR and VR experiences created by Microsoft

1 INTRODUCTION

VR and AR applications have become strong contenders for mainstream video games and game development. With the technology having taken tremendous leaps forward over the past few years, experiencing them firsthand through the eyes of an end-user or a developer has never been easier to approach. Indeed, the market is brimming with a variety of apparatuses capable of providing immersive experiences for a person by either constructing a completely artificial virtual world around them or simply overlaying meaningful pieces of digital content in front of their eyes. Furthermore, much more unexplored territory related to both reality and virtuality lies in between the two realms.

1.1 Objective of this thesis

The objective of this thesis is to introduce Microsoft's HoloLens device and to unravel its capabilities as the next-generation platform for gaming. In an attempt to achieve the goal, a sufficient level of understanding the theory behind the technology is required. Therefore, an examination into the subject will be carried out through deep analysis of e-books, articles, blogs, and developer documentations discussing the subject.

This thesis will cover some the ideas and technological advancements responsible for creating the foundations underneath HoloLens, discussing the subject of MR from variety of aspects and moving into the specifics of the HoloLens device, performing a brief inspection into the technical specifications in order to gain further insight as well as to examine the features it possesses.

In the later chapters, the thesis will shift the focus into applying the information gained to describe a conceptual design of a game known as Darts MR, a sports game candidate attempting at highlighting the physical interaction HoloLens greatly utilizes. The information related to MR and the possibilities it may propose are then transformed into features utilized in a game. Finally, the ideas and epiphanies indulged from the design phase are processed through a programming implementation of Darts MR in order to attempt proving the

adequacy of the subject. The chapter describes several specific tools recommended for developing a holographic application for HoloLens and then proceeds to provide a brief explanation in a step by step guide.

1.2 Initial approach

As the author had no preceding knowledge or insight of working with AR, VR, and MR platforms and devices in software development, the approach towards the thesis and implementation was cautious and slow. Although a substantial amount of work was spent on studying various blogs, articles, documentaries, and previous research work discussing the topics introduced in this thesis, a large portion of the total amount of work included personal testing and experimenting. Especially for the implementation, a multitude of different ways for implementing certain pieces of logic was discovered. Several sources proposed somewhat differentiating solution, however, what emerged the most interesting were the comprehensive case approaches providing a very detailed explanation from the very ground level. The cases were compared closely and resulted in practicable approaches for the author to pursue.

1.3 About the commissioner

GameLab is a studying environment provided for students learning game development at the South-Eastern Finland University of Applied Sciences. GameLab grants students with modern tools and equipment necessary for game development for many different platforms. The same tools are widely adapted in the industry, paving a smooth transition into working life and increasing the chances of employment in the game industry. The commission is based on the wish of the Gamelab to explore the capabilities offered by HoloLens in the context of game development.

2 MIXED REALITY

MR could be portrayed as a meeting ground for both real and virtual worlds, a bridge between the two environments, connecting them. MR forms a type of hybrid setting that includes elements from both virtual and physical realities. It is very often portrayed as sliding scale between completely real and virtual environments by many experts. (Techopedia 2019.)

Also, MR is sometimes used interchangeably with the term AR, as both realities generally involve placing virtual elements in a physical field of view. However, in this thesis they are considered different types of realities through their varying applications.

Whereas conventional VR applications meant to provide the participant with a complete reality separating immersion, further described as becoming part with the virtual game world and reasoning from the perspective of an artificial character (Fagerholt & Lorentzon 2009, 69), and AR applications utilized to overlay digital information and supplement data in front of the user's view whilst preserving the real-world objects around them visible, MR on the other hand brings together objects from both realities and blends them, merging the two realities into a single entity in which both the physical and the virtual objects could coexist, and interact with and among each other, as outlined in Figure 1.

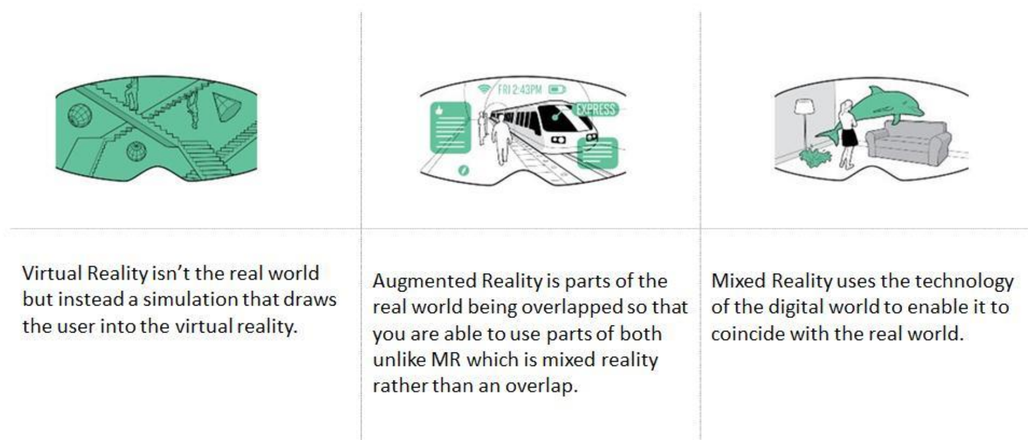

Figure 1. Differences between VR, AR and MR (Brown 2017)

It is quite possible to define MR simultaneously in more than one way. As a sovereign concept, mixed reality merges the elements from both the augmented environment and virtual environment. However, when applied to contain a larger range of reality technologies, it refers to the inclusion of all potential deviations of physical and digital content. (Reality Technologies 2019.)

Origins of the term MR can be traced all the way back to early nineties, when the concept was introduced on a research paper. The paper discussed about

the ideas of technologies involving the blending of the real and virtual environments somewhere along an axis known as “virtuality continuum”. The axis would go on to connect the elements of entirely physical to elements entirely digital. (Milgram & Kishino 1994, 2.) In the context of computer interfaces, this could be considered the first academic paper to use the term MR.

2.1 Virtuality Continuum

Developed by Milgram and Kishino in 1994, the “*Virtuality Continuum*” (VC), also referred to as “*Reality-Virtuality*” (RV) Continuum, is a continuous scale ranging between real and virtual environments. It is a monodimensional axis, with physical reality residing at one end of the continuum and virtual reality at the opposite. As represented in Figure 2, VC encompasses both AR and AV, with the applications for both ranging respectively somewhere on the axis. The closer an application situates towards virtual environment, the more immersive it becomes, presenting the participant with an increased amount of digital content. On the contrary, the closer the application moves towards real environment, the less virtual objects are presented and the more of reality is involved. Furthermore, and perhaps more so interesting, is the centrum of the axis. It is there, when the realities from both ends could be considered nearly indistinguishable from one another. The equivalent amount of reality and virtuality merged together. In this case, the participant could for example be presented in a situation with two identical objects, one being of physical realm and the other virtual. Visually, even aurally the objects could behave very much alike, with only the physical interaction, such as touching with a hand being the key factor in identifying which one is which.

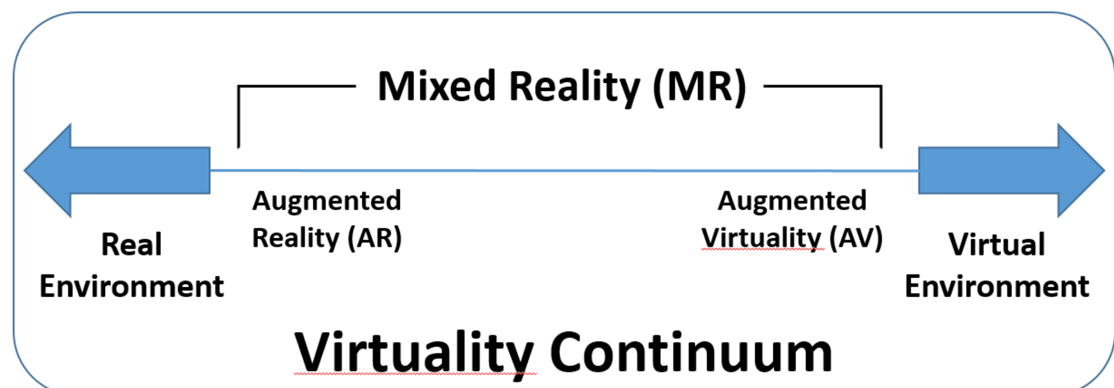

Figure 2. Visual interpretation of Milgram and Kishino’s Virtuality Continuum

Introducing the four most prominent elements responsible for composing the VC, first can be found the “Real Environment”. This is the unaltered physical reality consisting entirely of real objects. It is the conventionally perceived environment and includes all possible variations in which the real world is observed without any digital content inserted in between the observer and the world. Potential scenarios include but are not limited to cases such as viewing the world directly in person or through some sort of visual feed, for example a camera view of a smartphone or a tablet.

Moving forward along the axis lies the AR. In this state, the real environment is subjected to elements of virtual environment. The most commonly witnessed phenomenon can be considered a digitally rendered content placed on top of the real world, a type of overlay. This can be harnessed through different types of methods (such as rendering a digital HUD overlaid on top of an aircraft pilot’s visor, on the display of HMD, or smart glasses). Cases utilizing AR technology such as wearable Google Glass device and Pokémon Go! mobile game are both very well-known and have displayed great success in their respective fields (Lee 2018).

Tertiary element contributing to the continuum is the AV. The term is presumably the least known form of virtually enhanced reality when compared to its neighbors of AR and VR. This is very likely due to the straightforward evidence that it is a rather complex word to even say (VirtualiTeach 2017). However, its concept can be considered rather simple in that it is the reversal of AR. The location of the user interaction dictates the differences between the two terms. Should the interaction occur in the real world, it would be labeled as augmented reality. On the other hand, should the interaction take place in a virtual environment, it would be considered augmented virtuality. (Spacey 2016.) Therefore, if a highly immersive application is augmented with the properties of reality, it can be labeled as an AV application. For example, to design their own living room or kitchen, participant may utilize a touchscreen to relocate digital objects located in a virtual environment. As for an augmented virtuality example of gaming, the real objects and even the participant themselves may be projected into the virtual environment and interact whilst in there. (PC Mag 2019.)

Final component of the continuum is the VR. It is at the opposite end of real environment and can be considered the virtuality extrema. VR proposes a type of media consisting of interactive, computer generated simulations utilized to sense the location of the participant and replace or augment the feedback to various senses— granting the participant with the feeling of being immersed by the simulation (Craig 2013). In several cases, this involves the participant wearing a HMD that generates and projects a completely synthetic 3D world devoid of any involvement from the surrounding physical world in front of the participant's eyes. VR is notably the most commonly understood element inhabiting the VC, as there have been numerous peaks of interest in the related technology in the past few decades, mainly by early inventors and military. Ultimately, it could be considered that any element established somewhere along the axis ranging between the real and virtual environments could be described as a MR experience, for as long as the objects brought together from both the physical and digital realities are capable of interacting with and among each other.

2.2 Main display types

The means by which MR experiences could be introduced hardware-wise nowadays are various. Starting from the early days solutions of simple display systems in which the objects from both the real and virtual environments are presented together, to modern day HMD's that can display virtual objects in line of sight of the participant and place them among the physical objects inhabiting the real environment, as well as utilize hand gestures or voice recognition to detect input in order to interact with and manipulate the objects.

It is possible to categorize the display systems into two groups of optical see through and immersive displays. Devices utilizing optical see through method provide the participant with a direct view through optical elements such as holographic waveguides and similar systems, enabling overlaying of digital content on the real-world (Kore 2018). Furthermore, the display systems capable of producing true MR experiences are limited to optical see through glasses, as the participant's ability to observe the real environment among the virtual is pivotal for the blending of the two realities to succeed. Popular example devices featuring such properties include Microsoft's HoloLens and Magic Leap

One. The optical see through displays can be further divided into smaller subgroups of monocular and binocular displays. The measurement of the number of eyes required to observe something is known as ocularity (Kore 2018).

In case of monocular displays, the participant is provided with means of a singular channel for viewing and superimposing virtual information in front of one eye whilst keeping track of real environment with the other eye. Such display systems regularly come in very small-scale and consist of head-mounted unit and a single smart lens. Binocular display systems consisting of head-mounted unit and two lenses grant the participant with twice the channels for viewing by visualizing separate views for both eyes, thus producing a stereoscopic perspective. Although considered heavy, advanced and rather intensive performance-wise, these types of displays offer a significant amount of depth cues and the highest sense of immersion (Kore 2018).

Although some cases arise in which immersive displays can be categorized under MR displays, they generally situate at the distant end of virtual environment on the VC, and thus fall under the term VR instead. This is likely due to their nature of being are a closed ecosystem, as in the view of real world through the display is blocked. The participant is unable to see through the HMD and thus is unable to mix in the factor of real environment and the physical objects inhabiting it. However, Microsoft together with their partners have introduced a line-up of immersive headsets under new platform of WMR. Whereas the conventional immersive headsets generally have a requirement of being tethered to a computer and utilize base stations situated around a room to track participant's position and interactions, these WMR headsets on the other hand are completely wireless and do not require separate base stations for tracking purposes. Instead, they come equipped with front-mounted cameras as well as a set of built-in sensors intended to graph the physical position of the participant. In order to avoid the requirement of having external sensors, the devices feature design for six-degrees-of-freedom (6DoF) movement tracking known as inside-out tracking (Goldman 2018). The aforementioned features attempt at complimenting the properties of the real environment on the VC and may be used as reasons to move slightly towards that end on the axis, thus placing the WMR immersive headsets within range of MR as shown in Figure 3.

Figure 3. WMR immersive headsets within range of MR (Microsoft 2018d)

2.3 Free from the frame

The transition from a 2D frame into an immersive 3D world through utilization of various MR apparatuses presents an additional dimension for UX designers and creators to work with. As described by Microsoft (2018b), the long governing rule of "safely guarded within a frame" for displays no longer apply, as the participant is no longer situated in front of the digital world, but has rather moved forward, and situated within it. This transition and the additional dimension layer provided allows UI's to have depth, a third axis pivotal for creating a 3D position within the world and be located among the other objects and the participant interacting with them. As the transition from a classic 2D environment into a fully explorable 3D world space occurs, an important aspect of defining a new proper way of presenting meaningful information to the participant arises. The conventional practice for designing and implementing various UI's for a multitude of appliances has long been constrained into two axes, locked on a flat screen. This non-diegetic model, as seen in Figure 4a (in which player taking damage in a Call of Duty game is partially indicated by a red arcing 2D symbol in the middle of the screen), has proven a legitimate approach in many fields in which information presented to participant through different display systems of computers and smart devices is deemed necessary. However, as the interaction layer between human and computer shifts from 2D to 3D, it should be considered that the way the information is presented shifts, too. This introduces the diegetic and spatial displays.

Diegetic display could be described as a type of display that is linked narratively to a fictional world or a game. The display has a set location within the world, and should the world be a product of MR experience, the position of the display could be provided by either physical or virtual means. Furthermore, diegetic displays are visible to the fictional characters inhabiting the world, as

shown in Figure 4b (an example of the game Dead Space, in which the player health is indicated by a blue meter attached to the fictional character's back, and which in turn is a part of the fictional character's space suit and is visible to them). Diegetic UI elements, by extension, are visible or audible pieces of the game world that both the game characters and the player can see or hear and receive information from (Peacocke et al. 2018). In an additional gaming example, to make the information of an ammunition counter of a weapon visible to both the player and the character controlled by the player, the counter could be illustrated as a visual part of the weapon. The ammunition counter would thus be considered part of the game space as well as the game fiction, resulting in a diegetic display. (Peacocke et al. 2018.)

Spatial type of display could be presented as one that situates in between diegetic and non-diegetic types of displays. It exists within a 3D space of a world, but unlike diegetic display, does not possess a narrative connection to the world's fiction, as depicted in Figure 4c (an example of Watch Dogs, a game in which the player navigation is aided by a set of blue markers geometrically positioned within the game world and visible to the player, but not the character). The display can be observed by the participant interacting within the world, but not by the possible cast of fictional characters inhabiting the world. Therefore, it could be depicted as a stripped-down type of diegetic display. Furthermore, and similarly to a diegetic display, the spatial display has a location within the geometry of the world that can be provided by physical or virtual means, should the experience be set in MR. In the context of MR, an example case could be tooltips aiding the participant playing a MR game. A virtual object such as a digitally rendered piece of paper with the game rules written on it and attached onto a physical wall within real world could represent a display to aid the participant in the game by guiding them through the basic rules. The object would be geometrically present in the world and visible to the participant, but not to the possible fictional characters. A way-marker visualizing a route to an objective could be depicted as an additional example of utilizing a spatial UI (Harney 2017).

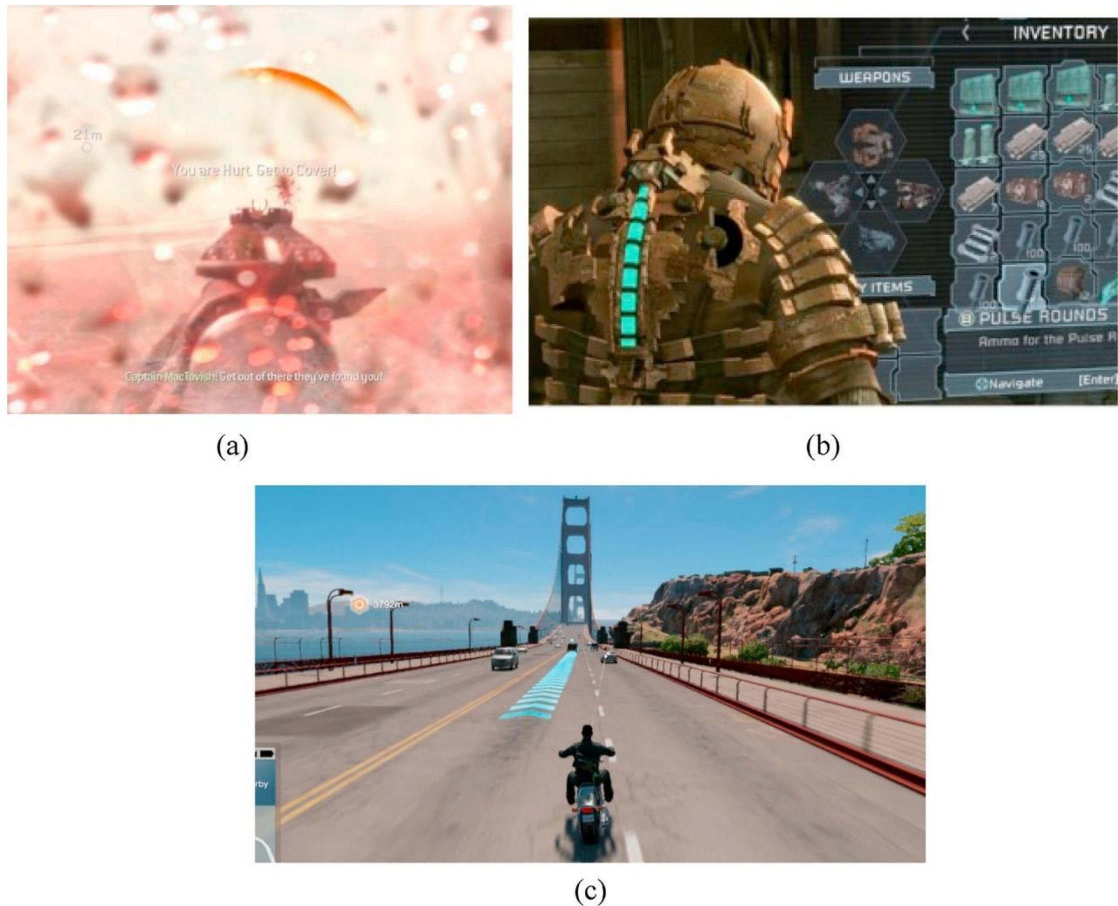

Figure 4. Different types of in-game displays. (a) non-diegetic, (b) diegetic, and (c) spatial (Peacocke et al. 2018)

Discovering proper balance between immersion and usability is considered pivotal when deciding the method of approach between non-diegetic, diegetic or spatial displays (Harney 2017). This should be considered exceedingly critical for MR implementations, in which participant immersion, as well as comfort, is of utmost importance. Not only may implementation of wrong type of display utilized in MR hinder participant’s overall performance and result in a loss of immersion, it may as well cause discomfort and fatigue. One such case could potentially be the vergence-accommodation conflict, in which the participant accommodates their eyes to the focal distance of the display to receive a sharp picture, but converge to the distance of the possible element of interest to receive a single picture. (Microsoft 2019a). Accommodating and converging to different distances will break the natural link between the two cues and may result in participant expressing visual discomfort or fatigue. However, it is possible to facilitate, if not completely counter the symptom with a diligent placement of the desired display type. Adjusting the distance between a display and

a participant correctly, as well as designing the display to appear natural inside the world and the experience the participant may be immersed in can indeed be the solution. For holographic HMD’s (such as HoloLens), Microsoft has provided guidance on what could be considered a healthy distance between the participant and the display, as portrayed in Figure 5. It is very possible to reduce or even completely prevent the fatigue and discomfort caused by vergence-accommodation conflict by adjusting the distance between objects and the participant as near to 2.0m as possible (Microsoft 2019a).

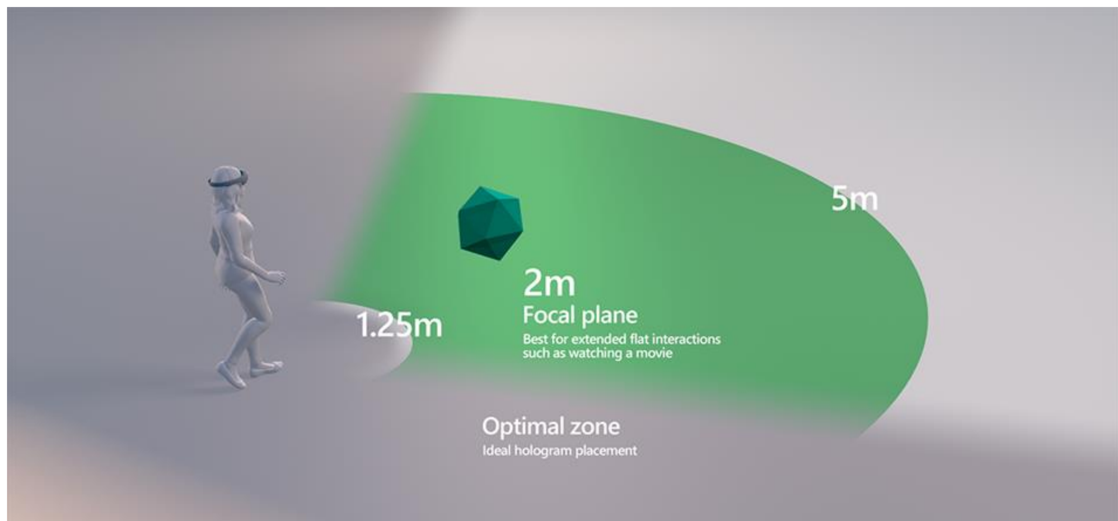

Figure 5. Ideal display range (Microsoft 2019a)

3 HOLOLENS

Designed and developed by Microsoft, HoloLens is a completely self-contained, holographic HMD and computing unit running on a special version of Windows 10 operating system, known as Windows Holographic. HoloLens can be depicted as a mixed reality device that attempts to merge the physical and virtual environments (Tuliper 2016). The development of HoloLens was overseen by Alex Kipman, the developer also related to Microsoft’s Kinect motion controller (a webcam-style peripheral device created for the Xbox console with a purpose of sensing user’s motion and using it as input for controlling and interacting with the console). Although manufacturing of the original Kinect has officially ceased, much of the technology left behind still remains, and has been further developed on. The sensors previously utilized in the Kinect are now employed to power Microsoft’s HoloLens devices. Furthermore, the methods from the Kinect voice-activated interface are now taken advantage of in some of the newer features such as Cortana and Windows Hello, as well as

in the new interface known as GGV that consists of Gaze, Gesture and Voice. (Kaelin 2017.) HoloLens was first released to the public in 2016 under the label of Developer Edition. Although the availability of the device could reach the end user, it was primarily targeted towards developers due to its rather heavy price tag of $3000, and the shortage of content available for the end user to consume. However, the advantages of HoloLens and its MR capabilities could present for industrial purposes became quite apparent, and it was widely adopted on several industrial sectors, particularly in manufacturing, healthcare, architecture, and construction fields. Perhaps by that very reason, later on, a new edition providing additional enterprise features was introduced under the label of Commercial Suite, which was targeted exclusively towards the developers and enterprise purposes and included substantially higher price tag of $5000. More recently, Microsoft has introduced a follow up device to the original HoloLens with the revised HoloLens 2 that further improves upon core performance and includes new features for enhanced experience.

3.1 The device

Physically, HoloLens is a hard plastic headband that comprises of two separate rings: an outer ring that can be considered the device's main frame with a built-in computing unit and a pair of smart lenses attached to it and shielded by a visor, and an inner ring with the purpose of wrapping the HMD around the user's head, as shown in Figure 6.

Figure 6. Physical appearance of HoloLens (Microsoft 2018d)

The inner ring can be extended in length as well as rotated vertically for better alignment with the user's head. Also, the inner sides of the inner ring are quite well cushioned to allow for soft and comfortable fit on the brim of the user's head. At the back of the inner ring also lies a small plastic dial responsible for the adjustment of the headband's tightness around the user's head. Although the HMD isn't necessarily designed to sit on the user's nose, it features a small concave of a socket positioned between the smart lenses, allowing the nose of the user to fit in. Furthermore, an optional rubber nose pad can be attached to the socket to increase comfortability as well as to increase versatility of different shapes of noses to fit in the socket. In terms of weight, the outer ring can be considered the heavier part of the apparatus, containing small speakers sitting on both sides above the user's ears to produce spatial audio (three-dimensionally localized sources of sounds or music positioned at specific point in an environment around the listener) as well as the holographic display system. Front-side interior of the outer ring includes the computing hardware. The hardware consists of various components such as Intel processor, a custom holographic processing unit (HPU), 2 gigabytes of memory, and up to 3 hours of battery life in normal active use (Evans et al. 2017) Additionally, HoloLens includes a variety of smart sensors responsible for sensing and tracking user input, including gaze, gestures, speech, environment, and movement.

Perhaps the most sophisticated piece of hardware equipped in HoloLens is the optics system. Capable of transmitting and visualizing holographic content into the eyes of the participant, the HoloLens HMD utilizes two holographic see-through-lenses, also known as waveguides (Types of thin plastic or glass lenses that employ internal reflection to guide projected particles of light into the eyes of the participant by bouncing them between the two surfaces) to project different shapes of holograms in front of the eyes of the participant, as can be seen in Figure 7. The holograms themselves, further beamed into the waveguides, are constructed by two HD 16:9 light engines, which consume the majority of the combined processing power of the device in order to create the holograms from light points (points of light emitted through the waveguide to appear at a fixed distance in front of the eyes of the participant). The amount of the light points emitted by the engines, also known as holographic

resolution, can reach up to a maximum of 2.3 million light points in graphical fidelity.

Figure 7. Screenshot taken by using HoloLens's Mixed Reality Capture, depicting a hologram projected and positioned into a 3D world.

3.2 Input

As the reality of the participant adjusts more towards a MR experience through the inclusion of holograms, the aspect of interaction becomes essential. Towards that end, HoloLens comes integrated with multiple capabilities for interacting with the holograms as well as understanding the surrounding physical environment. By exploring the aforementioned capabilities, several key forms of input emerge.

3.2.1 Gaze

Perhaps the most noteworthy form of input is the way HoloLens understands the position of the participant in the world as well as the direction they might be looking at, also known as gaze. In mixed reality appliances, the fundamental means for of both targeting and user input is gaze (Microsoft 2019b). By taking advantage of the position and orientation data of participant's head, HoloLens is capable of determining the head gaze vector. This can be described as a sort of ray pointing straight forward from directly between the participant's

eyes. Furthermore, the ray can be utilized in holographic applications developed for HoloLens to pinpoint holograms as well as to determine whether the participant is looking at a virtual or real-world object. To stimulate participant's intention in an application created for HoloLens, a very common practice is to indicate the gaze with continuous visual feedback such as a cursor (A visual indicator of the participant's current targeting vector, providing continuous feedback to aid the participant in understanding where their focus is at all times, as well as to indicate what possible hologram, area, or other point of interest might respond to input), as shown in Figure 8. It is possible to further utilize the cursor to target an object inside an application and attempt at interacting with it. Gestures and voice, as well as motion controllers are considered the essential means for the participant to perform interactions in mixed reality (Microsoft 2019b).

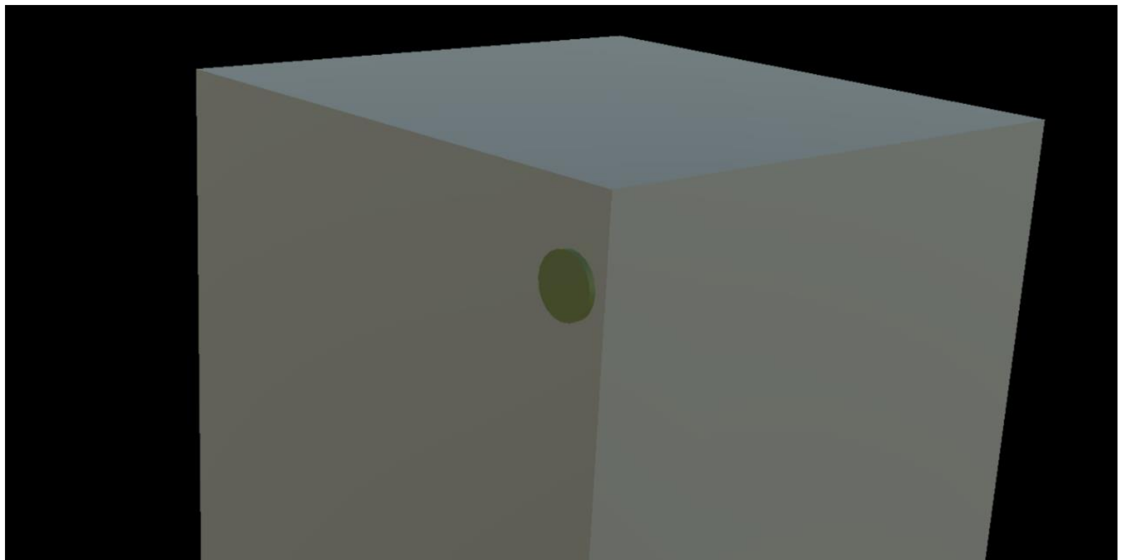

Figure 8. Depiction of a primitive cursor, a green tiny circle aligned on top of a cubic object in 3D space, visually indicating gaze.

3.2.2 Gestures

Whereas gaze is mostly responsible for determining the point of interest in a holographic application, gestures complete the formulae for interaction by introducing the element of acting. Interaction is formed by utilizing participant's gaze to target and hand gesture or voice to act upon a targeted hologram (Microsoft 2019c). The types of gesture sources recognized by WMR headsets include hand, motion controller, and voice. However, HoloLens recognizes solely the hand and voice variants as it lacks the necessary support for motion

controllers. Out of the two gesture types recognized by HoloLens, the more conventional method for interaction could be considered the hand gestures, which can be further subdivided into two core component gestures called air tap and bloom. Air tap is a type of gesture in which participant taps their index finger with their hand held upright, as seen in Figure 9. It is a universal action designed to perform a select type of action, similarly to conventional mouse clicks used in interacting with traditional computer applications.

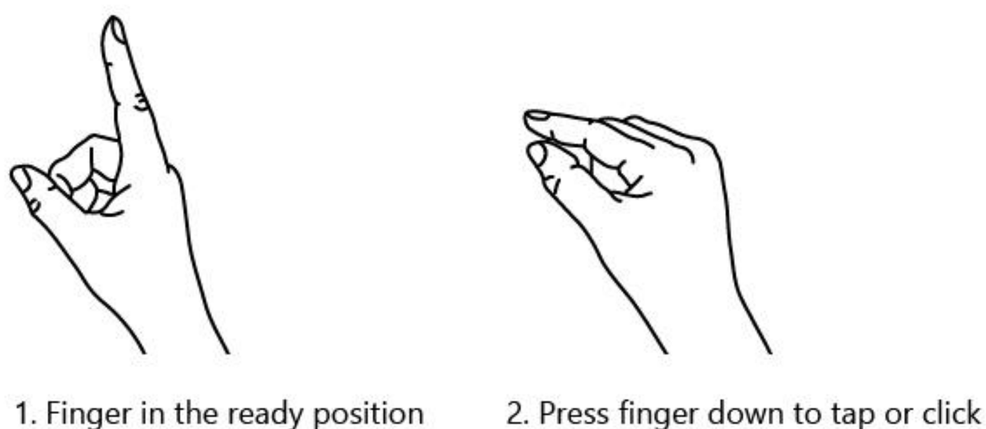

Figure 9. Performing air tap hand gesture (Microsoft 2019c)

Bloom is a special system action designed exclusively for HoloLens. The sole purpose for it is to bring up the holographic start menu (A central system menu containing a list or category of settings and applications installed on the system). The functionality of the bloom gesture can be considered identical to pressing the Windows key on a keyboard plugged into a system running on a Windows operating system. In order to execute a bloom gesture, the participant is instructed to hold out their hand with the palm facing upwards whilst keeping their fingertips enclosed, and then open the hand in a releasing fashion. The procedure is depicted in Figure 10.

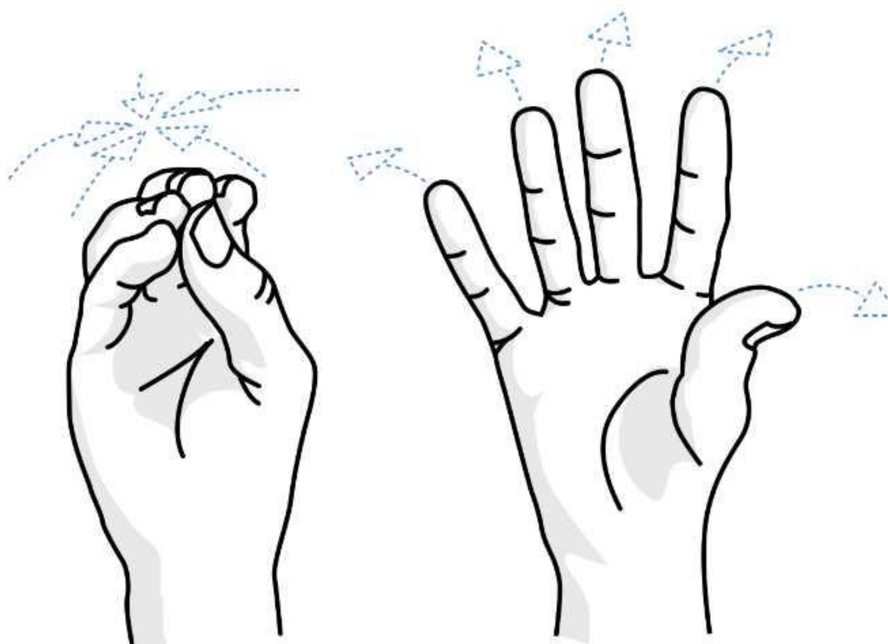

Figure 10. Performing bloom hand gesture (Klint 2018)

In addition to the two core hand gesture components, HoloLens provides support for several more advanced composite hand gestures. The first gesture is hold, which is quite similar to an air tap in that the gesture starts from the index finger of the participant being in ready position, and then proceeds to perform a continuous tap by holding the finger pressed down. Whilst the participant is holding down the index finger, the gesture can be utilized to perform a range actions such as initiating relocation of a hologram or pausing a hologram. This combination gesture can be considered equivalent to holding down a mouse button. The secondary gesture is manipulation, which can be considered an extended version of the hold gesture. Similar to the hold gesture, manipulation begins by participant holding down their index finger. The gesture can then be continued by participant moving their hand freely whilst keeping the finger held down, allowing the targeted hologram to react 1:1 to the participant’s hand movements. The manipulation gesture can be used for performing actions including relocation, resizing, or rotation of a hologram. The final, and perhaps the most complex composite hand gesture is the navigation gesture. The functionality of the navigation could be described equal to operating a virtual joystick. Navigation begins similarly to a hold gesture by the participant tapping and holding down the index finger, and then proceeding to move their hand along three separate axis (such as X,Y, and Z), ranging between -1 to 1,

with 0 being the starting point, therefore creating a normalized 3D cube centered around the initial press. It is possible to utilize navigation to create constant velocity-based scrolling or zooming gestures, equivalent to an example of using a mouse to scroll a 2D UI by pressing down the middle mouse and then shifting the mouse back and forth (Microsoft 2019c).

Performing hand gestures takes place within a gesture frame. The frame is an area generated in front of HoloLens by the gesture-sensing cameras and defines the boundaries in which the device can detect the gestures performed by the participant. The frame is set roughly from nose to waist depending on the height of the participant, as illustrated in Figure 11.

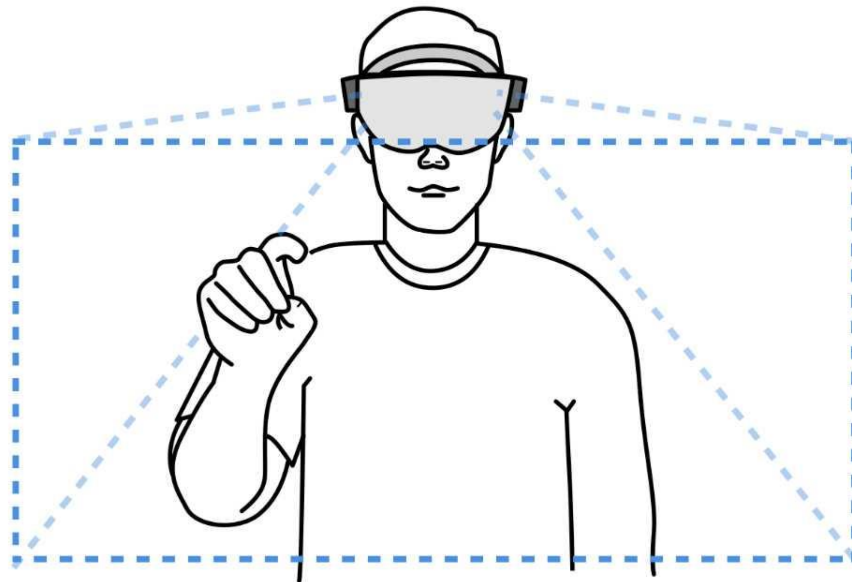

Figure 11. Gesture frame (Klint 2018)

Should a gesture such as manipulation or navigation take place completely or partly outside the gesture frame, as soon as the gesture cannot be detected, the HoloLens will lose the input (Klint 2018). Recognition of the hand gestures performed within the gesture frame can occur when either or both hands of the participant are visible to HoloLens, as in the hands of the participant being in line of sight of the gesture-sensing cameras. Furthermore, the recognition is affected by orientation of the participant’s hands, as HoloLens can see them when they are either in ready state (hand reached out with back facing towards the participant and index finger up) or the pressed state (hand reached

out with the back facing the participant and the index finger down). Should the hands of the participant be in any other pose, HoloLens will lose track of them.

3.2.3 Voice

In addition to the two key forms of input of gaze and gestures, HoloLens supports voice as source for receiving input from the participant. Voice enables the participant to directly command a hologram without the need for utilizing separate hand gestures, allowing in many cases for a much quicker and simpler way to interact with the targeted hologram. Voice can be considered excellent at managing complicated interfaces, as the participant may use a single voice command to quickly cut through the nested menus (Microsoft 2019e). HoloLens comes equipped with multiple built-in commands universally recognized by the platform. Perhaps the most basic of the commands is the select command. Select behaves equivalently to an air tap gesture, allowing the user to activate holograms or other holographic elements (such as UI elements) with a single line of speech. HoloLens then confirms the voice commands by letting the participant know that the command has been executed by displaying a tooltip with “Select”, as well as cueing a sound effect. Additional HoloLens-specific voice commands include but are not limited to examples such as “What can I say?”, “Go home”, “Launch”, “Move”, and “Take a picture”. As can be observed from the aforementioned examples, the voice commands are rather self-explanatory in describing their functionalities. Finally, HoloLens enhances the experience of using voice commands by displaying labels on various UI elements, telling the participant what voice commands they can use to interact with the elements. By using gaze to target and highlight a button in a holographic application (such as the “Exit” button frequently located in the top right corner of an application), the participant will be notified if the button possesses a voice command attached to it, by displaying a label above it, as shown in Figure 12. This model of voice input is known as “see it, say it”.

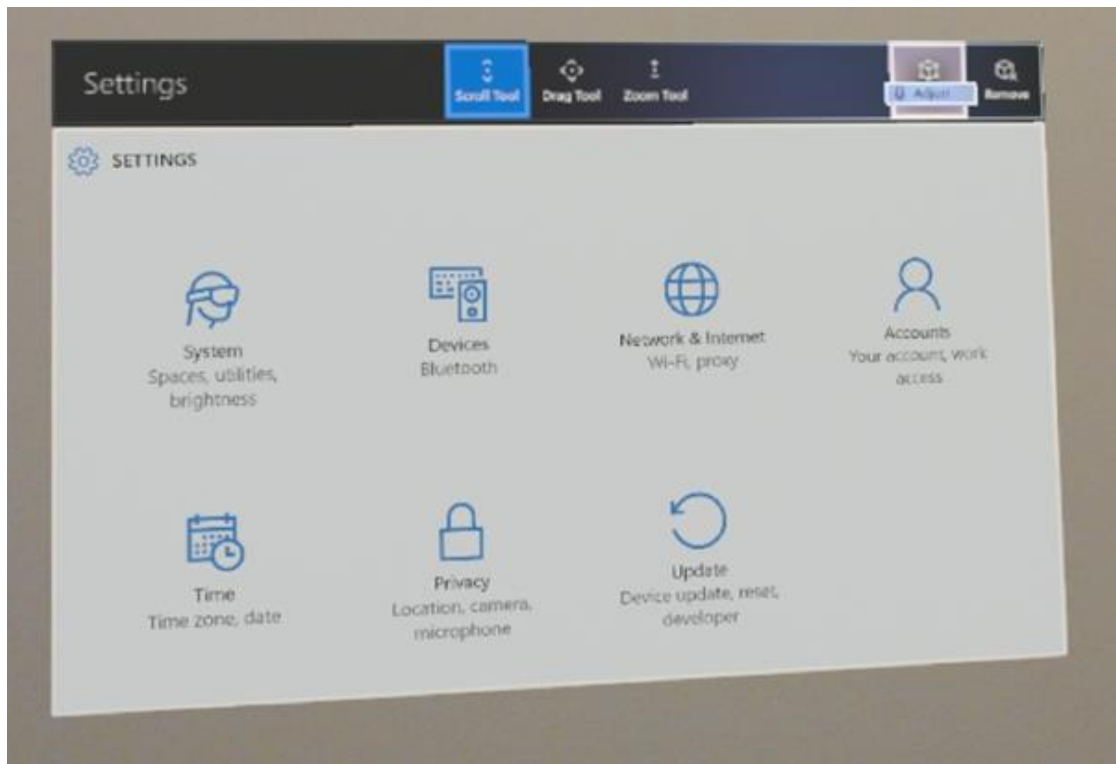

Figure 12. "See it, say it" label in the top right corner of a holographic application (Microsoft 2019e)

This sort of approach is highly recommended to follow when developing applications for HoloLens, as the participant can easily understand what to say to more efficiently control the system, and therefore receive a better experience.

3.3 Spatial mapping

Similar to understanding physical gestures performed by the participant, HoloLens is capable of understanding its surrounding physical environment. This feature is known as spatial mapping. Four environmental cameras on the front of the HoloLens are utilized to map the physical surroundings and objects to generate a 3D model of the real-world. This feature allows HoloLens to stand out from rest of the AR devices as a MR device. (Klint 2018.) Spatial mapping feature consists of two primary types of components, a spatial surface observer and a spatial surface. When utilized in a holographic application, the spatial surface observer can be described as the fundamental contributor of spatial mapping, tasked with the procedure of scanning of the physical surfaces. For each scanned real-world surface, the application utilizes what’s known as spatial surfaces, a tiny volume of space represented as triangular

meshes combined together to generate a complete digital mesh of the surroundings as seen in Figure 13.

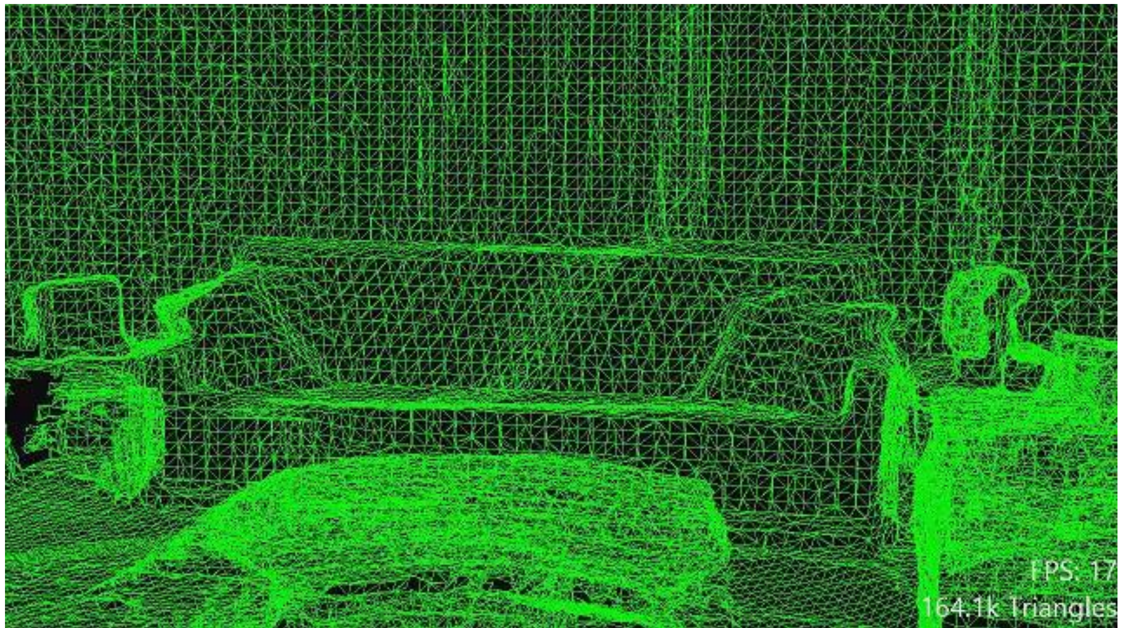

Figure 13. Model of a room, generated by spatial mapping (Microsoft 2018c)

Whilst using HoloLens, the surrounding physical environment of the participant is continually scanned, and from the scan results HoloLens generates a 3D model matching that of the space the participant is currently inhabiting. The model is actively updated as spatial surface observer studies the environment for changes, and if necessary, augments the model by adding new spatial surfaces from the physical real-world counterparts captured by the environmental cameras and removing spatial surfaces that no longer exist within the view bounds of the spatial surface observer. Additionally, the developer edition of HoloLens comes with a tool known as “Device Portal” (a web browser based application containing various settings and features to remotely monitor and interact with the HoloLens), which allows the participant to view the generation of the model in real time, as can be seen in Figure 14.

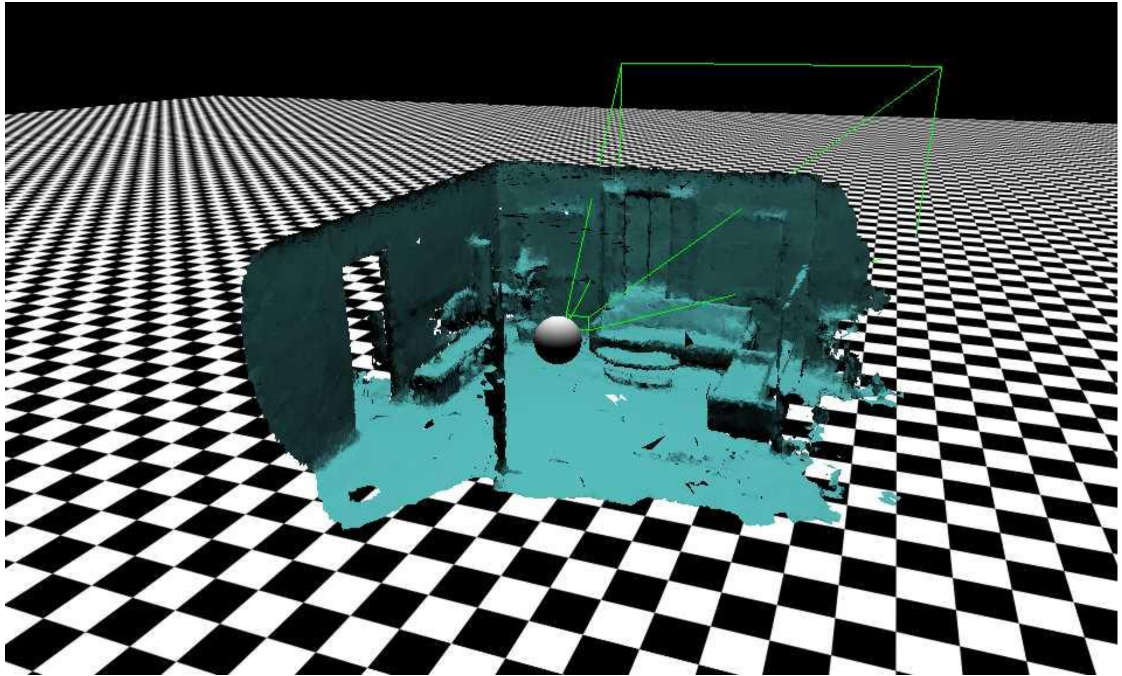

Figure 14. Real time feed of spatial mapping in progress (Klint 2018)

Furthermore, for each holographic application, the spatial surface observer is required to define one or more bounding volumes in which the spatial mapping may occur. In programmatic sense, the shape of bounding volume may also vary, and the shape can be modified to match that of a box or a sphere, for example. In terms of positioning, the volumes can be configured to be fixed (a static position with respect to the real-world) or they can be hooked up to the HoloLens, so that they move in tandem with the HoloLens as the participant moves about in the physical environment.

For the holograms to behave naturally with the physical environment and create a convincing illusion, such as having a digital ball roll smoothly on a physical floor, it is necessary for a holographic application to be aware of both virtual and physical realities. A very accurate positioning of the holograms is of paramount importance, so that the immersion of the participant remains coherent. For this purpose, HoloLens utilizes spatial coordinate system to calculate the positional interactions between holograms and the physical surfaces.

In order to understand the complexity of spatial coordinate systems for HoloLens, one is urged to consider a similar system utilized in a completely virtual environment. In a virtual application, a single master-coordinate system

can calculate the positioning of every object part of the experience very precisely, as the elements of physical world are out of scope. Every object part of the application is able to map and relate to same coordinate system. This can lead to a very stable and precise experience for the participant. However, in case of HoloLens and holographic applications, in which physical objects are in scope of the experience due to spatial mapping, the coordinate system must be able to calculate the positioning of holograms located in a physical environment. This can prove a synchronization challenge, as the spatial mapping utilized to digitalize the physical environment might not scan the surroundings accurately, leading to either incomplete or inaccurate virtual model of the space, which may further affect the position between holograms, as well as cause other malfunction (such as making the holograms float in air or suddenly shift positions). Consequently, the participant may become motion-sick at worst. However, HoloLens comes with a feature capable of countering the issue, known as spatial anchors. To illustrate a pivotal position in the world of which the system should be aware of over time, a spatial anchor may be utilized. The spatial anchors receive individual coordinates within the system that are managed if necessary to ensure the anchored elements remain still in their place. (Klint 2018.) Spatial anchors may prevent drifting of holograms as well as ensure that they remain at their designated positions, even as the spatial mapping updates the model of the space.

Other meaningful features and uses of spatial mapping include occlusion and visualization. Perhaps the more important is the occlusion, which is utilized to occlude holograms. In order to improve the experience of reality in a holographic application, spatial mapping can occlude holograms by either hiding them completely or partially from the view of the participant. For example, a hologram positioned behind a physical wall should be considered invisible to the participant in order to increase perceived reality. This, of course, presumes that the wall has been scanned and included in the spatial map model, as the hologram would be visible through the wall otherwise. However, sometimes, it can be considered undesirable to occlude holograms. Should the participant wish to interact with a hologram, it would have to be visible in some way. One method achieving this could be by rendering the hologram divergently when it is occluded by the spatial mapping (such as altering the level of

brightness). In that case, the hologram would be visible enough for the participant to visually locate it, whilst still being obscured enough to provide the impression of being hidden behind something. Being a MR experience in which physical real-world objects provide fundamental base for holograms to interact with, most of the times it can be considered appropriate for spatial surfaces generated by spatial mapping to be invisible, as in to reduce visual clutter and allow the real-world speak for itself. However, sometimes it can prove meaningful to visualize spatial surfaces regardless of the real-world counterparts being already visible. (Microsoft 2018c.) This could prove especially useful in cases in which the hologram and the participant seek to share understanding of something (such as determining whether a physical surface is solid or not). An example case could be a hologram of a painting. When attempting to place on a wall, it should provide visual feedback whether it is able to settle on the wall or not. It is possible to determine this by rendering the individual spatial surfaces behind the hologram differently, thus producing a “ground” effect by emitting a shadow onto the surface (Microsoft 2018c). This would allow the participant to have a much sharper feel of the precise physical distance between the hologram and the surface. A more general example of a similar practice could be depicted by participant visually previewing a change before committing to it (such as previewing a picture before printing it).

4 DARTS MR

This chapter discusses the design aspect of a video game known as Darts MR. It is a side project of the author with the purpose to serve as a test subject on which to apply the theory examined in previous chapters in pursuit of creating a MR game for the HoloLens device. The chapter introduces several principal design concepts of the game, including the game narrative and cast of characters, core gameplay blocks and logic, as well as control scheme and UI design decisions. The main purpose is to explain fundamental ideas of the game and create a layout for an actual implementation stage, introduced in later chapters. Designwise, MR applications are heavily influenced by the surrounding real-world environment through visuals and user interaction. Therefore, the design of Darts MR is shaped to follow closely its real-world counterpart sports game of Darts. The design of the game should aspire to present the player with an immersive, relaxed, and casual gaming experience, with the

game flow being straightforward to follow and to understand. To further assist the creation and development of Darts MR, a separate GDD will be created to provide a more highly detailed explanation of the vision for the game and describe the components planned for implementation.

4.1 Game analysis

HoloLens blends content from both real and digital worlds in order to generate Mixed Reality experiences. Therefore, the initial approach should revolve around a game type that could also be physically possible in the real world. A game based on interactions between physical and virtual objects and the player taking advantage of surrounding physical environment to interact with the game. Therefore, the potential of HoloLens could be efficiently harnessed as the core features of HoloLens would be utilized. As Darts MR strives to provide the player with easily accessible casual sports experience, the target audience of the game naturally emerges as rather broad. Sports games are most user welcoming in regards of complexity, which means that players of all age should be considered in the pool, limited only by the means of not owning, as well as capability of wearing and operating the HoloLens device. Therefore, the design of the game should not truly restrict any type of user from playing the game.

4.2 Story and characters

Depending on their genre, games can frequently come bundled with rich and deeply written stories in which the characters are given life by their personalities and duties in the game world. However, for Darts MR this should not be the case. As a casual single player sports game, it does not inherit any concrete written story or characters in it. Instead, the focus of the design should be maintained on the action and fluidity of the gameplay mechanics. Characterwise, Darts MR does not present any scripted, game world bound characters. Through HoloLens’s capability to render surrounding real world into a 3D model by utilizing spatial mapping feature, it is possible to include other personnel standing in the vicinity of HoloLens’s environmental cameras into the game world as static virtual objects. These objects, however, should not be categorized as genuine in game characters, as they are merely portion of the level output drawn in by HoloLens.

4.3 Gameplay overview

Relaxed sports and action are the governing factors for design of Darts MR. Therefore, the game utilizes fundamental elements and mechanics from both sports and AR game genres in pursuit of producing a hybrid form of a genre in which physical actions related to real world sports games performed by the player affect the movement and behavior of digital content, and ultimately control the flow of the game. Furthermore, the flow of the game is designed to be divided into several distinct states. Each state of the game is responsible for executing a particular segment of the gameplay elements included in the game. The details of the game states are further detailed in Appendix 1. The game being designed exclusively for MR, it requires various information from the real world as inputs for the game engine to process and generate the in-game world and ascertain that the game objects and functionalities adhere to rules of the game and operate correctly.

The first game element the player is designed to be introduced with after starting the game is the main menu. It includes various paths to different game features the player may be interested in exploring. These features are the game settings in which the player can modify independent game related details (such as the graphics quality and audio, as well as other more trivial game related options), leaderboards designed to showcase the record of scores of all players, as well as the option to start a new match. Darts MR is designed to present two distinctive game modes for the player to select from and play after starting a new match. A game mode offering more authentic experience which also takes a greater advantage of HoloLens's capabilities is known as "Free Mode", and a more simplified game mode known as "Stationary". In "Free Mode", the player is given freedom to move around and explore the environment while HoloLens utilizes spatial mapping to scan the physical surroundings such as walls, floor, ceiling, chairs and tables, and render a static mesh based on scanned information, further used to construct the level for the game. Once finished, the player will be able to tap and place the holographic board on any fit surface scanned by HoloLens and included in the final level mesh and begin the match. The secondary, and more simplified game mode is known as "Stationary". This game mode is designed best suited for

players who would rather prefer to stay still in specific location (such as sit on a chair) than move about in the surrounding environment. Also, instead of constructing a level from the physical surroundings of the player, HoloLens will simply just render the board and place it in front of the player at default distance, which the player will be able to manually control and set before starting the match. In either game modes, the player is given eight darts which to throw at the board. Each dart landed on the board will grant the player with set amount of points, and depending where the darts land on the board, the amount of points will either increase or decrease. Similar to the real-world counterpart, the closer the darts land towards the center of the board, the more points player is awarded with. Once all darts have been thrown, the match will end and the player will see the score board, an UI element with the purpose of displaying the final score achieved by the player as well as a text field in which to optionally enter the name of the player.

4.4 Control scheme

In order to provide intuitive MR gaming experience, the controls of Darts MR are centrally designed around the gaze, gesture and voice systems featured by HoloLens. The aforementioned systems are further utilized to handle input from the player in order to execute particular gameplay functionalities. For example, the gaze input is applied to move a cursor according to the head position of the player and the direction they may be looking at. This is designed so that the player can easily pinpoint over in-game content they seek to interact with, as described in Table 1. The interaction between the player and each game object can be achieved by using either hand gestures or voice commands, and in some cases, both.

Table 1. Proposed control scheme

Gesture / Voice Input	Action it Performs
Air Tap / Select	Selects a targeted game object or UI element.
Manipulation Started	Holds onto the dart in preparation of throwing it.
Manipulation Updated	Updates the position of the dart 1:1 to the player’s hand movement

Manipulation Completed / Throw	Releases the dart and sends it flying.
Manipulation Canceled	In case of HoloLens losing track of player’s hand, returns the dart to last known hand position.
Gaze	Moves the cursor by following player’s targeting vector to pinpoint and help player better understand where their interests are in the game world, and to indicate what area, hologram, or point will respond to input.

4.5 Game aesthetics and user interface

As the game utilizes spatial mapping to design, build and render the game world based on the physical counterpart, the core theme and art style of the game naturally emerges realistic. In this case, every game object as result of the scanning procedure (such as the game level) may inherit familiar and realistic shapes and features to them. This in turn can introduce the possibility for the game to present the player with a sense of “feel-at-home” impression when they enter the game, as everything they see could be of something they already know and are familiar with in the real world. Also, it could become easier for a player to get into the game, as they would not get lost as easily. In terms of player interaction (such as gestures), the experience should feel and look similar to the real-world counterpart. For example, picking up a dart and holding it in between their fingers should imitate the feel and look of that of the real-world dart. Similarly, throwing the dart at the board should feel and look as if they were throwing a dart in the real world, the dart soaring through the air and (hopefully) landing on the board. The dart and every other game object should inherit the very behavior of their real-world counterparts for a realistic and immersive game experience.

According to the several guidelines provided in earlier chapter discussing the UI elements within MR, the UI should generally blend smoothly into the game world. The player should be able to interact with UI elements as if they were real, physically in front of them. However, for the sake of simplicity, the UI ele-

ments such as menus and buttons for the first iteration of Darts MR are designed to be represented as non-dietetic elements, while attempting to adhere to the rule of not positioning them in a single fixed location, but rather allowing them to follow the gaze of the user at comfortable distance. Furthermore, all the panels, buttons and other interactable UI elements should be located in appropriate positions in the game view. They should never be placed obtrusively, as in blocking the player’s view of the actual gameplay, or placed in a difficult, out of reach position. The main menu window of the game should feature tappable buttons to begin a match, modify game settings or quit the game. The main menu UI element itself is designed to be aligned in the middle of the game view for quick and easy access.

During a match, the player score, multiplier and the amount of darts left should be displayed in the top left corner of the game view. The dart currently selected by the player should be shown in a small panel positioned in the top center border of the game view. By performing an air tap gesture on the dart panel, the player can bring up the dart selection panel that should be positioned in the middle center of the game view. The dart selection panel is a row of sub-panels that lists the darts available to the player, as well as providing buttons the player may utilize to scroll through the row in order to select a desired dart type. The selected dart type should be highlighted by a thick border around the sub-panel. Player may confirm the selection and close the window by performing an air tap gesture on the selected dart. The post-game UI includes score panel displaying the amount of points player may have received during a match. The panel is designed to include descending rows starting from the highest score as well as input field for the player to enter their name. Additionally, buttons for playing again or quitting back to main menu are considered. As the score screen panel displayed post-match and leaderboards panel accessible through the main menu share rather similar functionality, the UI design should be considered retained the same between the two features.

5 IMPLEMENTATION

This chapter discusses the implementation phase of the game Darts MR introduced in the previous chapter, depicting the development process of moving

from design into practice and unraveling the vital stages of realizing a complete and functional game. Although the implementation strives to greatly adhere to the initial design concepts previously discussed, the MR platform is very new to the author and everything that follows should be considered experimental. However, by examining documentaries, tutorials and generally perceived best practices from various enthusiasts, the implementation of the game is aimed to be executed in such manner that the final product could be considered a functional and genuine MR experience.

The chapter begins by offering a brief explanation of tools required to create the necessary game assets, write the code, and finally build and deploy the project. This is followed by steps of setting up the project, explaining the creation of the actual game project as well as defining some essential properties that enable creation of MR games within the game engine. Creation of the fundamental game objects is the primary subject of implementation, containing the process of creating the game objects and the code of the game. Finally, the implementation is concluded by going through the building process of the game application and deploying it to HoloLens.

5.1 Prerequisite tools

The implementation of Darts MR requires that several important tools are installed on the development system. Also, because of the tools utilized being rather intensive on system resources, it is highly recommended to perform development on a moderately performant computer in order to ensure a stable and efficient development environment. The selection of tools for developing on MR platform is heavily based on information available in Microsoft Mixed Reality documentation. The suggested software installation list elaborates upon each tool about their purpose for the development and the reason as to why they should be considered, as well as providing instructions on how to download and install them. According to the documentation, for optimal HoloLens development, the installed software set includes Visual Studio 2017, Windows 10 SDK, Unity, MRTK, and HoloLens emulator. From the aforementioned software set, however, only Visual Studio 2017, Unity, and HoloLens emulator will be included, as Windows 10 SDK already comes bundled in Visual Studio, and MRTK is considered redundant. Additionally, an optional 3D

creation suite known as Blender will be installed to assist at creating some of the essential game assets for Darts MR. Game assets depict elements that can determine whether an application is perceived as classic software or a game. A game asset can be anything that is utilized by a game, including graphics such as art and 3D models, music and sound, as well as code responsible for the game logic. A game engine can consume entire projects consisting of game assets. (Bouanani 2015.) To give further insight and help understanding the implementation chapter better, each tool is given a brief explanation in the following subchapters.

5.1.1 Visual Studio 2017

Visual Studio 2017 can be described as a versatile and feature-rich integrated development environment (IDE) for Windows 10 and the previous versions of Windows systems (Microsoft 2019d). For Darts MR, the main purpose for it is to enable developer to write the game code. Further utilization of the software will include debugging, testing and deploying the final game version to HoloLens. Visual Studio offers multiple different editions for a developer to utilize, including community, professional, and enterprise. For the sake of being completely free, the chosen edition in this thesis is the community edition. In order to enable full support for HoloLens development in Visual Studio, it is crucial that during the installation of Visual Studio, the developer opts to include the Universal Windows Platform Development tools from the features selection as depicted in Figure 16. Doing so includes the most recent version of Windows 10 SDK in the workload, which in turn supplies the necessary headers, libraries, metadata and utilities for creating Windows 10 applications further consumed by HoloLens.

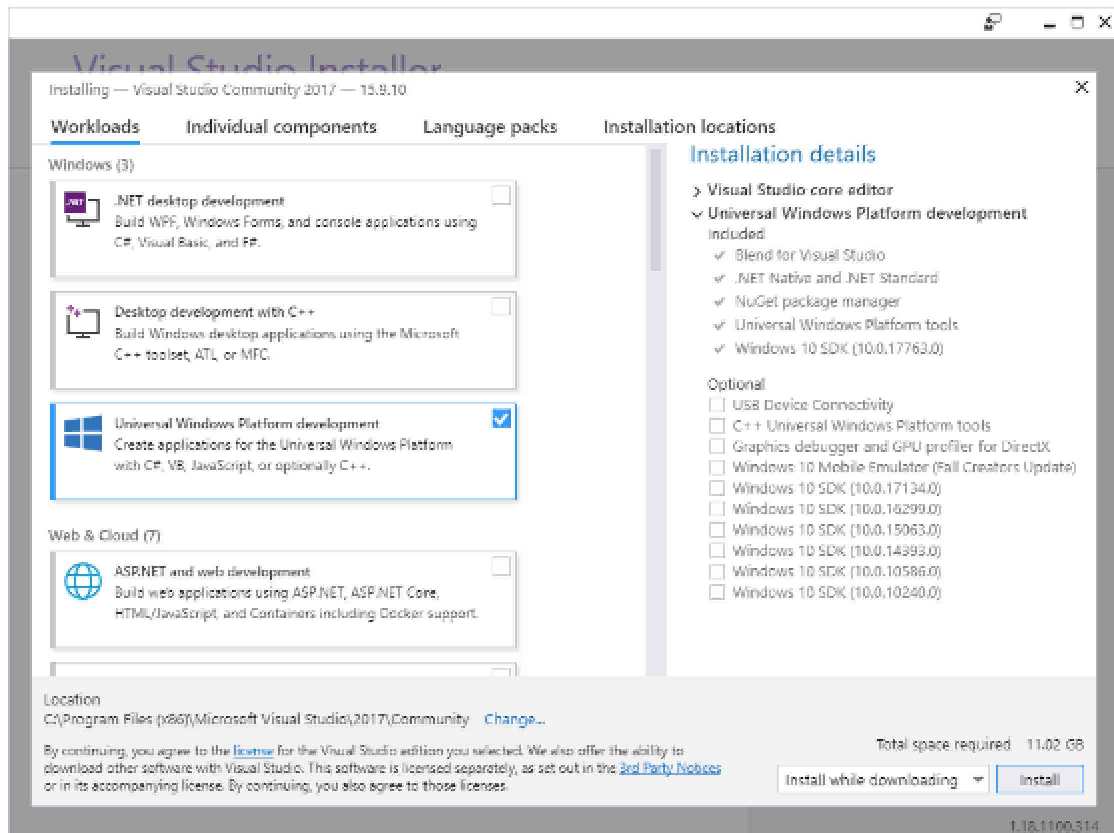

Figure 15. Selecting mandatory Visual Studio components

5.1.2 Unity

Unity is the chosen game engine for Darts MR. Equivalently to any other editing program such as Adobe Photoshop or 3ds Max, Unity comes with a sophisticated editor of its own, in which the developer is able to create, transform and build the game code and assets into a fully working piece of software. Constantly updated by its development team and adopted by millions of users all around the world, Unity has become a rather well documented tool with high compatibility. More particularly, and for the purpose of this thesis, the great support it offers for HoloLens is what makes it a preferred candidate for a game engine. The Unity game engine comes integrated with the support for WMR features and is capable of providing very simple approaches for creating unique mixed reality games (Microsoft 2019d). Unity has featured built-in support for WMR platform, including HoloLens since version 2017.2. At the time of writing this thesis, the development path forks and proposes three directions to choose from when selecting the platform version with Unity; 2017, 2018 and 2019 editions of Unity. Each propose quite similar internal features of development for HoloLens but differentiate when it comes to importing any external tool packages. For Darts MR, the chosen version of Unity is the 2018

edition. The main reason behind this is to utilize the most stable and up to date edition.

During the installation procedure of Unity, when the prompt to select which Unity components should be included in the workload appears, it is essential for the developer to select “UWP Build Support (.NET)” and “UWP Build Support (IL2CPP)” features as shown in Figure 17. These are Unity’s scripting back ends (frameworks that enable scripting in Unity) further supported by UWP’s and are necessary for developing holographic applications with Unity.

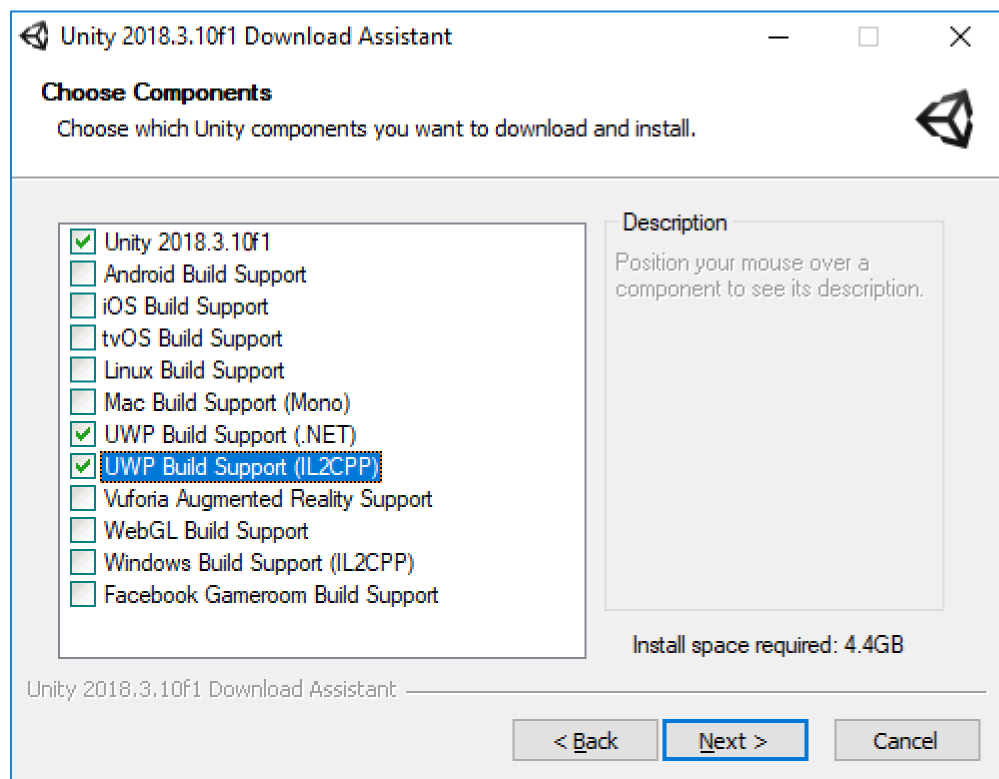

Figure 16. Recommended Unity components

5.1.3 Blender

The creation of 3D models further utilized the implementation process require a separate 3D modeling software. Towards that goal, a modeling tool known as Blender will be included. Using Blender is free, and it provides an open source 3D creation kit. The supported features include modeling, rigging, animation, simulation, rendering, compositing and motion tracking, as well as video editing and game creation tools. All together the features form the wholeness of the 3D pipeline. (Blender 2019.) Although the capabilities of

Blender extend to various fields related to creation of computer graphics, its primary goal for Darts MR is the creation of necessary 3D models.

5.1.4 HoloLens emulator

In addition to developing for a physical HoloLens device, the development can be relocated into a fully virtualized environment. A computer software capable of imitating functionality of a real physical version of HoloLens, the HoloLens emulator is a tool created by Microsoft to accelerate development of holographic applications. Suitable for scenarios in which utilizing a physical HoloLens device might be impossible due to varying reasons, the HoloLens emulator can prove a fine secondary option. However, due to some limitations (such as the lack of support for recognizing gestures) the HoloLens emulator may be considered more of a testing tool rather than complete replacement for an actual physical HoloLens device. Instead of utilizing sensors to read environmental and user input, the emulator simulates the functionalities with the usage of mouse, keyboard, or Xbox controller. The UI is similar to the UI of physical counterpart of the HoloLens, offering 3D space in which the participant is able to navigate and interact, as seen in Figure 18.

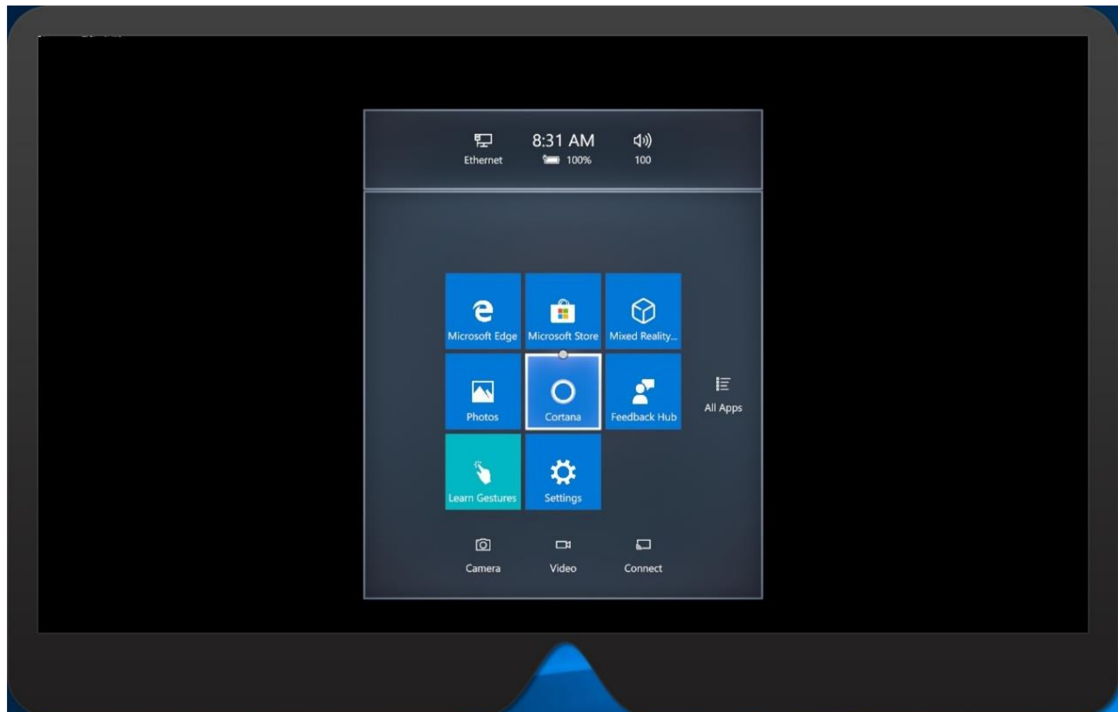

Figure 17. Holographic 3D menu of HoloLens emulator

5.2 Setting up the project

Once the installation and preparation of all required tools is complete, the first step in starting the actual development of the game is to create a new project in Unity. The process begins by developer launching Unity and clicking "New" in the projects window as depicted in Figure 18. This should be followed by a prompt on screen requesting the developer to enter a name for the project, browse a location for the project files to be saved in, and select a graphical template setting for the game to be rendered in. Due to the nature of holographic game objects, it is pivotal that the game environment and the content within Unity is rendered in 3D space. Therefore, the option for graphical template must be set to 3D. Additionally, an optional feature known as Unity Analytics is available. This feature enables Unity to collect project data in order to provide project metrics, benchmarks against similar projects, and insights into player behavior. However, for Darts MR, this feature is considered unnecessary and will be left out.

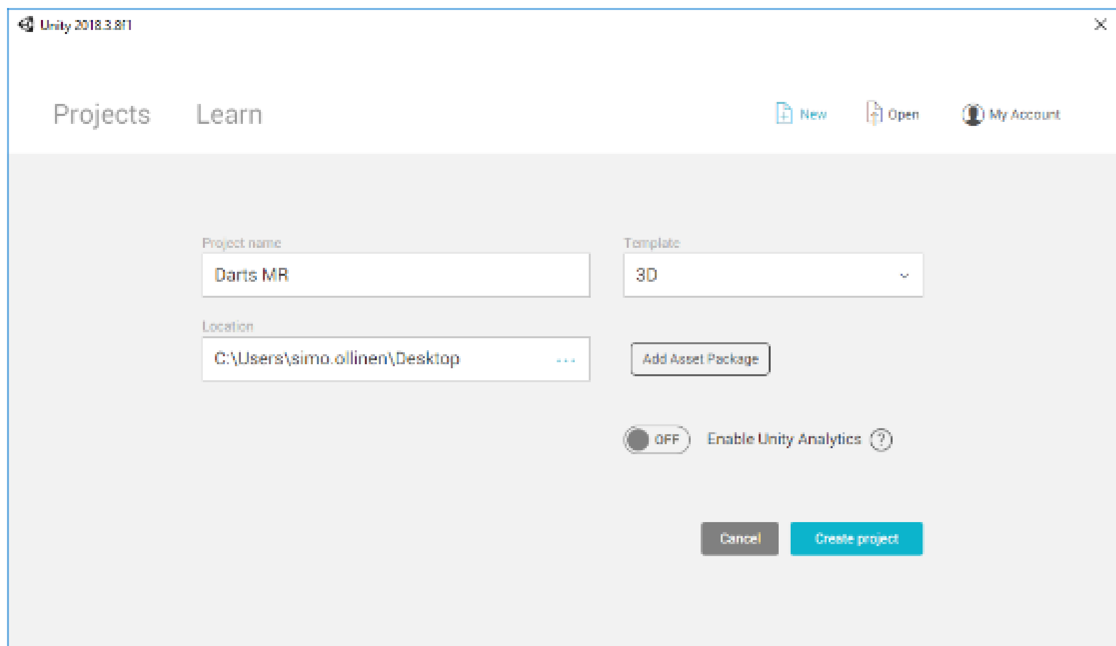

Figure 18. Creation of new Unity project

By creating a new project, Unity generates a new folder for the project with some initial core files included. At this point, the main editor window of Unity should open as well, giving the first glimpse of the main development workspace and its UI. Although the main window of Unity comes bundled with multitude of subpanels containing plethora of details and properties, describing

most of them is considered beyond the scope of this thesis. However, there are some key properties that should be explained in order to provide insight in creating MR experiences with Unity. As HoloLens limits the type of applications it can run exclusively to UWP applications, the game project created with Unity should be targeted towards the same platform. By default, after creating a new project, Unity targets the game projects towards PC, Mac and Linux platforms. However, it is possible to switch the platform of the project by selecting the desired platform from the list of platforms and pressing "Switch Platform" in the Build Settings pane of Unity, as shown in Figure 19.

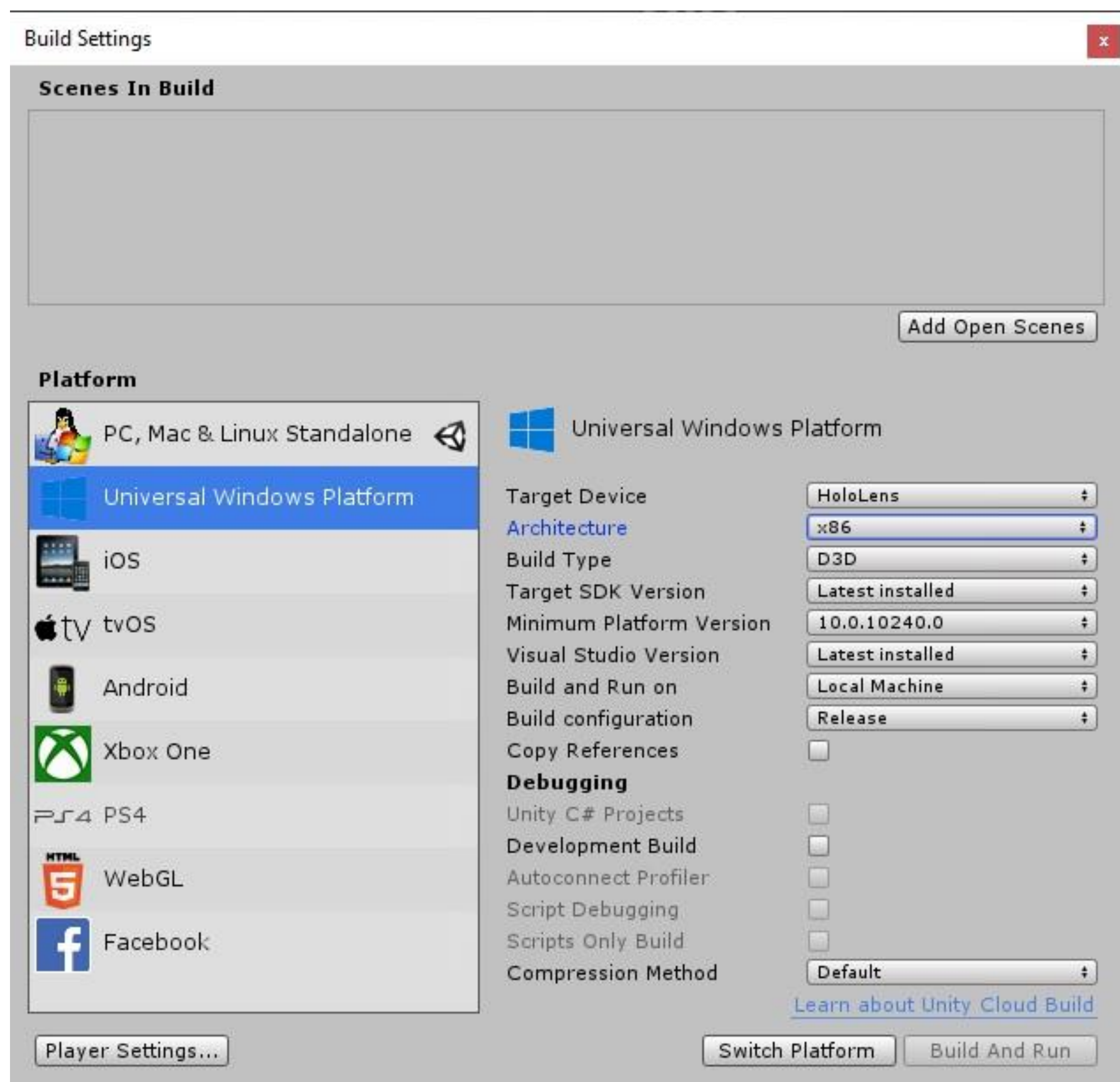

Figure 19. Build settings in Unity

By switching the platform to UWP, the application should become recognizable by HoloLens. This can be further proved by setting the target device property to "HoloLens". Although the physical processor included in HoloLens is capable of running 64-bit applications, the operating system of HoloLens is

merely 32-bit capable, and for that reason, the architecture property should be set to “x86”. Rest of the properties may be left to their default states. In addition to targeting the project towards correct platform, developing a holographic application requires that some virtual reality related libraries are included in the project. This can be configured in Player Settings pane accessible through the Build Settings panel. Checking “Virtual Reality Supported” under XR Settings submenu includes the necessary components required by Unity for developing holographic applications, as displayed in Figure 20. Finally, under Publishing Settings pane, checking “SpatialPerception” property from the list of Capabilities includes the support for several spatial mapping specific components of Unity, enabling HoloLens to understand them. Hereby, the project can be considered ready for further development of holographic content and functionalities.

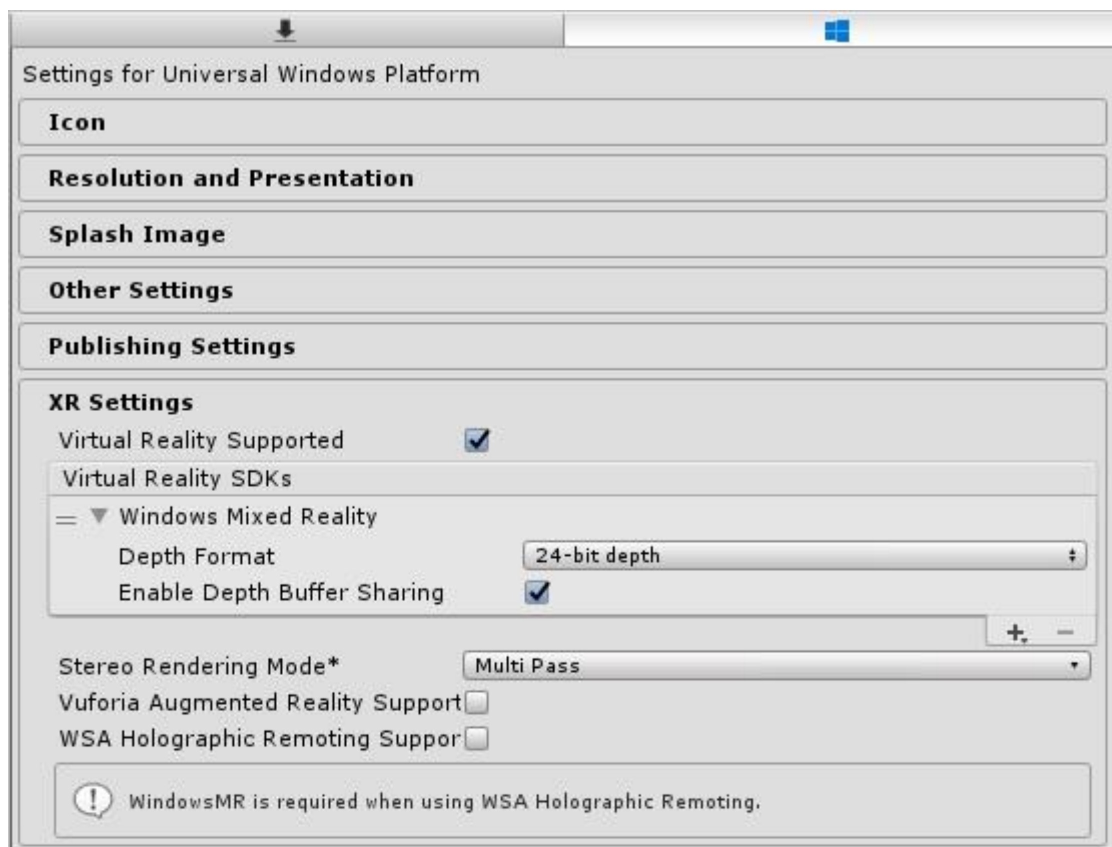

Figure 20. Inclusion of the support for virtual reality

5.3 Creation of fundamental game objects

Building of a game in Unity consists of setting up game objects. Furthermore, game objects can consist of smaller individual components (such as scripts

containing code, 3D model, audio clip, or all of them) that enable the game objects to behave in a way the developer desires to. The game objects occupy a large container known as the scene. Scene can be described as a 3D stage for the different game objects to act on while adhering to the internal rules defined in their logic (Thorn 2017). The objects, environments, and menus of a game are contained within scenes, and each unique scene file can be considered as a unique level of a game. Whilst working with scenes, the developer places the environments, obstacles and decorations into the scene, effectively designing and constructing the game piece by piece. For Darts MR, the functionality of the game is based on several fundamental game objects that are responsible for controlling certain features of the game, keeping track of actions and events, as well as interacting with the player input.

5.3.1 Camera

To provide feed of the game world, a game object known as the camera is required. Utilizing cameras is to transmit and visualize the look of the game world into the eyes of the player (Unity 2019a). Camera is a fundamental component of games created with Unity and provides an image of a particular viewpoint in the scene. Cameras can be configured to remain static in a fixed position or move around the game world independently, or more naturally by following a game object (such as a player's character). Furthermore, a scene may contain one or more cameras, and may be set to visualize the game world in any order and at any location on the screen. Through the inclusion of "Virtual Reality Supported" property previously mentioned, the camera is automatically configured to function with holographic applications and HoloLens. For a Darts MR, this allows the camera to adjust to the head movement and rotation of the participant and provide a visualization of the game world directly into the eyes, essentially positioning the player at center of the game world. However, it is necessary to configure several additional properties of the camera to fully optimize visual quality. Perhaps the most important property is the background color of the camera. By default, Unity configures cameras to visualize a completely virtual background image for the games, as they don't inherit a real world. However, in a mixed reality game, the real world should be just as visible as its virtual counterpart, and the background rendered by the camera shouldn't be blocked by an artificial illusion (Microsoft

2018a). This can be achieved by setting the Clear Flags property to “Solid Color” and Background color property to value black (RGBA 0, 0, 0, 0) as seen in Figure 21.

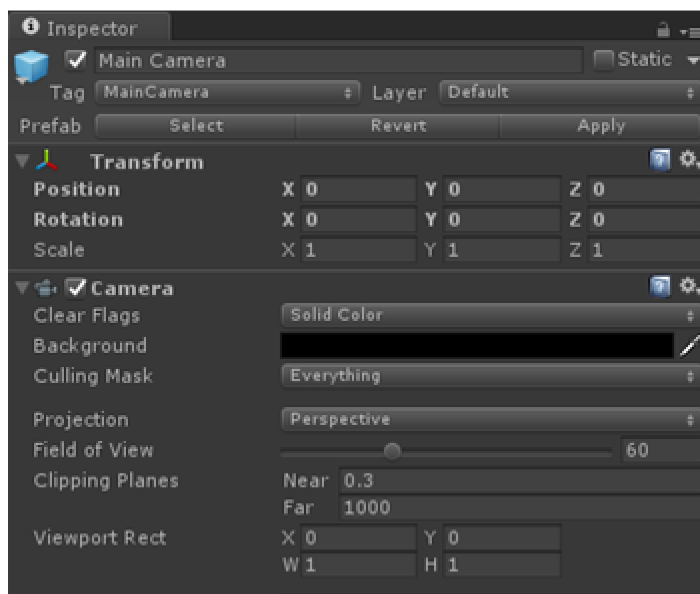

Figure 21. Camera settings for holographic environment

Finally, to complete the configuration of the camera, adjusting the “Clipping Planes” values correctly should prevent the camera from visualizing the game objects too close to the participant, which in turn could lead to possible user discomfort. This can be accomplished by setting the value of “Near” property to “0.85”.

5.3.2 GazeManager

“GazeManager” is an invisible game object responsible for handling the gaze input of the participant in Darts MR. It consists of a “GazeManager” script component. By tapping into the position, direction, and rotation data provided by the “Camera” game object, the “UpdateRaycast” method of the “GazeManager” generates a ray by utilizing Unity’s “Physics.Raycast” method as depicted in Figure 22 that represents the direction of the eyes of the participant within the game world. Additional data tracked by “GazeManager” includes whether the ray hit a game object (such as an UI menu item) or not, as well as the position, normal vector, and the transform (bundle of data including position, rotation, and scale of a game object) data of the game object hit. The data provided by the ray is publicly accessible for other game objects that

might require it. Especially for more optimal approach, a recommendation is that the gaze managing is handled in a single manager object rather than having multiple objects potentially interested in the data doing it themselves (Microsoft 2018e).

```
// Calculates the Raycast hit position and normal.
1 reference
private void UpdateRaycast()
{
    // Get the raycast hit information from Unity's physics system.
    Hit = Physics.Raycast(HeadPosition, GazeDirection, out RaycastHit hitInfo, MaxGazeDistance, RaycastLayerMask);

    // Update the HitInfo property so other classes can use this hit information.
    HitInfo = hitInfo;

    if (Hit)
    {
        // If the raycast hits a hologram, set the position and normal to match the intersection point.
        Position = hitInfo.point;
        Normal = hitInfo.normal;
        HitTransform = hitInfo.transform;
        lastHitDistance = hitInfo.distance;
    }
    else
    {
        // If the raycast does not hit a hologram, default the position to last hit distance in front of the user,
        // and the normal to face the user.
        Position = HeadPosition + (GazeDirection * lastHitDistance);
        Normal = GazeDirection;
        HitTransform = null;
    }
}
```

Figure 22. UpdateRaycast method of the "GazeManager" game object

5.3.3 SpatialMappingManager

Similar to the “GazeManager” game object, “SpatialMappingManager” is an invisible game object with the task of producing spatial mapping. The game object includes “SpatialMappingManager” script component. Spatial mapping capabilities are utilized to construct the game level in Darts MR. Implementing “SpatialMappingManager” proposed three ways of approach. The first approach included utilizing MRTK. The usage of the toolkit was limited to merely to gain insight of the capabilities of HoloLens by testing functionality of example code included in the toolkit. However, as the toolkit proved to be a rather quick and simple solution for implementing spatial mapping by utilizing complete and functional scripts, the amount of experience gained by doing so was deemed to be insufficient, and another way of implementing spatial mapping was to be established. The second approach consisted of utilizing two pre-existing components included in Unity, the “Spatial Mapping Renderer” and “Spatial Mapping Collider”. Equivalently to the MRTK, these components presented an easy and quick solution for implementing spatial mapping functionality using Unity. Both components include detailed properties the developer

can to configure and set up a fully working spatial mapping system without a need to write a single line of code. The reason for ultimately rejecting this way of approach was considered the same to the case with MRTK, the solution being excessively quick and simple to gain any valuable insight on the functionality. This essentially led to the decision of creating a custom script containing the necessary code for implementing spatial mapping by utilizing “SurfaceObserver”, a low-level component of Unity.

SurfaceObserver is a component of spatial mapping responsible for observing and setting one defined space by utilizing a group of bounding volumes. For that purpose, SurfaceObserver utilizes the SetVolumeAsAxisAlignedBox, SetVolumeAsOrientedBox, SetVolumeAsFrustum, and SetVolumeAsSphere methods to add the bounding volumes. (Newnham 2017.) As discussed earlier in the Spatial mapping chapter, each “SurfaceObserver” requires a defined space in which to scan for surfaces. For Darts MR, the decision of selecting volume as “AxisAlignedBox” was based on the shape of the room in which the implementation took place. The property is set within “Awake” method as seen in Figure 23.

```
0 references
void Awake()
{
    if (Instance == null)
        Instance = this;
    else if (Instance != this)
        Destroy(gameObject);

    surfaceObserver = new SurfaceObserver();
    surfaceObserver.SetVolumeAsAxisAlignedBox(Vector3.zero, new Vector3(4.0f, 4.0f, 4.0f));
}
```

Figure 23. Defining the bounding volume

Once the volume property is set, the “SurfaceObserver” waits for signal from the game to begin scanning surfaces every 2.5 second by continuously calling its “Update” method as seen in Figure 24.

```
1 reference
IEnumerator Observe()
{
    // While we are observing, update spatial mapping meshes every 2.5 seconds.
    var wait = new WaitForSeconds(timeBetweenUpdates);
    while (IsObserving)
    {
        surfaceObserver.Update(OnSurfaceChanged);
        yield return wait;
    }
}
```

Figure 24. Update method of the surfaceObserver

Furthermore, the method is passed with a delegate (a reference type containing a method) to call “OnSurfaceChanged” method in order to manage the scan results. The “OnSurfaceChanged” method is tasked with keeping track of changes within the physical environment and may be considered most fundamental part of the functionality of “SurfaceObserver”. Depending on changes in the environment, the “OnSurfaceChanged” method updates the mesh of spatial mapping by either adding, updating, or removing surfaces, by utilizing separate game objects with the purpose of visualizing the mesh of spatial mapping. The type of update chosen is further dictated by “SurfaceChange”, an enumeration (a bundle of named constants) of different types of events recognized by the “SurfaceObserver” that represent the changes in the environment. The events include “Add”, “Update”, and “Remove”. For both “Add” and “Update” events, “OnSurfaceChanged” method first performs a check to determine whether an identity of a scanned surface is contained within “SurfacesToBeRemoved” collection, and if so, removes the identity from the collection as seen in Figure 25.

```
1 reference
void OnSurfaceChanged(SurfaceId surfaceId, SurfaceChange changeType, Bounds bounds, DateTime updateTime)
{
    switch (changeType)
    {
        case SurfaceChange.Added:
        case SurfaceChange.Updated:
            {
                if (surfacesToBeRemoved.ContainsKey(surfaceId.handle))
                {
                    surfacesToBeRemoved.Remove(surfaceId.handle);
                }
```

Figure 25. Removal of surface identity

This is followed by another check to determine whether an identity of a scanned surface is not contained in a “cachedSurfaces” collection, and if so, the method creates a new surface game object for it and adds necessary

mesh related data to it as well as inserting the identity of the scanned surface into the “cachedSurfaces” collection as depicted in Figure 26.

```
if (!cachedSurfaces.TryGetValue(surfaceId.handle, out GameObject surface))
{
    surface = new GameObject();
    surface.name = string.Format("surface_{0}", surfaceId.handle);
    surface.layer = LayerMask.NameToLayer("SpatialSurface");
    surface.transform.parent = transform;
    surface.AddComponent<MeshRenderer>();
    surface.AddComponent<MeshFilter>();
    surface.AddComponent<WorldAnchor>();
    surface.AddComponent<MeshCollider>();
    cachedSurfaces.Add(surfaceId.handle, surface);
}
```

Figure 26. Creation of a new surface game object

The data from the newly created surface game object is subsequently used to populate a separate “surfaceData” struct (a value type able to contain groups of variables) that is responsible for indicating what information “SurfaceObserver” requires to generate the meshes. Additional data inserted include “trianglesPerCubicMeter” which indicates the level of quality HoloLens renders the meshes, as well as the sign to mark the mesh for creation process as displayed in Figure 27.

```
SurfaceData surfaceData;
surfaceData.id.handle = surfaceId.handle;
surfaceData.outputMesh = surface.GetComponent<MeshFilter>() ?? surface.AddComponent<MeshFilter>();
surfaceData.outputAnchor = surface.GetComponent<WorldAnchor>() ?? surface.AddComponent<WorldAnchor>();
surfaceData.outputCollider = surface.GetComponent<MeshCollider>() ?? surface.AddComponent<MeshCollider>();
surfaceData.trianglesPerCubicMeter = 1000;
surfaceData.bakeCollider = true;
```

Figure 27. Creation and population of surfaceData struct

The “surfaceData” struct is then passed to “SurfaceObserver.RequestMeshAsync” method, along with a delegate calling “OnDataReady” method in order to begin process of creating the mesh data (baking) as illustrated in Figure 28.

```
if (!surfaceObserver.RequestMeshAsync(surfaceData, OnDataReady))
{
    Debug.LogWarningFormat("Is {0} not a valid surface", surfaceData.id);
}
break;
```

Figure 28. RequestMeshAsync method

Finally, in the case where a surface was removed by the scan result, the "OnSurfaceChanged" method is concluded by a check determining whether an identity of a removed surface is included in "cachedSurfaces" collection, and if so, sets that surface to be removed later by adding it to the "surfacesToBeRemoved" collection as shown in Figure 29.

```
case SurfaceChange.Removed:
    {
        if (cachedSurfaces.TryGetValue(surfaceId.handle, out GameObject surface))
        {
            // Instead of removing surfaces on instant, pass the surfaces marked for removal into a collection
            surfacesToBeRemoved.Add(surfaceId.handle, Time.time + removalDelay);
        }
        break;
    }
```

Figure 29. Marking an expired surface for removal

In the case of successful creation of a new or updated surface, the "OnDataReady" method performs a check to determine whether the identity of the surface created has been added into "cachedSurfaces collection". In that case, the surface is simply assigned a material to visualize it as displayed in Figure 30.

```
1 reference
void OnDataReady(SurfaceData bakedData, bool outputWritten, float elapsedBakeTimeSecond)
{
    if (!outputWritten)
        return;

    if (cachedSurfaces.TryGetValue(bakedData.id.handle, out GameObject surface))
    {
        MeshRenderer renderer = surface.GetComponent<MeshRenderer>();

        if (surfaceMaterial != null)
        {
            renderer.sharedMaterial = surfaceMaterial;
        }
        renderer.enabled = SurfacesVisible;
    }
}
```

Figure 30. Visualization of a surface

The "Update" method of "SpatialMappingManager" script monitors the surface identities contained in "surfacesTobeRemoved" and "cachedSurfaces" collections, and removes the identities marked for removal previously in the "OnSurfaceChanged" method. Additionally, the surface game object responsible for visualizing a piece of the spatial map is destroyed in the process as described in Figure 31.

```
0 references
void Update()
{
    var surfaceIds = surfacesToBeRemoved.Keys;
    foreach (int surfaceId in surfaceIds)
    {
        if (surfacesToBeRemoved[surfaceId] >= Time.time)
        {
            surfacesToBeRemoved.Remove(surfaceId);

            if (cachedSurfaces.TryGetValue(surfaceId, out GameObject surface))
            {
                cachedSurfaces.Remove(surfaceId);
                Destroy(surface);
            }
        }
    }
}
```

Figure 31. Removal of surfaces within Update method

5.3.4 GameManager

The game object responsible for keeping track of the game logic is the “GameManager”. It is an invisible game object similarly to “GazeManager” and “SpatialMappingManager” game objects and consists of a script component including the necessary code to function. The tasks include tracking player score, the amount of darts left, as well as the scanning time given for spatial mapping to complete. Additionally, managing game states is part of the core functionality. The game states depict the current state of the game and are considered interchangeable with the term game state in this thesis. The game is divided into five different game states which include “MainMenu”, “Scanning”, “Placing”, “Playing”, and “GameOver” game states. Each game state calls for a specific coroutine (a type of method capable of suspending its execution for a given time limit) that manages game logic pivotal for each game state of the game. Handling which game state is active at any given time is tasked to “OnStateChanged” method. By utilizing Unity’s “StopAllCoroutines” method, “OnStateChanged” is capable of suspending every coroutine and executing only the particular coroutine connected to the current game state of the game as seen in Figure 32.

```
1 reference
void OnStateChanged()
{
    StopAllCoroutines();

    switch (_currentState)
    {
        case GameState.MainMenu:
            StartCoroutine(MainMenuStateRoutine());
            break;
        case GameState.Scanning:
            StartCoroutine(ScanningStateRoutine());
            break;
        case GameState.Placing:
            StartCoroutine(PlacingStateRoutine());
            break;
        case GameState.Playing:
            StartCoroutine(PlayingStateRoutine());
            break;
        case GameState.GameOver:
            StartCoroutine(GameOverStateRoutine());
            break;
    }
}
```

Figure 32. Management of game states

The “MainMenuStateRoutine” coroutine executes functionalities related to “MainMenu” game state of Darts MR as depicted in Figure 33. It includes adjusting “RaycastLayerMask” property of the “GazeManager” game object to “UI”, which will allow HoloLens gaze to only interact with game elements tagged as “UI”, as “MainMenu” game state of the game consists solely of “UI” elements. Other functionality that did not make it in time for the writing of this thesis include displaying the main menu as well as the button UI elements utilized to navigate the menu.

```
1 reference
IEnumerator MainMenuStateRoutine()
{
    GazeManager.Instance.RaycastLayerMask = LayerMask.GetMask("UI");

    // Show the main menu
    // Buttons to start game, options, leaderboards and quit the game
    yield return null;
}
```

Figure 33. MainMenuStateRoutine coroutine

The “ScanningStateRoutine” coroutine is utilized at the start of a match to activate functionality of the “SpatialMappingManager” game object as seen in Figure 34. It calls for the “SurfaceObserver” component to begin observing the

environment by setting the “IsObserving” property to true as well as making the surface meshes visible to the player. The duration of the coroutine is governed by the “scanningTime” variable, after which the game enters the next game state of “Placing” by setting the “CurrentState” variable.

```
1 reference
IEnumerator ScanningStateRoutine()
{
    // Activate SpatialMappingManager to scan the surfaces with SurfaceObserver
    // and visualize the spatial mapping meshes

    Debug.Log("Drawing spatial mapping started: Move around the room");
    SpatialMappingManager.Instance.IsObserving = true;
    SpatialMappingManager.Instance.SurfacesVisible = true;

    yield return new WaitForSeconds(scanningTime);

    CurrentState = GameState.Placing;
}
```

Figure 34. ScanningStateRoutine coroutine

The next game state in the game logic is placement of the “Board” game object as depicted in the Figure 35. The “PlacingStateRoutine” coroutine includes the tasks for setting the “RaycastLayerMask” property of the “GazeManager” game object to “SpatialSurface” and “UI” in order for the player gaze to be able to interact with the surface meshes scanned in previous game state as well as with the possible UI elements. “SurfaceObserver” component is requested to stop scanning the environment by setting the “IsObserving” property of the “SpatialMappingManager” to false, whilst still keeping the surface meshes visible. The coroutine then proceeds to instantiate the virtual “Board” game object in front of the player at distance of roughly 1.5m and oriented towards the player. The “Board” game object is added with “Placeable” script component that contains the code responsible for the functionality. The coroutine is executed for as long as the “Board” game object remains unplaced, giving the player necessary time to find a proper placement position for it. Once placed, the “Placeable” script is destroyed, fixing the board in its place as well as suspending the functionalities related to placement of the “Board” game object. From that point, the game advances to the next game state.

```
1 reference
IEnumerator PlacingStateRoutine()
{
    // Stop the SurfaceObserver
    // Spawn the board and place it on a surface by following player's gaze

    GazeManager.Instance.RaycastLayerMask = LayerMask.GetMask("SpatialSurface", "UI");

    SpatialMappingManager.Instance.IsObserving = false;
    Debug.Log("Drawing spatial mapping ended: Place the board");

    Board = Instantiate(_boardPrefab, Camera.main.transform.position + (Camera.main.transform.forward * 1.5f), Quaternion.identity);

    var placeable = Board.GetComponent<Placeable>();

    if (placeable == null)
        placeable = Board.AddComponent<Placeable>();

    while (!placeable.Placed)
    {
        yield return null;
    }

    Destroy(placeable);
    Board.AddComponent<WorldAnchor>();

    CurrentState = GameState.Playing;
}
```

Figure 35. PlacingStateRoutine coroutine

Perhaps the most important game state of the game is the “Playing” game state, responsible for executing the “PlayingStateRoutine” coroutine as shown in Figure 36. Firstly, the coroutine hides the surface meshes previously scanned to give the player a more realistic view. Secondly, it sets the “RaycastLayerMask” property to include “SpatialSurface”, “Hologram” and “UI” elements. Thirdly, the game state is supposed to display the player score and amount of darts left, which at the time of writing this thesis are not yet implemented in the game. Finally, the game instantiates a “Dart” game object in front of the player by calling “SpawnDart” method. The player may pick up the dart by using hand gestures and throw at the “Board” game object previously placed. After each throw, the game instantiates a new “Dart” for the player to throw at the “Board”, up to eight “Darts” in total after which the game will end

```
1 reference
IEnumerator PlayingStateRoutine()
{
    // Hide the spatial mapping mesh
    // Display player score and darts
    // Spawn the dart for the player

    Debug.Log("Hiding Spatial Mapping meshes: Enjoy playing!");
    SpatialMappingManager.Instance.SurfacesVisible = false;
    GazeManager.Instance.RaycastLayerMask = LayerMask.GetMask("SpatialSurface", "Hologram", "UI");

    while (!gameOver)
    {
        if (Dart == null || Dart.GetComponent<Throwable>().HasLanded == true)
            Dart = SpawnDart();
        yield return null;
    }

    CurrentState = GameState.GameOver;
}
```

Figure 36. PlayingStateRoutine coroutine

The “SpawnDart” method is utilized by “PlayingStateRoutine” coroutine to instantiate the “Dart” game objects in front of the player at a set distance and oriented towards right as seen in Figure 37. The game object is attached with “Throwable” script component, containing the functionality of the “Dart” game object.

```
private GameObject SpawnDart()
{
    GameObject Dart = Instantiate(
                        _dartPrefab,
                        Camera.main.transform.position + (Camera.main.transform.forward * 1.0f),
                        Quaternion.Euler(Camera.main.transform.right));
    var throwable = Dart.GetComponent<Throwable>();

    if (throwable == null)
        throwable = Dart.AddComponent<Throwable>();

    return Dart;
}
```

Figure 37. SpawnDart method

The final game state of the game is “GameOver”. It executes the “GameOverStateRoutine” coroutine that is tasked with displaying the overall score of the player on a large panel and enabling the player to enter their name to be saved in the leaderboards. Additionally, showing the buttons to play again or returning back to main menu would be included. However, none of the aforementioned features made it in time to implementation phase at the writing of this thesis.

5.3.5 Cursor

The purpose of “Cursor” game object is to help the player to understand where their eyes focus in the game world at all times as earlier discussed in the Gaze chapter. The game object includes components for a small circular 3D model to provide simple and clear look, as well as a script that contains the code for the functionality. The method responsible for enabling the “Cursor” to move to and align onto the surface of other game objects by referencing the gaze ray emitted by the “GazeManager” game object in the “LateUpdate” method as illustrated in Figure 38. “Cursor” applies a small offset between the game object hit and the “Cursor” in order to avoid the meshes of the objects from clipping into each other. Additionally, should the “Cursor” hit another

game object, the color of the “Cursor” will smoothly change to green, notifying the player that their eyes are focused on an interactable game object.

```
0 references
void LateUpdate()
{
    if (GazeManager.Instance == null || meshRenderer == null)
    {
        return;
    }

    // Place the cursor at the calculated position.
    gameObject.transform.position = GazeManager.Instance.Position + GazeManager.Instance.Normal * distanceFromCollision;

    // Reorient the cursor to match the hit object normal.
    gameObject.transform.up = GazeManager.Instance.Normal;
    gameObject.transform.rotation *= cursorDefaultRotation;

    cursorTargetColor = defaultColor;

    if (GazeManager.Instance.HitTransform != null && (interactiveLayers.value & (1 << GazeManager.Instance.HitTransform.gameObject.layer)) > 0)
    {
        cursorTargetColor = interactiveColor;
    }

    meshRenderer.material.color = Color.Lerp(meshRenderer.material.color, cursorTargetColor, 2.0f * Time.deltaTime);
}
```

Figure 38. LateUpdate method of the "Cursor" game object

5.3.6 Dart

The “Dart” game object is perhaps the most interactable object within Darts MR. The player is able to pick it up in their hands and throw it at a board by using gestures. “Dart” game object consists of a mesh (collection of vertices), a rigidbody (affects positional data of an object through Unity’s physics simulation), and script components to give it the necessary functionality in the game. The mesh is used to visualize the 3D model of the “Dart” in the game as seen in Figure 39. Rigidbody component on the other hand includes the simulation of physics to make the game object behave realistically by introducing properties such as mass and gravity. Finally, the “Throwable” script component contains the code necessary for the “Dart” to function properly within the game logic.

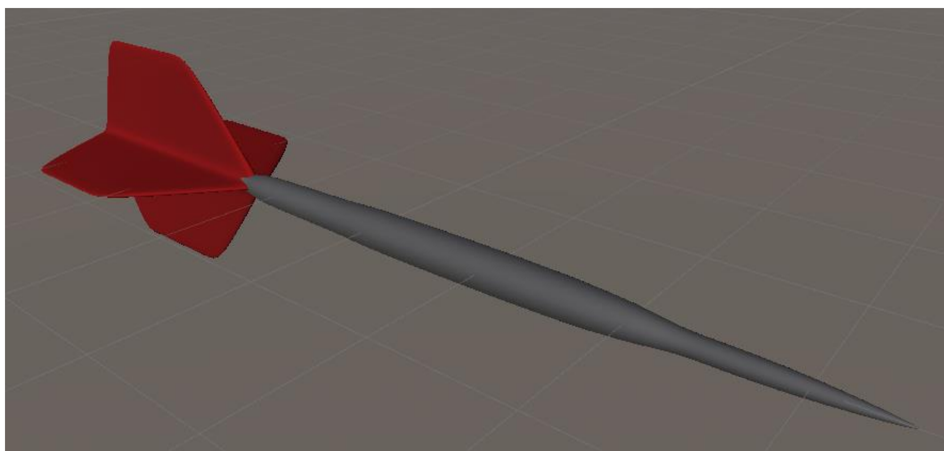

Figure 39. Mesh of the dart

The core functionality contained in the script component includes recognizing gestures when interacting with the "Dart" game object. More precisely, the gestures are based on the manipulation gestures discussed earlier in the HoloLens chapter. The script utilizes "GestureRecognizer" component of Unity, a component allowing the developer to specifically set the types of gestures recognized by the system as depicted in Figure 40.

```
manipulationRecognizer = new GestureRecognizer();
manipulationRecognizer.SetRecognizableGestures(GestureSettings.ManipulationTranslate);
```

Figure 40. Definition of manipulation gestures

Manipulation gestures recognized by the "GestureRecognizer" component consist of four different actions (delegates that take zero parameters), "ManipulationStarted", "ManipulationUpdated", "ManipulationCompleted", and "ManipulationCanceled". Each action is triggered depending on how the environmental cameras of HoloLens manage to sense the player hands. Furthermore, each action can be set to call upon a particular method or block of code. The "ManipulationStarted" action is triggered when "GestureRecognizer" is informed that player has started manipulating (such as using a manipulation gesture to pick up the "Dart") as displayed in Figure 41. The action is registered to "ManipulationStartedEventArgs" class which provides the data relevant to the manipulation event. The data is then captured by a lambda expression (anonymous method, or a block of code treated as an object) and is utilized to process the data. The lambda performs a check to determine whether the data contains information about the position of the player's hand, and if so, it disables the gravity property of the rigidbody component as well as outputting the data into a "handPosition" variable. The data is then utilized to position the "Dart" in the position of the player's hand as well as to orient it towards the direction player is looking at. Finally, additional variables saved for later use include "manipulatedObjectOriginalPos" to track the last known positional data of the "Dart", and "manipulationStartTimeStamp" to save the current time since the start of the game.

```
manipulationRecognizer.ManipulationStarted += (ManipulationStartedEventArgs) =>
{
    if (!ManipulationStartedEventArgs.sourcePose.TryGetPosition(out Vector3 handPosition))
        return;

    rb.useGravity = false;
    gameObject.transform.position = handPosition + Camera.main.transform.forward * 0.25f + Camera.main.transform.TransformVector(FingertipsOffset);
    gameObject.transform.rotation = Quaternion.LookRotation(Camera.main.transform.forward, Camera.main.transform.up);
    manipulatedObjectOriginalPos = gameObject.transform.position;
    manipulationStartTimeStamp = Time.time;

    Debug.Log("Dart picked!");
};
```

Figure 41. Functionality called by ManipulationStarted action

Due to each manipulation related action sharing a very similar pattern of implementation, the explanation will focus on the core functionalities for the rest of the actions in order to avoid unnecessary repetition. “ManipulationUpdated” action utilizes cumulativeDelta (Total distance moved since the start of the manipulation gesture) property from the registered data to continuously calculate and update the position and rotation of the “Dart” game object. Additionally, the data is saved into “accumulativeVelocity” variable for later use. In order to send the “Dart” flying when player finishes the manipulation gesture, “ManipulationCompleted” action is triggered as can be seen in Figure 42. The “Dart” is given “acceleration” variable calculated from previously saved “accumulativeVelocity” and “manipulationStartTimeStamp” variables as well as the current time since the start of the game. The value of “acceleration” variable is further altered by several other variables in order to remain within certain limits of velocity to avoid unnatural behavior as well as to help adjusting the final value of the velocity. Finally, the gravity property of the rigidbody component is enabled, and the final “acceleration” value is applied to the velocity property of the rigidbody, setting the “isFlying” variable true and sending the “Dart” flying. The “GestureRecognizer” calls for “StopCapturingGestures” method in order to prevent the player from accidentally performing manipulation gestures on the “Dart” once it has been thrown. Additionally, a “DelayedDestroyIfMissed” coroutine is called to destroy the “Dart” game object after five seconds, should it miss the board.

```
const float minAcceleration = 0.0009f;
const float maxAcceleration = 0.3f;

Vector3 acceleration = accumulativeVelocity / (Time.time - manipulationStartTimeStamp);

if (acceleration.magnitude > minAcceleration)
    acceleration = acceleration.normalized * ((acceleration.magnitude / maxAcceleration) * throwForceMultiplier);

rb.useGravity = true;
rb.velocity = acceleration;
isFlying = true;

if (manipulationRecognizer != null)
    manipulationRecognizer.StopCapturingGestures();

StartCoroutine(DelayedDestroyIfMissed());
Debug.Log("Dart thrown!");
```

Figure 42. Functionality called by ManipulationCompleted action

In the case HoloLens loses track of player's hand, the "ManipulationCanceled" action is triggered. This action is utilized to simply reposition the "Dart" game object back to its last known position by utilizing the "manipulatedObjectOriginalPos" variable earlier saved in the "ManipulationStarted" action. In order to further increase realism of the "Dart" once thrown, the orientation of the tip is continuously altered in the "FixedUpdate" method of the "Throwable" script by calling "Quaternion.LookRotation" method that creates a rotation based on the direction the "Dart" is flying at. To calculate physical contact between other game objects, the "Dart" game object includes a collider component. Collider is an invisible component attached to a game object that mimics the shape of the game object's mesh to calculate physical collisions. However, a collider does not need to be precisely the shape of the game object's mesh. Frequently, a close approximation is considered efficient enough. (Unity 2019b.) To utilize collisions, a method "OnTriggerEnter" is called as shown in Figure 43. The method checks whether the other game object hit matches to the board through comparison of tags (user set labels for game objects), and then sets the "IsFlying" variable of the "Throwable" script and useGravity property of the rigidbody to false, as well as enabling the isKinematic property of the rigidbody to stick "Dart" to the board. Finally, the player score tracked by the "GameManager" game object is increased and the remaining darts available to the player is reduced.

```
void OnTriggerEnter(Collider other)
{
    if (other.tag == "Board")
    {
        isFlying = false;
        rb.useGravity = false;
        rb.isKinematic = true;
        HasLanded = true;
        GameManager.Instance.PlayerScore += 10;
        GameManager.Instance.PlayerDarts -= 1;
    }
}
```

Figure 43. Collision handling of the dart

5.3.7 Board

Similar to the “Dart” game object, “Board” represents an interactable game object. By utilizing gaze to position and air tap gesture to place, the player can decide where to place the “Board”. The game object consists of a mesh, collider, and script components. Identically to the “Dart” game object, “Board” utilizes mesh component to visualize the 3D model as portrayed in Figure 44. The collider is used to calculate the collisions with other game objects. Finally, the “Placeable” script contains the code for fundamental functionalities.

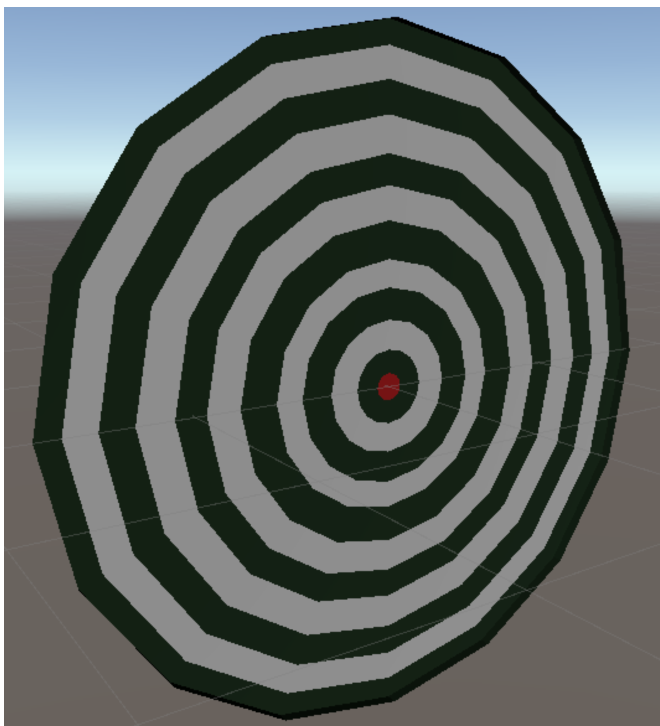

Figure 44. Mesh of the board

Equally to manipulation gestures utilized to manage the positioning of the "Dart" game object, "Board" applies air tap gestures to confirm the positioning. By including the "GestureRecognizer" component, the script t is set to recognize air tap gestures as shown in Figure 45.

```
// Set up a Gesture Recognizer to detect our tap gestures
tapGestureRecognizer = new GestureRecognizer();
tapGestureRecognizer.SetRecognizableGestures(GestureSettings.Tap);
```

Figure 45. Definition of tap gestures

To capture the air tap gestures detected by HoloLens, actions are utilized similarly to the manipulation detection in previous chapter. The script includes "Tapped" action which is registered to "TappedEventArgs" data class in order to receive the information relevant to air tap gestures. Equivalently to manipulation handling, a lambda expression is utilized to capture the data received and then manage it. Within the lambda expression, on each air tap gesture detected by HoloLens, a check is first performed to determine whether the player is currently placing the "Board". Depending on the result, either "OnPlacementStart" or "OnPlacementStop" method is called as displayed in Figure 46.

```
tapGestureRecognizer.Tapped += (args) =>
{
    // Attempt to place the board when tap gesture is detected
    if (!IsPlacing)
    {
        Debug.Log("Placement started!");
        OnPlacementStart();
    }
    else
    {
        Debug.Log("Attempting to place!");
        OnPlacementStop();
    }
};
```

Figure 46. Functionality called by Tapped action

The two methods are responsible for enabling and disabling the placement of the "Board" game object. "OnPlacementStart" method enables the "Board"

game object to follow gaze of the player in order to continuously update its position in the game world by setting “IsPlacing” property to true. The collider component is also activated so that the “Board” may simulate collisions with other game objects (such as spatially mapped objects in the physical environment). Additionally, the “Cursor” game object is deactivated and hidden to prevent it from interfering with the placement of the “Board”. The “OnPlacementStop” method performs a check to determine whether the surface is flat enough for the “Board” to fit on it by calling “IsValidatedPlacement” method. If the method considers a surface valid enough, the method then provides a value for “targetPosition” variable utilized to calculate the final position of the “Board”. Furthermore, the value of the “targetPosition” variable is incremented with a small offset to prevent clipping. Finally, the “Board” is oriented to face opposite direction from the surface, and the execution of the method is considered complete by setting “IsPlacing” property to false.

Within the “Update” method of the “Placeable” script, a check is first performed to determine whether “IsPlacing” property is set to true. If so, “Move” method is called to continuously update the position of the “Board” game object according to current gaze position data passed by “GazeManager” game object. The “Board” is moved by hovering it above the spatial surfaces, aligning to their normal vectors whilst following the precise position of the gaze of the player. Additionally, in order to aid the player visualizing surfaces considered valid by the “IsValidatedPlacement” method, a shadowing effect is applied as discussed in the Spatial mapping subchapter earlier. “DisplayShadow” method utilizes a simple rectangular plane to provide a separate area aligned in between the “Board” and the spatial surfaces behind it. The plane is colorized with either green or red material to indicate whether the surface beneath the “Board” is considered valid as illustrated in Figure 47.

Figure 47. Indication of a valid surface

Once the position of the “Board” has been confirmed by the player performing an air tap gesture, and thus calling “OnPlacementStop” method, the “Board” is smoothly placed on the surface behind it by utilizing a “Lerp” method. The method calculates the final position of the “Board” from information based on the current position in the game world, the value of “targetPosition” variable previously calculated, as well as values from variables “placementVelocity” and “dist” which consist of values for determining the velocity of placement and the distance between the “Board” and the surface behind it. Finally, “Placed” property is set true, marking the completion of the “Board” placement and signaling “GameManager” to shift to next state of the game. Additionally, the “Cursor” is activated again to aid player focusing their gaze.

5.3.8 UI

The UI was a rather thoroughly discussed subject in the earlier chapters of this thesis. An approach of diegetic design was considered most appropriate due to the nature of MR and holographic applications. However, due to the time restrictions, a simplified way of non-diegetic implementation of the UI was chosen. Furthermore, the implementation of the UI is very much work in progress and for that reason the subject will be covered very briefly. Currently, the sole UI element of Darts MR is “PlayerStats” game object displaying the player score and amount of darts left. For this purpose, UI components fea-

tured by Unity are applied to create the necessary elements. The “PlayerStats” game object consists of several components including two scripts, a canvas (a parent area containing all other UI elements), an image (core graphic element of UI systems in Unity), and two textmeshes (3D container for displaying text) as depicted in Figure 48.

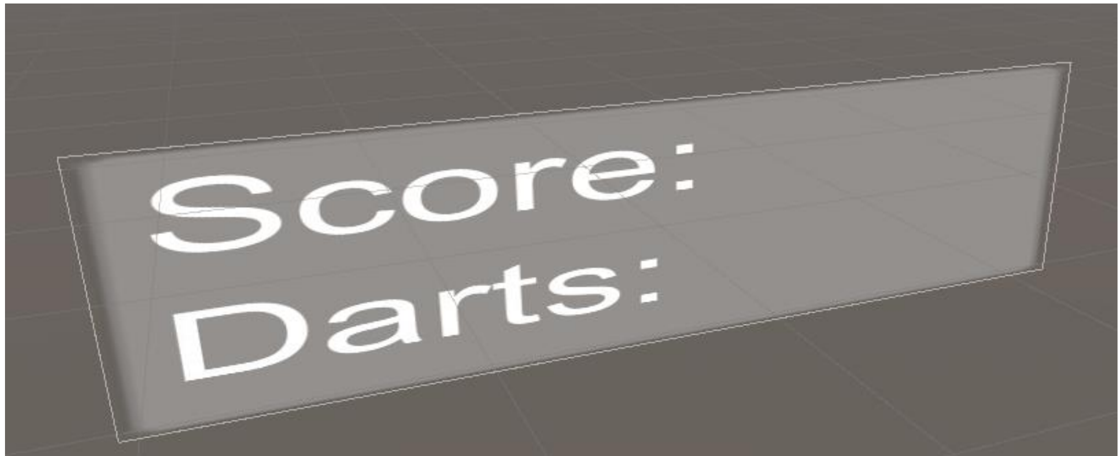

Figure 48. PlayerStats UI game object

The canvas UI element parents the image component, the element utilized to generate the background image for the “PlayerStats” game object. On the background is subsequently rendered two textmesh components displaying the player score and darts. The textmeshes are referenced in “StatusText” script component in order to allow the textmeshes dynamically update the contained text when necessary. Within the “UpdateStatusText” method, the data related to score and darts is passed from “GameManager” game object tasked with keeping track of the data. The data is then passed to the text property for both textmesh components, resulting in a real time visualization of both the score and darts data as shown in Figure 49.

```
1 reference
private void UpdateStatusTexts()
{
    string score = GameManager.Instance.PlayerScore.ToString();
    string darts = GameManager.Instance.PlayerDarts.ToString();
    statusTexts[0].text = "Score: " + score;
    statusTexts[1].text = "Darts: " + darts;
}
```

Figure 49. Passing tracked game data to PlayerStats UI game object

Instead locking the “PlayerStats” game object in fixed position in the game world, it is designed to follow the gaze of the player from a distance. By utilizing “UpdatePosition” method within “BillBoard” script component, the position of the game object is anchored to top left corner of the vision of the player by setting the value of “targetPosition” variable to match desired location within the game coordinate system, enabling it to smoothly reposition towards the point whenever the player moves their head through utilization of “Lerp” method as seen in Figure 50.

```
1 reference
void UpdatePosition()
{
    targetPosition = GazeManager.Instance.HeadPosition
    + (GazeManager.Instance.GazeDirection * 3.0f)
    + (Camera.main.transform.up * 0.2f)
    + (Camera.main.transform.right * -0.2f);
    float distance = Vector3.Distance(gameObject.transform.position, targetPosition);
    gameObject.transform.position = Vector3.Lerp(gameObject.transform.position, targetPosition, Time.deltaTime * 6.0f);
}
```

Figure 50. Positioning of the PlayerStats UI game object

5.4 Building and deploying the project

The process of packing up all the game objects and constructing an executable application supported by HoloLens begins by building the project in Unity. Building the project can be performed by opening the build settings of Unity and pressing the “Build” button. This results in creation of a folder structure containing all the required game files, as well as a Visual Studio solution file further utilized to deploy the game to HoloLens. By opening the solution file, the developer can deploy the application to HoloLens by selecting “Deploy Solution” from the Build menu of Visual Studio as displayed in Figure 51.

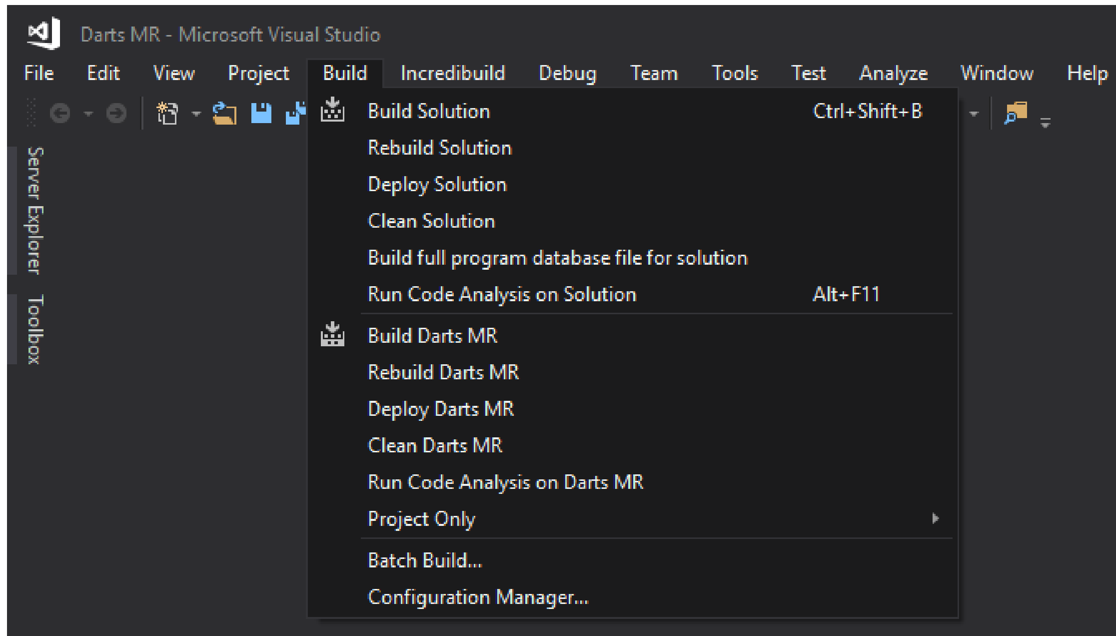

Figure 51. Deploying Darts MR to HoloLens

Once the operation is complete, the application should be visible in the holographic start menu of HoloLens. At this point, the game can be launched similarly to any other holographic application.

6 CONCLUSIONS

The objective of this thesis was to introduce the topics of MR and HoloLens as well as to present an example of design and implementation process of a holographic application created with Unity. The thesis may be considered insightful experience in the regard that it strives to delve into detail of most essential subjects related to each topic, therefore providing an ideal guide for possible alternative approaches.

MR was considered the fundamental source of background information examined throughout the writing process of this thesis. The topic in general was found enormous in size and stretched far onto other fields of industry. Furthermore, similarly to neighbor topics of AR and VR, MR was discovered to possess some history, as related early research performed decades prior was found. The surface of the MR phenomenon has merely been scratched and should present groundbreaking results as further research and development upon the topic is performed.

As a topic, HoloLens was found most interesting. A sophisticated apparatus capable of influencing reality with virtuality, making something that is not perceived real seem most real. The potential of the device was deemed significant. However, what first seemed impeccable through visual representations turned out to be a rather quirky experience in application. The measures of interaction through gestures were found somewhat cumbersome, as trying to handle holograms without full hand articulation proved a limiting challenge.

With natural support for HoloLens and holographic application development, Unity was considered the perfect tool which to utilize in creating and testing a holographic gaming experience. Unity comes packed with quality of life improving features for HoloLens, such as the way to stream directly from the editor to HoloLens wirelessly via networking thus enabling a quick method of testing applications. However, the same feature came with a cost, as it turned out to be rather unstable, causing Unity to crash frequently during the development of Darts MR. Regardless, at the time of writing this thesis, Unity is considered the ideal environment for creating holographic experiences for HoloLens.

This thesis approached various subjects relatively new to the author which proved to be somewhat of a challenge of information overload. Attentively studying each topic resulted in large consumption of time as searching for genuine sources and most up to date information took considerable amount of work and time. However, as a result, a clearer picture of the wholeness was acquired, and could be considered of significant importance to the author. In contrast, the project employed for the implementation chapter suffered slightly, as time became a limiting factor and had to be rerouted for the writing part of this thesis. Some of the initially planned features had to be left either unfinished or cut out from the game completely. However, this left for a perfect opportunity for future development.

7 FURTHER DEVELOPMENT

The implementation process included most of the fundamental features provided by HoloLens. Controlling holograms and game objects in holographic applications via gaze and gestures proved sufficient enough of an experience

overall. However, some example features that could easily enhance the experience could be included in the future versions of the project (such as speech recognition and spatial understanding). Handling UI elements (such as menu items) with speech would surely prove much more efficient when comparing to physical hand gestures. The spatial mapping feature could be improved to include more advanced spatial understanding. This could possibly consist of additional functionality such as a way of providing the information necessary for the holographic application to understand what is considered floor, ceiling, or walls, as well as automating the placement of holograms during the level creation so that the positioning of each hologram would be optimized to include merely the surfaces determined valid. The types of gestures recognized by the application varies depending on each experience, for Darts MR the manipulation gestures were discovered most comfortable. However, additional testing of different recognizing methods could prove better results in the end. Furthermore, the full hand articulation model featured in the recently released successor of HoloLens 2 is of great interest regarding hand gestures and should most definitely provide increased fidelity for hand gestures in future approaches.

REFERENCES

Blender. 2019. About. WWW document. Available at: https://www.blender.org/ [Accessed 24 May 2019].

Bouanani, O. 2015. How to Fund Your Games By Creating and Selling Game Assets. Article. Available at: https://gamedevelopment.tutsplus.com/articles/how-to-fund-your-games-by-creating-and-selling-game-assets--cms-24380 [Accessed 14 March 2019].

Brown, L. 2017. Differences between VR, AR and MR. Article. Available at: https://filmora.wondershare.com/virtual-reality/difference-between-vr-ar-mr.html [Accessed 7 May 2019].

Craig, A-B. 2013. Understanding Augmented Reality : Concepts and Applications. E-book. Elsevier Science & Technology. Available at: https://kaakkuri.finna.fi/ [Accessed 5 February 2019].

Evans, G. & Miller, J. & Iglesias Pena, M. & MacAllister, A. & Winer, E. 2017. Evaluating the Microsoft HoloLens through an augmented reality assembly application. PDF document. Available at: https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1178&context=me_conf [Accessed 20 May 2019].

Fagerholt, E. & Lorentzon, M. 2009. Beyond the HUD - User Interfaces for Increased Player Immersion in FPS Games. Chalmers University of Technology. Department of Computer Science and Engineering. Master of Science Thesis. PDF document. Available at: https://www.semanticscholar.org/paper/Beyond-the-HUD-User-Interfaces-for-Increased-Player-Fagerholt-Lorentzon/16ee02a8839923752c6bc93f294bec67d73a586e [Accessed 9 May 2019].

Goldman, J. 2018. Windows Mixed Reality headsets are here and they're affordable. Article. Available at: https://www.cnet.com/news/windows-mixed-reality-headsets-coming-this-fall-acer-lenovo-dell-hp-asus/ [Accessed 8 May 2019].

Harney, W. 2017. A New Dimension for UI: Using Unity for Virtual Reality. Blog. Available at: https://www.developereconomics.com/unity_virtual_reality [Accessed 13 May 2019].

Kaelin, M. 2017. Microsoft shuts down the Kinect to concentrate on the HoloLens and augmented reality. Article. Available at: https://www.techrepublic.com/article/microsoft-shuts-down-the-kinect-to-concentrate-on-the-hololens-and-augmented-reality/ [Accessed 16 May 2019].

Klint, L. 2018. HoloLens Succinctly. E-book. Syncfusion, Inc. Available at: https://www.syncfusion.com/ [Accessed 21 May 2019].

Kore. 2018. Augmented and Virtual Reality displays. Article. Available at: https://hackernoon.com/displays-for-augmented-and-virtual-reality-2d77b5199a8b [Accessed 8 May 2019].

Lee, S. 2018. The Rise of Augmented Reality: How AR Could Impact the Future. Blog. Available at: https://thisisglance.com/the-rise-of-augmented-reality-how-ar-could-impact-the-future/ [Accessed 1 May 2019].

Microsoft. 2018a. Camera in Unity. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/camera-in-unity [Accessed 27 May 2019].

Microsoft. 2018b. Case study - My first year on the HoloLens design team. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/case-study-my-first-year-on-the-hololens-design-team [Accessed 13 May 2019].

Microsoft. 2019a. Comfort. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/comfort [Accessed 3 June 2019].

Microsoft. 2019b. Gaze. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/gaze [Accessed 21 May 2019].

Microsoft. 2019c. Gestures. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/gestures [Accessed 21 May 2019].

Microsoft. 2019d. Install the tools. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/install-the-tools [Accessed 26 March 2019].

Microsoft. 2018c. Spatial mapping. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-mapping [Accessed 22 May 2019].

Microsoft. 2019e. Voice input. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/voice-input [Accessed 22 May 2019].

Microsoft. 2018d. What is mixed reality?. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality [Accessed 8 May 2019].

Microsoft. 2018e. Head gaze in Unity. WWW document. Available at: https://docs.microsoft.com/en-us/windows/mixed-reality/gaze-in-unity [Accessed 3 June 2019].

Milgram, P. & Kishino, F. 1994. A Taxonomy of Mixed Reality Visual Displays. PDF document. Available at: https://www.researchgate.net/publication/231514051_A_Taxonomy_of_Mixed_Reality_Visual_Displays [Accessed 24 April 2019].

Newnham, J. 2017. Microsoft HoloLens by example: create immersive augmented reality experiences. E-book. Birmingham: Packt Publishing Ltd. Available at: https://kaakkuri.finna.fi/ [Accessed 28 May 2019].

PC Mag. 2019. Encyclopedia. WWW document. Available at: https://www.pcmag.com/encyclopedia/term/38187/augmented-virtuality [Accessed 2 May 2019].

Peacocke, M. & Teather, R. & Carette, J. & MacKenzie, S. & McArthur, V. 2017. An empirical comparison of first-person shooter information displays: HUDs, diegetic displays, and spatial representations. PDF document. Available at: https://www.sciencedirect.com/science/article/pii/S1875952117300435?via%3Dihub [Accessed 9 May 2019].

Reality Technologies. 2019. The Ultimate Guide to Understanding Mixed Reality (MR) Technology. WWW document. Available at: https://www.realitytechnologies.com/mixed-reality/ [Accessed 23 April 2019].

Spacey, J. 2016. Augmented Reality vs Augmented Virtuality. WWW document. Available at: https://simplicable.com/new/augmented-reality-vs-augmented-virtuality/ [Accessed 2 May 2019].

Techopedia. 2019. Mixed Reality. WWW document. Available at: https://www.techopedia.com/definition/32501/mixed-reality [Accessed 25 April 2019].

Thorn, A. 2017. Mastering Unity 2017 Game Development with C# - Second Edition. E-book. Birmingham: Packt Publishing Ltd. Available at: https://kaakkuri.finna.fi/ [Accessed 6 March 2019].

Tuliper, A. 2016. Introduction to the HoloLens. Article. Available at: https://msdn.microsoft.com/en-us/magazine/mt788624.aspx [Accessed 14 May 2019].

Unity. 2019a. Camera. WWW document. Available at: https://docs.unity3d.com/Manual/class-Camera.html [Accessed 27 May 2019].

Unity. 2019b. Colliders. WWW document. Available at: https://docs.unity3d.com/Manual/CollidersOverview.html [Accessed 28 May 2019].

VirtualiTeach. 2017. Exploring the Virtuality Continuum. Article. Available at: https://www.virtualiteach.com/single-post/2017/08/04/Exploring-the-Virtuality-Continuum-and-its-terminology/ [Accessed 2 May 2019].

FIGURES

Figure 1. Differences between VR, AR and MR (Brown 2017) 9
Figure 2. Visual interpretation of Milgram and Kishino's Virtuality Continuum 10
Figure 3. WMR immersive headsets within range of MR (Microsoft 2018d) ... 14
Figure 4. Different types of in-game displays. (a) non-diegetic, (b) diegetic, and (c) spatial (Peacocke et al. 2018) 16
Figure 5. Ideal display range (Microsoft 2019a) 17
Figure 6. Physical appearance of HoloLens (Microsoft 2018d) 18
Figure 7. Screenshot taken by using HoloLens's Mixed Reality Capture, depicting a hologram projected and positioned into a 3D world. 20
Figure 8. Depiction of a primitive cursor, a green tiny circle aligned on top of a cubic object in 3D space, visually indicating gaze. 21
Figure 9. Performing air tap hand gesture (Microsoft 2019c) 22
Figure 10. Performing bloom hand gesture (Klint 2018) 23
Figure 11. Gesture frame (Klint 2018) 24
Figure 12. "See it, say it" label in the top right corner of a holographic application (Microsoft 2019e) 26
Figure 13. Model of a room, generated by spatial mapping (Microsoft 2018c) 27
Figure 14. Real time feed of spatial mapping in progress (Klint 2018) 28
Figure 15. Selecting mandatory Visual Studio components 38
Figure 16. Recommended Unity components 39
Figure 17. Holographic 3D menu of HoloLens emulator 40
Figure 18. Creation of new Unity project 41
Figure 19. Build settings in Unity 42
Figure 20. Inclusion of the support for virtual reality 43
Figure 21. Camera settings for holographic environment 45
Figure 22. UpdateRaycast method of the "GazeManager" game object 46
Figure 23. Defining the bounding volume 47
Figure 24. Update method of the surfaceObserver 48
Figure 25. Removal of surface identity 48
Figure 26. Creation of a new surface game object 49
Figure 27. Creation and population of surfaceData struct 49
Figure 28. RequestMeshAsync method.. 49
Figure 29. Marking an expired surface for removal 50

Figure 30. Visualization of a surface 50
Figure 31. Removal of surfaces within Update method 51
Figure 32. Management of game states 52
Figure 33. MainMenuStateRoutine coroutine 52
Figure 34. ScanningStateRoutine coroutine 53
Figure 35. PlacingStateRoutine coroutine 54
Figure 36. PlayingStateRoutine coroutine 54
Figure 37. SpawnDart method..55
Figure 38. LateUpdate method of the "Cursor" game object 56
Figure 39. Mesh of the dart 56
Figure 40. Definition of manipulation gestures 57
Figure 41. Functionality called by ManipulationStarted action..........................58
Figure 42. Functionality called by ManipulationCompleted action 59
Figure 43. Collision handling of the dart 60
Figure 44. Mesh of the board 60
Figure 45. Definition of tap gestures..61
Figure 46. Functionality called by Tapped action 61
Figure 47. Indication of a valid surface 63
Figure 48. PlayerStats UI game object 64
Figure 49. Passing tracked game data to PlayerStats UI game object...........64
Figure 50. Positioning of the PlayerStats UI game object 65
Figure 51. Deploying Darts MR to HoloLens 66

Table 1. Proposed control scheme 33

APPENDICES Appendix 1/1

Flowchart of the proposed game logic