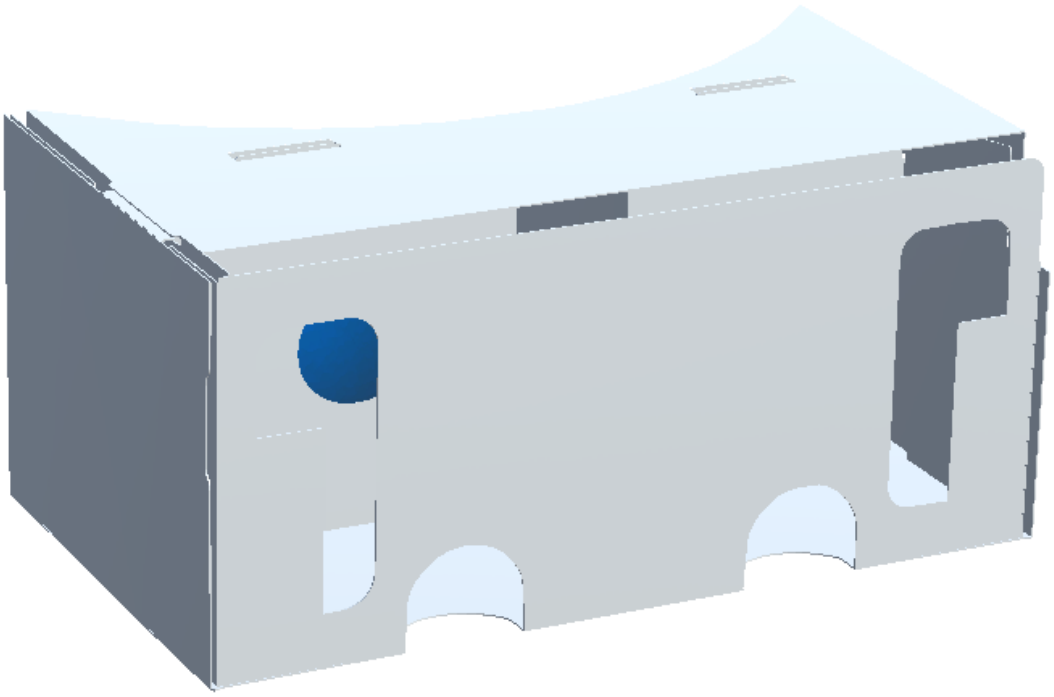
CHALMERS



Monteringsintruktioner i Augmented Reality

En HoloLens-applikation för ihopvikning av kartongglasögon

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

ANTON HURTIG, BETIM RAQI

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2017

Examensarbete 2017:09

Monteringsintruktioner i Augmented Reality

En HoloLens-applikation för ihopvikning av kartongglasögon

ANTON HURTIG, BETIM RAQI

CHALMERS

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg 2017

Monteringsintruktioner i Augmented Reality
En HoloLens-applikation för ihopvikning av kartongglasögon
ANTON HURTIG, BETIM RAQI

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag: En bild på de ihopvikta kartongglasögonen
Institutionen för Data- och Informationsteknik
Göteborg 2017

Monteringsinstruktioner i Augmented Reality
En HoloLens-applikation för ihopvikning av kartongglasögon
ANTON HURTIG, BETIM RAQI
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

De undersökningar som denna rapport redogör för ämnar att förenkla bruksanvisningar med hjälp av Augmented Reality. Med bruksanvisningar menas här de tillhörande instruktionsblad som ofta följer med byggbara försäljningsprodukter. För att åstadkomma detta har en applikation utvecklats, som instruerar användaren hur ett par kartongglasögon ska vikas ihop till den färdiga produkten. Det är AR-plattformen HoloLens, en ledande produkt inom området, som applikationen är utvecklad för. Applikationen visar att AR är en lämplig teknologi för att underlätta instruktionsblad. Med visuella och interaktiva moment kan användaren se hela monteringsprocessen och *samtidigt* bygga ihop den fysiska produkten. Resultatet av undersökningarna kan generaliseras till att omfatta även andra typer av instruktioner.

Nyckelord: Augmented Reality, HoloLens, applikation, bruksanvisning.

Mounting instructions in Augmented Reality
A HoloLens application for folding instructions of cardboard glasses
ANTON HURTIG, BETIM RAQI
Department of Computer science and engineering
Chalmers University of Technology

Abstract

The investigations of this report aim to simplify manual sheets with the aid of Augmented Reality. Manual sheets in this case refer to the instruction papers that often come with commercial products that need some form of assemblying. An AR-application has been developed to achieve this, which instructs the user how to fold some cardboard glasses into the commercial product. The application has been developed for the AR platform HoloLens, a leading product in the field. The application shows that AR is a suitable technology for simplifying manual sheets. With visual and interactive elements the user can see the whole assembly process and simultaneously assamble the physical product. The result of the investigations can be generalised to include other forms of instructions.

Keywords: Augmented Reality, Hololens, application, instruction manual.

Förord

Ett stort tack till de inblandade aktörer i projektet Smarta Fabriker. Speciellt tack till våra handledare från Cybercom, Gustav Svensson (IT-konsult) och Gabriel Ibáñez (innovations-ledare). Vi vill även tacka vår handledare från Chalmers, Sakib Sistek, som håller god anda.

Anton Hurtig, Betim Raqi, Göteborg, Juni 2017

Innehåll

Figurer

1
Inledning

Denna rapport redogör för ett projekt som utförs på företaget Cybercom Group AB i Göteborg. Projektet sträcker sig över 10 veckor, våren 2017.

Cybercom är ett svenskt IT-företag som specialiserar sig på molntjänster och ämnar att hjälpa företag att använda dessa tjänster.[1]

1.1 Bakgrund

Många kommersiella försäljningsprodukter kräver att kunden själv monterar ihop den till dess slutgiltiga form. Ett välkänt exempel är de möbler som företaget IKEA säljer. Till dessa produkter följer vanligtvis någon form av bruksanvisningar, som instruerar monteringsprocessen för kunden. Dessa anvisningar är vanligtvis i pappersform, men eftersom försäljningsprodukterna ofta är av tredimensionell natur så finns det anledning att utforska alternativa lösningar.

En lockande teknologi att tillämpa är Augmented Reality (AR), som låter användaren koppla ihop den "riktiga" världen med en utökad digital värld. Det är en ny teknologi vars arbetsområden utforskas aktivt.[2] Det närmaste som AR kan jämföras med är den besläktade tekniken Virtual Reality (VR), där endast en virtuell värld visas.[4] AR visar upp verkligheten som den är, med *tillägg* av virtuella komponenter. Ett företag som arbetar med AR är Microsoft och det är deras produkt HoloLens som ligger till grund för detta projekt.[3]

HoloLens är en sorts speciella glasögon, med vilka virtuella 3D-hologram kan tillföras till den riktiga världen.[3] Det är t.ex. möjligt att placera ut en virtuell blomma, som bara syns genom glasögonen, på den fysiska marken. Ett annat exempel skulle kunna vara en virtuell hund som kan röra sig fritt och göra sig hörd i HoloLens:ens högtalare.

Cybercom är en partner i ett projekt som kallas för *Smarta fabriker*[1], där det utvecklas fabriker som ska tillverka VR-glasögon i kartong. I dessa glasögon ska användaren kunna placera sin telefon, som med speciell mjukvara visar upp en virtuell 3D-värld. Kartongen som fabriken tillverkar kommer att vikas ihop till VR-glasögon

[1] "Smarta" fabriker är digitaliserade och som utnyttjar internet för att förbättra en rad industriella aspekter. Maskiner är uppkopplade mot ett nätverk vilket skapar möjligheter för nya sätt att interagera med och underhålla en fabrik.

av konsumenten själv, och det är här denna rapportens projekt tar vid.

Med AR skulle användaren kunna se ett hologram av produkten framför sig, som steg för steg byggs ihop till dess slutgiltiga form. Användaren kan *samtidigt* härma instruktionerna på sin fysiska produkt, något som inte skulle vara möjligt med t.ex. VR.

1.2 Problem

Bruksanvisningar för företags produkter är vanligtvis i pappersform, vilket kan begränsa tydligheten av instruktionerna för användaren.

För att lösa problemet ämnas följande punkter att besvaras:

- Är instruktioner som ges m.h.a. AR tydligare än vanliga instruktioner?
- Hur skapar man tydliga instruktioner med AR?
- Kan en användare enkelt följa guiden i HoloLens-applikationen för att vika ihop VR-glasögonen?

1.3 Syfte

Projektets syfte är att visa hur AR kan användas för att förenkla företags bruksanvisningar som följer med diverse försäljningsprodukter. Vissa produkter behöver byggas ihop av kunden själv, och till dessa följer det med anvisningar som instruerar detta. Anvisningarna är vanligtvis illustrerade på papper, men eftersom försäljningsprodukterna ofta är av tredimensionell natur så finns det anledning att undersöka alternativa lösningar. Ett teoretiskt ramverk ska dessutom sammanställas, som ska ligga till grund för arbetet. Detta avser rapportens förundersökning.

1.4 Mål

Målet är att utveckla en demo-applikation för HoloLens, som med AR visar hur ett par VR-glasögon i kartong skall vikas. Applikationen ska steg för steg visa hur användaren ska vika kartongen för att nå den slutgiltiga formen.

1.5 Avgränsningar

Följande punkter kommer inte att behandlas i detta projekt:

- Bruksanvisningar för annat än kartongglasögonen.
- Bildigenkänning som kan analysera användarens fysiska kartongglasögon och på så sätt avgöra vilket steg i vikningsprocessen användaren är på.
- Funktionalitet för handgester, såsom att “dra” i den virtuella modellen för att vika den.
- Utveckling på andra AR-plattformer än HoloLens.

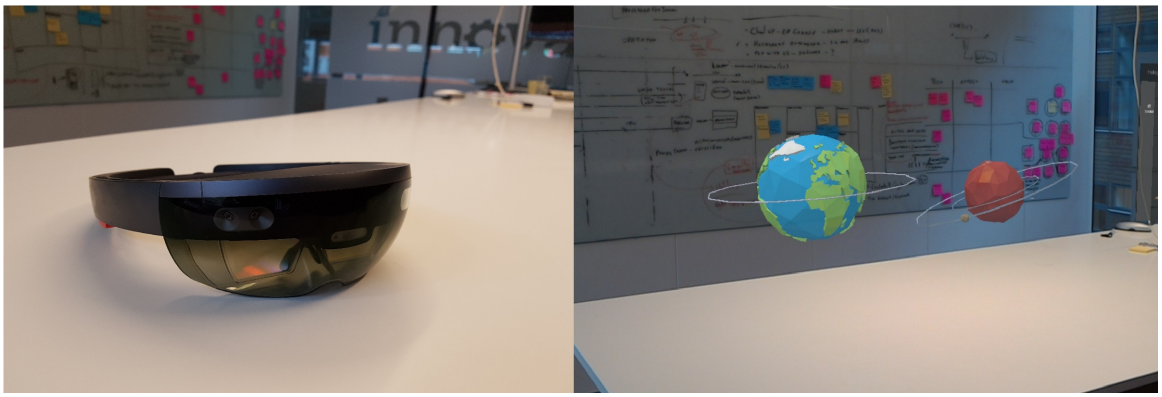
- Teknologin VR
- Hårdvaruutveckling

2
Teknisk bakgrund

Den HoloLens-applikation som denna rapport ämnar redogöra för, kommer framöver att benämnas som FoldableVR. Detta är namnet som programmet tilldelats. Vidare gäller även att med benämningar av ordet *projekt* avses det projekt som denna rapport ämnar redogöra för, om inget annat explicit står uttryckt.

2.1 Hårdvara

2.1.1 HoloLens



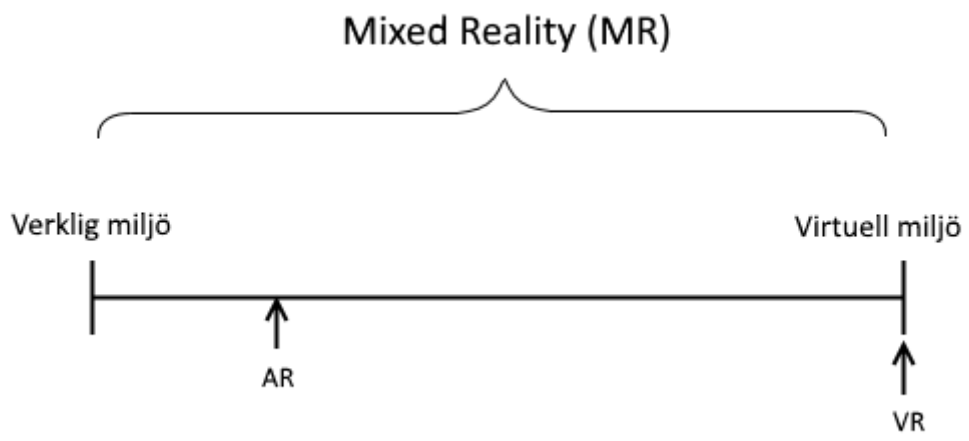
Figur 2.1: Till vänster syns HoloLens och till höger syns HoloLens-applikationen Holograms ur användarens perspektiv

Microsoft HoloLens är ett par AR-glasögon som kan projecera hologram för användaren. Den är utrustad med stereoskopiska kameror som avläser omgivningen och gör det möjligt att interagera med den "riktiga" världen. Glasögonen är mycket stora jämfört med vanliga glasögon, vilket gör det möjligt att få plats med en kraftfull mikrodator i de. Intressanta egenskaper som HoloLens är utrustad med är bl.a.:

- Genomskinliga holografiska linser som möjliggör projicering av hologram
- 4 stereoskopiska kameror
- Inertial measurement unit (IMU) som m.h.a. gyroskåp, accelerometer och andra enheter kan bestämma olika viktiga positionsparametrar.
- En kamera som mäter djupet i den projicerade bilden
- 2MP kamera/HD-videokamera

- Mikrofon
- Högtalare
- Ljussensor
- Wi-Fi
- Bluetooth
- Intel-processor
- Specialbyggd dedikerad processorenhet som bearbetar datan från HoloLens:ens sensorer för att t.ex. kunna tolka handgester.

AR och VR tillhör en mer generell grupp av teknologier vars samlingsnamn är Mixed reality (MR). MR kan förklaras som en skala mellan verklig och virtuell miljö, vilket illustreras i figur 2.2. Ju längre höger ut på skalan desto mindre av verkligheten finns tillgänglig och vice versa. HoloLens som använder AR ligger på den vänstra sidan av spektrummet eftersom man ser verkligheten som den är med *tillägg* av virtuella komponenter.[6]



Figur 2.2: Figuren visar hur Mixed Reality är en skala mellan verklig och virtuell miljö, där HoloLens (AR) ligger i det vänstra spektrat.

När det gäller HoloLens:ens egenskaper i praktiken så finns det ett par viktiga att nämna. Dessa är främst:

- Synfält - HoloLens har ett mycket begränsat synfält, dvs. hur mycket av ögonens synfält den projicerade bilden upptar. Det är svårt att uppskatta men det är i alla fall under 25% av det mänskliga synfältet. För FoldableVR innebär detta att användaren behöver kolla på monteringsinstruktionerna på lagom avstånd, så att allt syns.

- Syndistans - Avståndet mellan användaren och ett virtuellt objekt skall enligt Microsofts rekommendation vara minst 0,85 m. Detta är inställt i FoldableVR:s

inställningar vilket innebär att allt inom radien av 0,85 m klipps bort, dvs. inte syns.[7]

2.2 Mjukvara

2.2.1 Blender

Blender är ett gratis modelleringsprogram från Blender Foundation, som används för att skapa två- och tredimensionella objekt. Programmet är gratis att använda.[8]

I detta projekt så har blender använts för att skapa 3D-modellerna för kartongglasögonen.

2.2.1.1 Bildfiler i Blender

Blender kan inte importera bildfiler, dessa måste först omvandlas till vektorgrafik. Vektorgrafik är ett sätt att representera tvådimensionella bilder på. Istället för att arbeta med bildpunkter som konventionella ritprogram gör, så används istället linjer för att representera bilden. Detta gör att bilderna i vektorgrafik-form kan skalas utan att förlora bildkvalitet. Detta sätt att arbeta med bilder används mycket för logotyper av olika slag, just för att dessa sedan kan skalas utan försämrad kvalitet. Formatet associerat med vektorgrafik är s.k. SVG-filer, ett filformat som Blender stödjer. SVG står för Scalable Vector Graphics.[9]

2.2.2 Unity

Unity är en spelmotor utvecklad av företaget Unity Technologies. I Unity kan 3D-objekt skapas och sedan styras på olika sätt med programkod, som i FoldableVR är skriven i C#. Man kan modifiera objektens egenskaper på flera olika sätt som t.ex. att objekten ska följa fysikens lagar eller inte. Det gör att man kan skapa mer realistiska animationer.[10]

Unity har mycket begränsad funktionalitet vad gäller 3D-modellering, för detta ändamål bör ett dedikerat modelleringsprogram såsom Blender användas. I följande delkapitel kommer grundläggande funktioner i Unity att introduceras. Dessa kunskaper är viktiga för förståelsen av rapportens senare kapitel.

2.2.2.1 Scenen

För att skapa tredimensionella modeller i Unity så behövs någon form av kanvas, där objekt kan placeras i komplexa strukturer för att t.ex. skapa realistiska spel. Denna kanvas kallas i Unity för en s.k. *scen*. I scenen kan *spelobjekt*, en sorts grundläggande byggsten i Unity, tillföras och manipuleras på olika sätt. Scenen är hierarkiskt uppbyggd, spelobjekt kan vara s.k. *barn* till andra spelobjekt.[11] I nästa delkapitel förklaras spelobjekten mer ingående.

2.2.2.2 Spelobjekt

Alla komponenter i scenen, såsom 3D-modeller; menyer och grafiska textfält, är i Unity spelobjekt. Ett spelobjekt kan ha ett eller flera barn, vilka i sin tur kan ha egna barn. Det hela bildar en sorts hierarkisk trädstruktur. Ett spelobjekt kan även vara tomt, dvs. att det inte är associerat med något grafiskt som syns på bildskärmen. Detta är användbart om många spelobjekt behöver någon gemensam egenskap. Då kan det tomma spelobjektet vara förälder till spelobjekten och låta barnen ärva dess egenskaper[12].

2.2.2.3 Collider

För att ett spelobjekt ska kunna interageras med så behöver det en Collider. En collider kan kopplas ihop med ett GameObject för att bestämma hur stor plats den ska ta i spelrummet. För att fånga upp blicken från HoloLens:en så krävs det att ett objekt har en Collider, då det är Collider:n och inte själva objektet som HoloLens:ens kamera ser[13].

2.2.2.4 Coroutines

Det finns ett problem med att exekvera oändlig eller mycket tidskrävande kod i Unity, nämligen att bilduppdateringen fryses. För att förstå problematiken behövs förståelse för hur Unity visar det som syns på bildskärmen[1].

Unity uppdaterar bildrutan flertalet gånger varje sekund. Tiden mellan varje bildruta är mycket liten och kallas i Unity för *deltaTime*. Om ett program har bildhastigheten 20 fps så är deltaTime 1/20 = 0,05 sekunder.[14]

Inför varje bildruta som genereras exekverar Unity en s.k. *Update*-metod som varje spelobjekt tillhandahåller.[15] Vanligtvis är detta inte ett problem eftersom tiden det tar att exekvera metoden är mindre än deltaTime. Men om *Update* innehåller kod som är mycket tidskrävande så hinner inte koden att exekvera klart innan det är dags att visa bildrutan. Resultatet blir i värsta fall att bilduppdateringen fryses.

Lösningen på ovanstående problematik är s.k. *Coroutine*:s. Dessa är metoder[2] i vilka speciella avbrott kan tillkallas under exekvering. Avbrottet återger kontrollen av programmet till Unity, som då kan utföra vitala operationer såsom att uppdatera bildrutan. I nästa bildruta fortsätter sedan metoden att exekvera där den slutade efter avbrottet.[16] För att tillkalla ett avbrott används syntaxen *yield return*.[17] Coroutine:s möjliggör att metoder exekverar över *flera* bildrutor, till skillnad från konventionella metoder som måste exekvera färdigt innan nästa bildruta genereras.

Det finns vissa obligatoriska moment som måste utföras när en Coroutine implementeras.

[1] Med skärm menas här det användaren ser i HoloLens under exekvering av ett Unity-program
[2] Med metod menas här funktioner i programmeringsspråk.

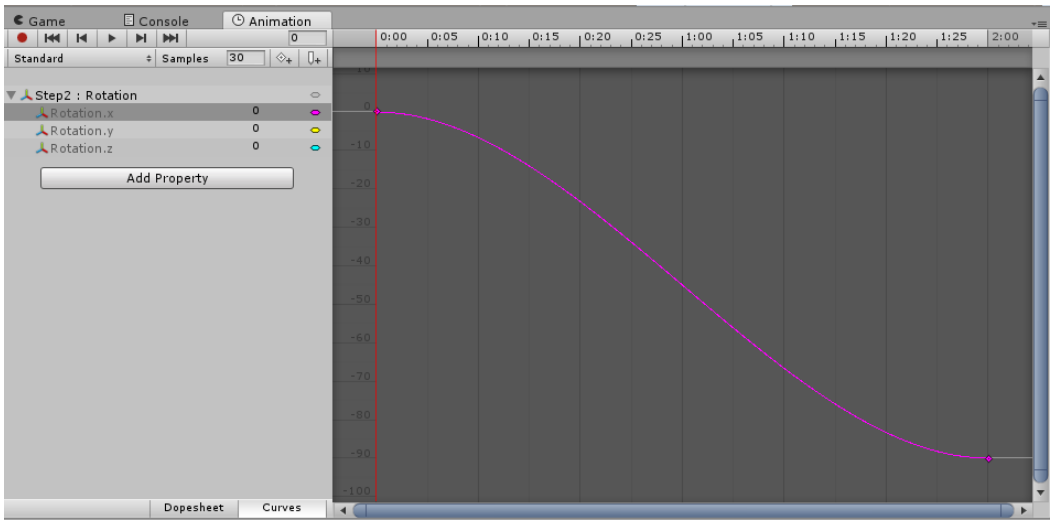
- En Coroutine måste returnera ett s.k. *IEnumerator*-objekt. Vad exakt detta är lämnas att utforskas av den nyfikne läsaren[3]. En fördel med denna konstruktion är att en annan Coroutine kan returneras, eftersom den också returnerar en *IEnumerator*.[18]
- För att tillkalla en Coroutine så måste metoden *StartCoroutine* användas. Om en Coroutine heter *exampleCR* så tillkallas den genom syntaxen *StartCoroutine(exampleCR())*. [18]

2.2.2.5 HoloToolKit

För att programmera för HoloLens så kan en "verktygslåda" vid namn HoloToolkit användas. HoloToolkit är en samling färdiga funktioner som Microsoft utvecklat för att underlätta programmerarens arbete. Hjälpmedlet finns att ladda ner gratis på Github.[19]

2.2.2.6 Animationsobjekt

Ett animationsobjekt är ett objekt som animerar spelobjekt[4]. Med animationsobjektet kan man bestämma vad som ska hända med ett spelobjekt över en viss tid. T.ex. kan man ställa in att ett spelobjekt ska roteras 90° under 5 sekunder. Hur rotationen sker kan styras och automatiseras i Unitys animationsfönster. Där kan man med grafer ställa in de olika objektparametrar, såsom rotation, som ska animeras.[20]



Figur 2.3: Den lila kurvan visar hur ett spelobjekt roterar under loppet av två sekunder.

I figur 2.3 syns detta animationsfönster, där den lila grafen visar hur ett spelobjekt roteras. Den horisontella axeln representerar tid och den vertikala axeln represen-

[3]Läs om IEnumerator i dokumentationen för C#: https://msdn.microsoft.com/en-us/library/system.collections.ienumerator(v=vs.110).aspx

[4]I kapitlet *Teknisk bakgrund* förklarades vad ett spelobjekt är

terar gradantalet. Grafen visar att spelobjektet gör en rotation på 90 grader under loppet av två sekunder.

2.2.2.7 Animationskontroller

Animationskontrollern är ett objekt som hanterar de olika animationsobjekten. Ett spelobjekt kan ha flera olika animationer (i form av animationsobjekt), som behöver sättas samman på något sätt. Detta är animationskontrollerns roll, som kan styra i vilken ordning de olika animationerna ska ske och hur övergångarna mellan dem ska se ut.[21]

Ett viktigt verktyg som animationskontrollern tillhandahåller är s.k. animationsvillkor. Med dessa kan man låta animationer spelas endast om vissa villkor är uppfyllda. Ett sådant villkor är s.k. Triggers, som är en avtryckare som kan starta igång animationer. Triggers går att avtrycka i programkod, vilket gör det möjligt att skapa komplexa animationer.[22]

2.2.2.8 Animator

I Unity kan alla spelobjekt animeras genom att en s.k. *Animator*-komponent fästes till objektet. Animatorn är en sorts övergripande komponent som kopplar samman allt som behövs för att animera i Unity. Ett Animator-objekt behöver en animationskontroller som parameter, Animator-objektet är alltså den högsta länken i kedjan. Animatorn styr animationskontrollern som i sin tur styr animationsobjekten. För att associera en Animator till ett spelobjekt, lägger man till det i spelobjektets komponentlista.[23]

2.2.2.9 Versionshantering med Unity

Det finns flera olika sätt att sammarbeta på vad gäller Unity. Asset Server är den tjänst som Unity Technologies själva tidigare har förespråkat för ändamålet, men numera är tjänsten föråldrad.[24] Unity Technologies slutade underhålla tjänsten från och med version 5.6 av spelmotorn.[25] De har istället lanserat Collaboration, som är en online-tjänst ämnad för sammarbete i Unity. Tjänsten är i skrivande stund i Beta-stadie, så för FoldableVR har istället programmet Git använts.[26] Git är ett s.k. Version Control System (VCS) som kan användas oberoende av vilken utvecklingsmiljö program utvecklas i.

För att kunna använda Git tillsammans med Unity behöver vissa steg utföras:

- Importera en s.k. "Git-ignore"-fil som exkluderar vissa mappar från versionshantering. Eftersom vissa av projektfilerna som skapas i Unity är binärfiler så kan dessa inte versionshanteras.[27]
- I Unity behöver inställningarna "Version Control mode" och "Asset Serialization mode" ställas in. I Unity:s manual finns en guide över hur detta ska göras.[28]

2.2.3 Visual Studio

Visual Studio är en IDE (Integrated Development Environment) från Microsoft[29]. Programmet har stöd för många språk, bland annat C# vilket är det som används för det här projektet.

Programmet kan även användas för att kompilera kod till andra enheter som t.ex. en HoloLens.

3
Metod

3.1 Upplägg

Den metodik som kommer att användas för projeketet är likt Scrum. Det kommer ske dagliga möten på morgonen där aktuella uppgifter gås igenom. Det kommer även ske större möten varannan vecka där den föregående sprinten reflekteras och planering om vad som ska ske i nästa sprint.

De första veckorna av det här projektet kommer läggas på informationsinsamling och att bekanta sig med de nödvändiga verktygen. Mest fokus kommer att läggas på Unity då HoloLens-applikationer utvecklas främst med det programmet.

Huvudarbetet kommer att börja med 3D-modellering. Utifrån en ritning skall delmodeller skapas som tillsammans ska representera hela ritningen. Modelleringen kommer ske med programmet Blender. Dessa modeller behöver till en början inte vara perfekt dimensionerade då det är viktigare att kunna starta med animationerna på objekten än att få de helt perfekta då det skapas tydligare modeller mot slutet. Det är dock viktigt att de övergripande dimensionerna ska stämma så att det inte blir några oklarheter om vilken del som är vilken.

När en första upplaga av modellerna är skapad så ska de överföras till Unity. Där ska animationer skapas som ska representera de olika stegen av instruktionerna. I HoloLens-applikationen ska användaren stegvis kunna "vika" glasögonen. Det ska även vara möjligt att gå tillbaka i stegen. Animationerna ska vara tydliga och enkla att följa.

Med animationerna färdiga så ska projektet överföras till HoloLens. Därefter ska det läggas till funktionalitet för att interagera med hologrammet som har skapats i HoloLens-applikationen, så att instruktionerna med animationer kan kommas åt på ett intuitivt sätt. Konventionen för HoloLens-applikationer är att använda röstkommandon och med en interaktiv meny. Det behöver även skapas en metod för att flytta hologrammet.

Funktionaliteten från metoden som flyttar hologrammet ska vara lätthanterlig och kännas verklig. För att flytta på ett objekt så ska användaren säga ett röstkommando eller trycka på en knapp i menyn. Därefter ska hologrammet följa blicken och anpassa sig efter vilken yta som HoloLens:ens blick stöter på. När objektet ska

placeras så ska användaren återigen använda ett röstkommando eller trycka på en bestämd plats.

Den interaktiva menyn ska vara tydlig med vad de olika knapparna gör. För att få knapparna att bli tydliga så behöver de modelleras med hjälp av Blender. Menyn ska placeras på en position där den är lättåtkomlig men inte i vägen för modellen. Den ska även följa med hologrammet när flyttningsmetoden är aktiv.

Efter att en meny är skapad så ska röstkommandon läggas till. Det är viktigt att kommandona som användaren säger ska vara starkt kopplade till vilken funktionalitet som görs. Det är även viktigt att de olika kommandon som skapas ej är för lika varandra så att de förväxlas av HoloLens:en. När man skapar röstkommandon så är det bra att tänka på att göra kommandona långa, då ju fler stavelser i kommandot, desto mindre risk att HoloLens:en hör fel.

När interaktionsmetoderna har lagts till i applikationen så är det tid att förbättra och göra applikationen tydligare. Menyn ska förbättras genom att tydligt visa vilken del av hologrammet som användaren tittar på och då även förklara vad den delen gör.

4

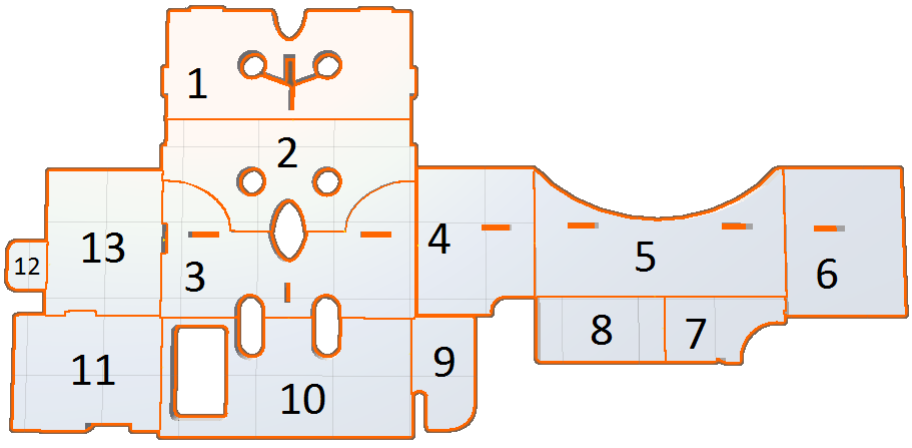
Genomförande

4.1 Modellering

4.1.1 3D-modellering

Det finns många program som kan användas för att skapa 3D-modeller. Eftersom att de modeller som skulle skapas inte var så komplicerade så skedde det enbart en liten undersökning om vilket program som kan användas. Efter diskussion med gruppens handledare så bestämdes det att programmet “Blender” ska användas.

Det första som behövde bestämmas för modelleringen var vilka delar av kartongen som skulle separeras, alltså vilka delar som ska animeras. Ritningen (Appendix A.1) var tydligt uppdelad vilket gjorde beslutet enkelt om vilka delar som skulle skapas. Efter att uppdelningen var gjord så var det dags att börja modellera de olika delarna.



Figur 4.1: 3D-modellen uppdelad i segment

För att få modelleringen att ske på ett logiskt sätt så valdes de segment som ansågs kräva minst ändringar först. Den delen som såg ut att vara enklast var segment 6, vilket innebar att det segmentet modellerades först.

Först skapades det ett rätblock i Blender med de dimensioner som segmentet skulle ha. Det skapades ett rätblock som skulle användas för att skapa det hål som finns i

segment 6 på ritningen. Rätblocket som används för hålet behöver vara dubbelt så tjockt som segmentet är så att det sticker ut på båda sidorna från segmentet. Det här tar bort risken med att hål inte skapas på båda sidorna. Därefter placerades det mindre rätblocket på det större rätblocket i den position där hålet skulle vara och slutligen klipptes ett hål ut.

Nästa del som skapades var segment 5. Hålen skapades på samma sätt som på det tidigare segmentet, och den runda kanten skapades på ett liknande sätt. För att få den runda kanten så skapades en cylinder med ungefär den formen som ritningen hade. Det gick inte att få några exakta mått då ritningen inte visade det. Efter att cylindern var skapad så klipptes formen ut från rätblocket.

Nästa problem som stöttes på skedde när segment 9 skapades då det behövde skapas en rund kant. För att få till den runda kanten så skapades det ett rätblock och en cylinder utöver grundblocket. Cylindern fick formen av hur kanten skulle böjas, och därefter klipptes cylindern ut från det första rätblocket. Det rätblocket placerades på kanten av grundblocket och klipptes därefter ut för att forma en mjuk kant.

Resten av modellen skapades med hjälp av de metoder som lärdes under skapandet av föregående segment. När segmenten var färdiga sattes alla delar ihop för att kunna se hur hela modellen såg ut. Ihopslagningen av alla segment var märkbart annorlunda mot hur ritningen var. Anledningen till att det blev så var på grund av att det saknades dimensioner på ritningen, vilket ledde till att beslut gjordes utifrån vad som kändes passande för segmenten.

Den här lösningen var inte tillräckligt bra då det uppstod uppenbara fel under animeringen. Ett problem var att hålen inte matchade överallt vilket gav ett dåligt intryck av applikationen.

4.1.2 3D-modellering med vektorgrafik

För att komma tillrätta med problematiken i föregående avsnitt behövs nya, mer utförliga 3D-modeller. Att återigen själva modellera skulle vara tidsödande, eftersom modelleringskompetens saknas i projektgruppen.

Ett annorlunda sätt att skapa 3D-modellerna är att med programvara översätta 2D-ritningarna till 3D-modeller. Detta resulterar i mer utförliga modeller eftersom själva modelleringen görs av datorn. Modelleringen blir inte perfekt, en erfaren designingenjör skulle kunna göra ett mycket noggrannare arbete. Datorns modeller är emellertid fullt tillräckliga för att tillfredsställa projektets behov. Glasögonens ritning kan, efter bearbetning, importeras till Blender. Där kan ritningen sedan transformeras till 3D-modeller genom att lägga till ett djup i glasögonens delar.

Det första steget är att bearbeta ritningen på glasögonen. All onödig information på bladet, såsom dimensionssiffror och annan text, behöver raderas eftersom endast glasögonen skall få ett djup. Vidare behöver glasögonen delas upp i dess separata

delar, eftersom de senare ska vikas genom animationer. Det finns två olika sätt att göra detta på, separera delarna innan eller efter importering till Blender. För att välja det andra alternativet, dvs. att dela upp modellen i Blender, behövs erfarenhet av de operationer som sköter detta. Istället delades modellen upp i ett vanligt ritprogram, i detta fall programmet Paint. Det är ett detaljrikt arbete att manuellt dela upp glasögonen i dess komponenter, men där finns mest kontroll över dimensionerna.

Efter uppdelningen av glasögonens, så fylldes de separata delarna med svart färg. När bildfilen konverteras till vektorgrafik så kommer konverteringen att skapa vektorobjekt av de svartfyllda komponenterna.

Att konvertera en bildfil till vektorgrafik kan göras på flera olika sätt, men i detta fall valdes den kostnadsfria tjänsten convertio.co. Convertio är en hemsida som kan konvertera mellan olika kända filformat.[30] Med ett enkelt klick på hemsidan konverteras bilden från PNG- till SVG-fil.

4.2 Animationer

I Unity finns det flera olika sätt att skapa rörelse i 3D-objekt. I början av projektet var animationerna programmerade "för hand", där ett objekt positioneras och roteras gradvis i en programslinga. Detta sätt att animera på är egentligen inte standardsättet i Unity, och längre fram i texten kommer en bättre metod presenteras. Projektgruppen saknade erfarenhet av Unity, vilket ledde till återvändsgränder som denna.

En tidig version av den typ av kod som "för hand" utför animationer finns i Kodavsnitt 5.1.

Kodavsnitt 4.1: Tidig version av programkod som animerar objektet *obj*.

```
1 public GameObject obj;
2
3 while (obj.transform.eulerAngles.z < 90)
4 {
5     obj.transform.RotateAround(new Vector3(0.1f, 0, 0), Vector3.forward
          , 58 * Time.deltaTime);
6
7     yield return null;
8 }
```

Koden visar hur objektet *obj*, som representerar en av glasögonens delar, gradvis roteras 90°. Det finns detaljer i koden som inte är viktiga att diskutera i detta avsnitt, men i grova drag fungerar algoritmen på följande sätt:

1. På rad tre kontrollerar algoritmen om rotationen av objektet *obj* har uppnått den önskade vinkeln på 90°. Om så inte är fallet så fortsätter exekveringen i

while-satsen att gradvis inkrementera objektets rotationsparameter.

2. Rad fem sköter den ovannämnda rotationsinkrementeringen, m.h.a. Unity-funktionen *RotateAround*. Funktionen tar emot tre parametrar i följande ordning: en positionsvektor som rotationen ska ske kring, en riktningsvektor som bestämmer riktningen av rotationen och slutligen en vinkel som bestämmer storleken av rotationen. Detaljer kring varför funktionen kallas, med de i koden angivna parametrarna, lämnas till den nyfikne läsaren.[1]

3. Näst sista raden i koden utför en s.k. *yield return*, som tillfälligt återger Unity kontrollen av programmet. Detta måste ske för att *while*-satsen ska uppfattas som en animation, som *gradvis* ökar vinkeln på *obj*. Om slingan skulle exekvera utan korta avbrott så skulle objektets vinkel öka med 90° på ett ögonblick.

När animationerna var få till antalet var det en acceptabel lösning att använda den här typen av programslinga. Men ju fler objekt som skulle animeras desto mer komplicerad och svåröverskådlig blev koden. Alla parametrar till anropen av *RotateAround* var hårdkodade vilket gav kod som var svår att generalisera.

Det finns ett annat sätt att animera på i Unity, nämligen att låta Unity sköta animeringarna "under huven", och istället animera på en högre nivå. Med s.k. Animations-objekt kan en 3D-modells olika parametrar såsom position, rotation och storlek styras kontinuerligt över tid. För att "vika" den virtuella kartongbiten behövs endast rotations-parametern ändras.[20]

Den virtuella modellen av glasögonen är uppdelad i flera segment, som tidigare presenterades i figur 4.1. Varje segment är en vikbar del på de fysiska VR-glasögonen. För att simulera en vikning i Unity roteras, m.h.a. ett Animations-objekt, den del av glasögonen som ska vikas.

Den ovannämnda animationsmetodiken är standardutförandet i Unity och är det som kommer att användas framöver. Det är värt att notera att kurvan i figur 2.3 inte är linjär, vilket innebär att animationen blir jämn och visuellt mer tilltalande. Detta skulle vara mycket svårare att åstadkomma med den typen av programkod som visades i kodavsnitt 4.1. Koden behövs alltså inte längre, animeringen sker nu på en högre abstraktionsnivå.

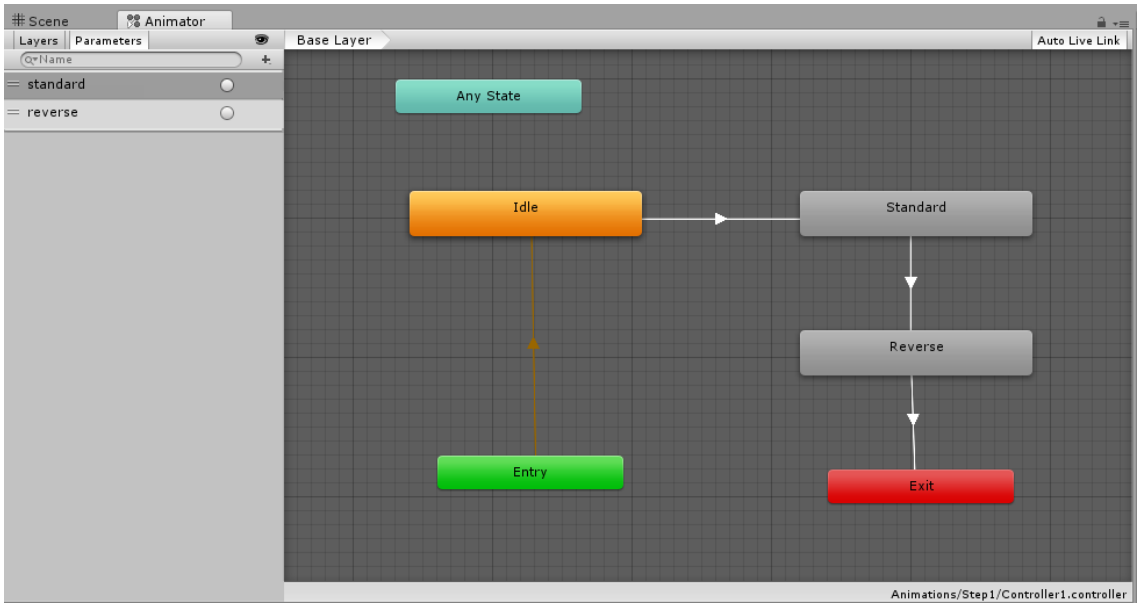
4.2.1 Animationerna i FoldableVR

Som tidigare nämndes i Metod-kapitlet så ska FoldableVR bestå av ett antal steg, där glasögonen stegvis kommer närmare slutgiltig form. I applikationen har varje vikningssteg två olika animationsobjekt, ett som kallas Standard och ett som kallas Reverse. Standard roterar spelobjektet 90° under loppet av två sekunder och Reverse gör ogjort det Standard har utfört. Dessa animationer är alltså varandras "antagonister". Standard är animationen som ska visas när användaren av FoldableVR stegar

[1] I Unity:s API och forum kan *RotateAround* studeras närmare.

framåt i vikningsprocessen medan Reverse går bakåt. Om användaren är på steg 2 i animationerna och sedan stegar sig framåt ett steg, så ska Standard-animationen för steg 3 spelas upp. Om användaren sedan skulle stega sig tillbaka så skulle istället Reverse spelas upp.

Hela processen är koordinerad av animationskontrollerna. I FoldableVR är varje vikningssteg representerat av ett vikningsobjekt. Dessa objekt kallas för Step1, Step2,... Varje vikningssteg har sin egen animationskontroller, d.v.s. varje vikningsobjekt har en kontroller kopplad till dess Animator-komponent.



Figur 4.2: Animationskontrollern för *Step1*-komponenten i *Glasses*

I figur 4.2 syns animationerna Standard och Reverse för *Step1*-objektet i *Glasses*. Animationerna utgör, tillsammans med de andra komponenterna på bilden, en s.k. *tillståndsmaskin*. Denna fungerar som ett sorts flödesschema som beskriver hur animationerna ska spelas. Pilar kan användas för att bestämma vilken ordning animationerna ska spelas i. I figur 4.2 finns förutom tillstånden *Standard* och *Reverse* även *Idle*, *Entry*, *Exit* och *Any State*. De tre sistnämnda är tillstånd som Unity själv har skapat. *Entry* och *Exit* definierar start- respektive slutpunkter för animationerna. När animationerna har nått tillståndet *Exit* så sker en övergång till *Entry* igen omedelbart och bildar på så sätt en oändlig slinga.[21]

Tillståndet *Idle* är till för att hindra Standard att starta direkt efter Entry. Detta görs för att användaren av FoldableVR själv bestämmer när Standard-animationerna ska aktiveras, genom att stega vidare i programmet. För att åstadkomma detta använder animationskontrollern avtryckare. I figur 4.2 syns avtryckarna i den vänstra spalten med namnen *standard* och *reverse*. Dessa ska inte förväxlas med animationsobjekten som har samma namn fast med stor första bokstav. När avtryckaren *standard* aktiveras så startar den motsvarande animationen Standard. Samma gäller för *reverse* och Reverse.

Hela animationsprocessen kan demonstreras med en numrerad lista, där varje nummer i listan representerar ett tillstånd i animationskontrollern:

1. **Entry** - Här börjar animationskontrollern animationsslingan. Det enda som sker i detta tillstånd är en omedelbar övergång till Idle.
2. **Idle** - Fortfarande har inga animationer visats i FoldableVR. Idle väntar nu på att avtryckaren *standard* ska aktiveras. Detta gör användaren genom att klicka på nästa-knappen i FoldableVR och först då övergår Idle till Standard.
3. **Standard** - Nu ser användaren en visuell animation, det är Standard-animationen som spelas. Animationen i fråga är en vikning av en del i glasögonen. När animationen är klar så väntar den på att avtryckaren *reverse* ska aktiveras. Innan denna har aktiverats så händer alltså ingenting.
4. **Reverse** - När avtryckaren *reverse* aktiverats så spelas animationen Reverse, som är den omvända animationen till Standard. När animationen spelat klart så övergår tillståndet Reverse till Exit.
5. **Exit** - Animationerna är klara. Exit övergår omedelbart till Entry och sluter därmed animationsslingan.

Det är värt att notera några saker med ovanstående slinga.

- Varje steg i anvisningarna för VR-glasögonen har en egen animationskontroller. Varje animationskontroller ser till utseendet likadana ut, men de behövs eftersom de refererar till *olika* animationer. *Step1* exempelvis har en tillhörande animation som heter Standard. Det är den som syns i figur 4.2 i form av ett tillstånd i tillståndsmaskinen. *Step2* har också en animation som heter Standard och detsamma gäller för alla steg i glasögonen. Standard-animationerna för Step1, Step2, Step3... är dock helt olika animationer, unika för just det steg de är associerade med. Därför behövs en egen animationskontroller för varje steg, som kontrollerar just det stegets animationer.

- Slingan pågår hela tiden fram tills att FoldableVR avslutas. Eftersom varje steg har sin egen kontroller så är totalt 12 sådana slingor under samtida exekvering.

4.3 Programlogik

4.3.1 Animationer

Det finns i huvudsak tre olika metoder som ansvarar för animationerna i applikationen. Dessa är följande:

- *StartAnimation*
- *CompleteInstructions*
- *ReverseInstructions*

Endast implementationen för *StartAnimation* kommer att visas. De andra metoderna kommer att förklaras ytligt och lämnas att utforskas för den nyfikne läsaren.

Kodavsnitt 4.2: Metoden som startar animationerna genom att avtrycka en *Trigger*.

```
1  private void StartAnimation(GameObject instructionStep, string
       animation)
2  {
3      Animator anim = instructionStep.GetComponent<Animator>();
4      anim.SetTrigger(Animator.StringToHash(animation));
5      if (animation.Equals(STANDARD))
6      {
7          step++;
8      }
9      else if (animation.Equals(REVERSE))
10     {
11         step--;
12     }
13 }
```

Metoden tar emot ett spelobjekt som representerar det steg som ska animeras. Detta motsvarar i *VR-glasses* objekten Step1, Step2, Step3... Utöver detta tar metoden emot en textsträng, *animation*, som bestämmer vilken animation som ska spelas. De möjliga animationerna är Standard- och Reverse-animationerna.

Efter att koden kommit åt Animator:n för spelobjektet, så avtrycker den Triggern associerad med animatorn. Det är denna Trigger som sätter igång själva "vikningen" av glasögonens delar. Efter rad fyra så uppdateras variabeln *step*, som är programmets sätt att hålla reda på vilket steg i vikningarna användaren är på.

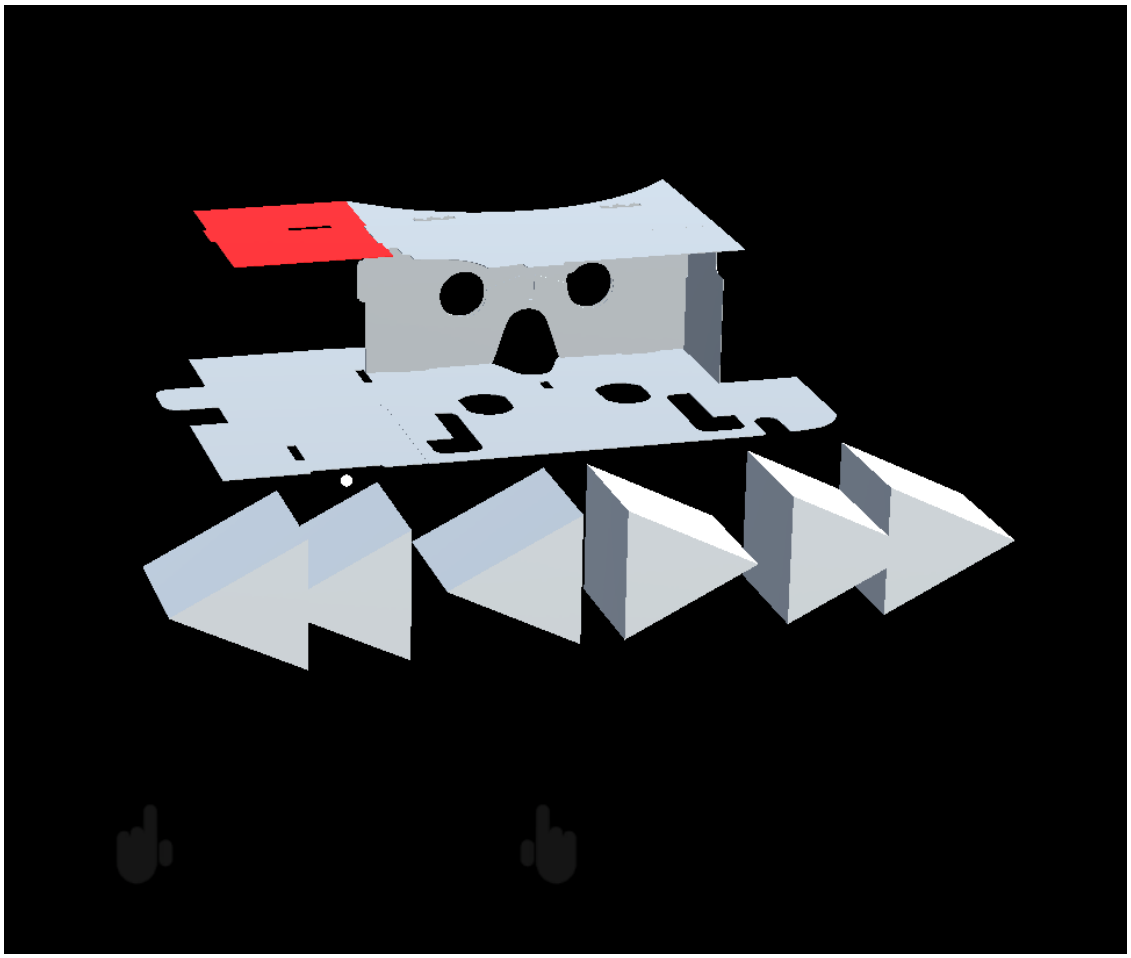
Den ovanstående metoden är den kodmässiga grunden för animationerna i *FoldableVR* och är det rekommenderade sättet att animera på i Unity. Metoden används i CompleteInstructions och ReverseInstructions, vilka presenterades i listan ovan.

CompleteInstructions "viker" ihop glasögonen till slutgiltig form. Glasögonens delar viks ihop i kronologisk ordning, med korta avbrott emellan vikningarna. Metoden använder StartAnimation i en slinga för att åstadkomma detta.

På liknande sätt gör ReverseInstructions för att nollställaglasögonen, dvs. vika alla delar till den ursprungliga formen. Funktionen är viktig eftersom den tillåter användaren att börja om från början.

4.3.2 Metoden Highlight

Highlight är metoden som ansvarar för att visuellt markera den komponent på glasögonen som härnäst ska vikas. Detta gör den genom att tillfälligt byta färg på komponenten i en blinkande stil.
Metoden ser ut på följande sätt.



Figur 4.3: Markering av ett objekt

Kodavsnitt 4.3: Koden markerar de delar av glasögonen som ska vikas härnäst.

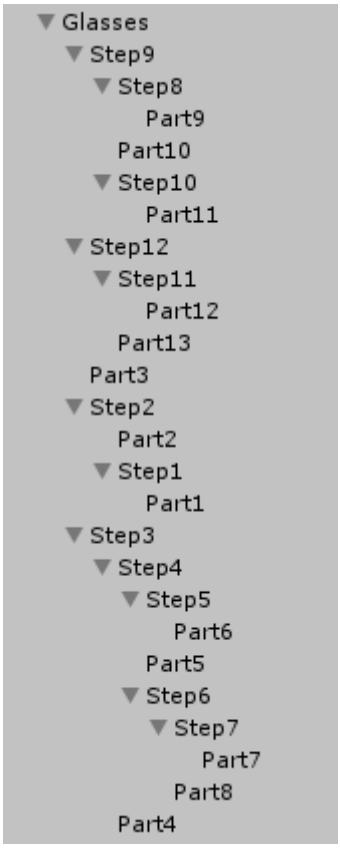
```
1  IEnumerator Highlight(GameObject obj)
2  {
3      Color originalColor = obj.GetComponent<Renderer>().material.color;
4
5      yield return new WaitForSeconds(0.4f);
6      changeColor(obj, Color.red);
7
8      yield return new WaitForSeconds(1);
9
10     changeColor(obj, originalColor);
11
12     yield return new WaitForSeconds(0.4f);
13
14     changeColor(obj, Color.red);
15
16     yield return new WaitForSeconds(0.4f);
17
18     changeColor(obj, originalColor);
19
20     yield return new WaitForSeconds(0.3f);
21 }
```

Metoden tar som parameter emot en av glasögonens sidor, vars ursprungsfärg sparas temporärt på rad tre. Notera att metoden är en Coroutine och därmed returnerar en *IEnumerator*. För att åstadkomma en blinkande effekt, byter metoden färg på *obj* med korta avbrott mellan skiftningarna. *WaitForSeconds* är metoden som avbryter fortsatt exekvering under den, som parameter, angivna tiden.[31]

Det finns även en *överlagrad* version av metoden *Highlight* som markerar *flera objekt samtidigt*.

4.4 Trädstruktur i Unity's spelobjekt

I Unity är VR-glasögonen uppdelade i många mindre delar, där varje del representeras av ett spelobjekt. Uppdelningen är gjord utefter den uppdelning som gjordes i kapitlet om Modellering. I figur 4.1 syns hur glasögonen är uppbyggda.



Figur 4.4: Glasögonen i en trädstruktur

"VR-glasses" är förälder till alla andra spelobjekt som bygger upp glasögonen. Barnen till VR-glasses är uppdelade på ett sätt som gör det lätt att programmera logiken för animationerna. Varje animationssteg är i hierarkin ett spelobjekt, vars barn är de spelobjekt som det steget animerar. Alla objekt i hierarkin som består av ordet "Part" tillsammans med en siffra, representerar delen av glasögonen med

motsvarande siffra i figur 4.4. För att exemplifiera: Step1 har endast Part1 som barn eftersom detta är det enda spelobjekt som animeras i steg ett. I Step2 ingår Part2 som barn, men även Step1 (och därmed Part1) eftersom bägge dessa ska animeras i steg två. Detta fortsätter sedan och utgör trädstrukturen.

4.4.1 FindChildren-algoritm

I FoldableVR finns som tidigare nämnts en metod som heter Highlight, och denna ansvarar för att rödmarkera vikningsinstruktionerna. Highlight tar emot en lista av objekt som ska markeras, och hur denna lista skapas är upp till anroparen av metoden. Det finns mer eller mindre bra sätt att generera denna lista på. Ett naivt sätt skulle vara att statiskt referera till objekten i trädstrukturen Glasses.

Kodavsnitt 4.4: Ett naivt sätt att generera listan som Highlight behöver. Här refererar varje listelement till en specifik del av Glasses.

```
1 List list = new List();
2
3 list.Add(Part1);
4 list.Add(Part2);
5 list.Add(part3);
6
7 Highlight(list);
```

Koden är inte generell, om fler objekt skulle läggas till i Glasses så skulle listan *list* behöva utökas med referenser till dessa objekt. Dessutom så skulle en sådan lista behöva skapas varje gång Highlight anropas, vilket orsakar oönskad kodduplicering.

Lösningen blottas i Unitys hierarkiska struktur på sina spelobjekt. Dessa är som tidigare nämnts uppbyggda i trädstruktur, varför rekursion är ett verktyg att ta i beaktning. Genom att rekursivt iterera genom trädets element skulle dessa element tillsammans kunna utgöra en lista som sedan anges som parameter till Highlight.

Ett problem är dock att Highlight-metoden endast tar emot element som ska markeras. I Glasses finns även föräldrar till dessa element. Föräldrarna är de element som där "Step" ingår i elementnamnet, t.ex. Step1. Dessa objekt finns till för att lättare dela upp glasögonen i en logisk ordning, men också för att de används som ett stöd i den fysiska placeringen av barnobjekten i Unitys scen. För att förklara lösningen till problemet behövs en definition.

Definition. *Ett löv i en trädstruktur är en gren i trädet som saknar barn.*

En viktig observation för lösningen är att alla objekt som ska markeras är *löv* i Glasses. Lösningen består av en rekursiv metod som endast itererar genom trädets löv-objekt.

Kodavsnitt 4.5: Istället för att referera till listobjekten statiskt så kan en rekursiv metod returnera listan.

```
1     /// <summary>
2     /// Finds all leaf children of the given parent in a recursive
3     /// manner. A child is considered a leaf if it has no children of
4     /// its own.
5     /// </summary>
6     /// <param name="parent">The parent whose children are to be found.
7     /// </param>
8     /// <returns>A list containing the children objects.</returns>
9
10    private List<GameObject> FindChildren(GameObject parent)
11    {
12        int nrOfChildren = parent.transform.childCount;
13        List<GameObject> list = new List<GameObject>();
14
15        if (nrOfChildren == 0)
16        {
17            list.Add(parent.transform.gameObject);
18            return list;
19        } else
20        {
21            for (int i = 0; i < nrOfChildren; i++)
22            {
23                List<GameObject> localList = FindChildren(parent.
                      transform.GetChild(i).gameObject);
24
25                foreach (GameObject gameObject in localList)
26                {
27                    list.Add(gameObject);
28                }
29            }
30            return list;
31        }
32    }
```

Metoden FindChildren är rekursiv eftersom den anropar sig själv på rad 23. Som inparameter tar funktionen emot det spelobjekt vars *löv* den ska hitta, och som utparameter returnerar den en lista på löven. Algoritmen fungerar på följande sätt:

1. Metoden börjar med att ta reda på hur många barn parent har. Den skapar även en tom lista som sedan ska fyllas med *löven*.
2. I if-satsen kontrollerar den sedan om det aktuella *parent*-objektet är ett löv. Om det är det så lägger den till lövet i listan och returnerar den.
3. Om *parent* inte är ett löv så anropas FindChildren på parents barn, och deras löv läggs till i listan som sedan returneras.

Nu kan Highlight anropas på följande sätt:

Kodavsnitt 4.6: Highlight anropas med FindChildren som inparameter.

```
1     public GameObject step3;
```

```
2
3       ...
4
5       Highlight(FindChildren(step3));
```

step3 är en publik[2] variabel som refererar till *Step3*-objektet i Glasses. Anropet kan tolkas som "Markera de objekt som är associerade med steg 3 av vikningen".

4.5 HoloLens-applikation

Den här delen av projektet började med informationsinsamling. Microsoft har grundläggande guider om hur HoloLens-utveckling fungerar vilket gjorde det enkelt att börja[32]. För att få mer bred kunskap så samlades information även in från hemsidor som stackoverflow[3]. Under det här stadiet så skapades ett antal testapplikationer för att få lite känsla om hur verktygen fungerade.

4.5.1 Skapa 3D-hologram för HoloLens

När grundläggande information hade samlats in kunde arbetet med att överföra FoldableVR till HoloLens börja. Det första steget som behövde göras var att bygga projektet för den platform som ska användas, i detta fall HoloLens, och överföra det från Unity till Visual Studio. Det är viktigt att man använder korrekta inställningar när man bygger projektet för HoloLens:en. Det här ställdes in genom att på Build Settings i Unity så väljer man "Universal 10" som SDK, "D3D" som UWP Build Type samt att se till att "Unity C# Projects" är itryckt och slutligen att lägga till den scen där projektet finns. När projektet var byggt så sparar man den nya applikationen i en mapp som i projektets fall har döpts till App och i den mappen öppnar man filen med filslutet .sln i Visual Studio.

I Visual Studio så behöver ett par inställningar ställas in innan applikationen kan överföras till en HoloLens. Debug ska ändras till Release, ARM ska ändras till x86 och slutligen skall Simulator ändras till Remote Machine. När de inställningarna är aktiva så är det dags att koppla ihop datorn med HoloLens:en. Det görs genom att IP-adressen för HoloLens:en skrivs in på den pop-up rutan som dök upp när Remote Machine trycktes. IP-adressen finns under inställningar på HoloLens:en. Om korrekt IP-adress har skrivits in ska enheterna vara ihopkopplade.

När alla inställningar var skapade för att överföra Unity-projektet till HoloLens:en så var det dags att skapa en applikation. Den här applikationen blev först inte som planerat då den visade sig som en 2D-tavla med 3D-objektet liggande där i, likt hur det såg ut i Unity. För att fixa det problemet så krävdes det enbart att en parameter var inställd och den hittas under Edit -> Project Settings -> Player -> Other Settings. Där ska Virtual Reality Supported vara ibockad samt att Windows

[2]Variabeln är publik eftersom man i Unity enkelt kan referera till objektet *Step3* i *Glasses*.
[3]Stackoverflow är en hemsida där utvecklare kan ställa frågor och diskutera lösningar

Holographic ska finnas under Virtual Reality SDKs. Projektet byggdes igen och den här gången skapades det ett 3D-hologram som placerades framför användaren.

4.5.2 Förflyttning av objekt

Nu när det gick att skapa 3D-hologram så var det dags att starta arbetet med att modifiera hologrammet. Den första funktionen som prioriterades var att kunna förflytta modellen på ett intuitivt sätt. Tanken var att användaren ska klicka på kartongmodellen för att greppa tag i den och därefter så flyttas den dit HoloLens:ens blick är. När objektet ska placeras så klickar användaren på det. För att det ska bli tydligare för användaren om vart objektet kan placeras så ska ett verktyg som heter "Spatial Mapping" aktiveras. "Spatial Mapping" används för att visa vilka ytor som HoloLens:en ser[33].

Implementationen av den här flyttningsmetoden var problematisk eftersom programmet inte fångade upp användarens klickningar. Det var möjligt att flytta på objektet om det ställdes in att användaren skulle hålla objektet vid start, men då gick det inte att placera objektet. Ett annat problem som uppstod var att "Spatial Mapping" inte aktiverades under förflyttningen. Det visade sig att problemet var lätt att lösa, då det enbart var en parameter som behövde ställas in under Edit -> Project Settings -> Player -> Publishing Settings -> Capabilities -> SpatialPerception.

För att kunna hantera klickningar från användaren så behövdes ett objekt från HoloToolKit som heter InputManager, vars syfte är att ta hand om all input från användaren. Det behövde även ställas in ett par inställningar från HoloToolKit som man hittar under HoloToolKit -> Configure -> Apply Project Settings så att Unity kan användas som en simulator av HoloLens:en.

När problemen med "Spatial mapping" och klickningarna var lösta så var det dags att skapa objekt som kan fånga upp klickningarna och hantera dem. Det skapades ett skript kallat ClickHandler som ärvde gränssnittet IPointerClickHandler, skriptet var ansvarig för alla klickningar på kartongen. ClickHandler kopplades till ObjectCollection som låg i scenen på Unity[34].

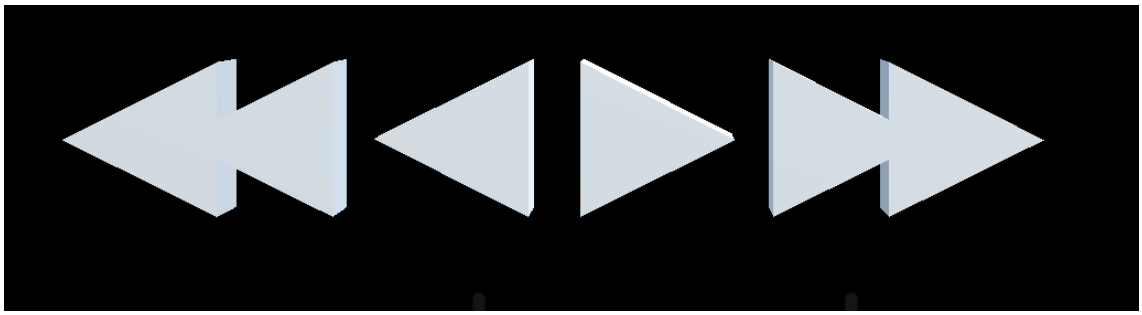
Metoden "move" som styr förflyttningen av objektet får inte vara för kompliceras då det behöver vara exekverbart på varenda frame då den kallas i objektets updatemetod. Metoden flyttar objektet 10% närmare HoloLens:ens blick varje gång den kallas, alltså en gång per frame. Det innebär att objektet hänger med blicken mjukt och inte bara följer blicken helt. En nackdel med att enbart flytta objektet 10% varje frame är att objektet aldrig kommer hamna precis där användaren vill, men det är så extremt små mariginaler. För att räkna ut hur nära objektet kommer vara används $0,9^n$ där n är antal bildrutor. Efter 1 sekund kommer objektet vara $0,9^{60} = 0,0018 \approx 0,2\%$ från där användaren kollar vilket är obetydligt för det mänskliga ögat. Koden för move metoden finns på Appendix B.

Efter att grundlägganda funktionalitet för att fånga användarens inmatning hade

implementerats, så testades applikationen igen. Det märktes snabbt att det inte var tillräckligt med att användaren behövde klicka på objektet för att placera det. Efter en del informationssökande så bestämdes det att objektet skulle placeras oavsett vart man klickade om det aktivt förflyttades. För att få det att fungera så behövde det skapas ett nytt objekt i scenen vars syfte var att fånga upp klick som inte fångades upp av någon annan klass. Om objektet förflyttades när klickningen fångades, så skulle objektet placeras där det befinner sig.

4.5.3 Menyknappar

Det bestämdes att det skulle skapas ett par knappar vars syfte var att styra spelflödet. De funktionerna som skulle läggas till i första iterationen var: nästa steg, föregående steg, starta om och en genomgång av alla animationer. Knapparna behöver vara lätta att förstå utan text vilket är varför de modellerats som pilar.



Figur 4.5: Från vänster till höger: Reset, Gå tillbaks ett steg, Gå fram ett steg, Spela upp ett demo.

Den första knappen som skapades var den som skulle ansvara för nästa steg. Först skapades ett GameObject i Unity av modellen. En Collider med en rektangulär form som kallas Box Collider skapades så att det kunde bli möjligt att klicka på objektet. Collidern fick en rektangulär form även fast modellen var än pil för att göra det enklare för användaren att trycka på objektet. Det var viktigt att se till att Collider:n inte blev för stor, så att det inte skulle bli fel knapp som man tryckte på. När modellen var importerad till Unity var det dags att skapa ett skript som fångar upp klickningarna och hanterar att rätt knapptryck leder till rätt funktion. När knappen som ansvarade för nästa steg var färdig, så var det bara att göra likadant för de andra funktionerna. Slutligen så skapades ett förälderobjekt för hela menyn där skriptet placerades som döptes till ControllerObjects.

4.5.4 Ihopslagning av meny och modell

För att menyn ska följa med när man flyttar på modellen av kartongen så behöver de slås samman till ett gemensamt objekt. Genom att låta ControllerObjects vara barn till ObjectCollection så flyttas ControllerObject automatiskt när Glasses flyttas. Nu

när menyn var kopplad till modellen så var det dags att bestämma sig för vart menyn ska placeras i relation till modellen. Första positionen som testades var ovanför modellen till vänster men det visade sig vara en dålig position då under vissa delar av animationen så blockerade modellen menyns vy. Nästa position som testades var att menyn skulle vara ännu högre upp, men det märktes tidigt att det här inte var bra då synfältet på HoloLens:en var för litet och att det var onaturligt att behöva kolla så långt upp för att se nästa animation. Den slutgiltliga positionen för menyn var rakt under modellen, det var nära till hands och andra HoloLens-applikationer har ofta menyn under modeller också.

Flyttningsmetoden behövde justeras ännu en gång då det uppstod problem efter att menyn hade placerats under modellen. Metoden tog inte till hänsyn vart menyn var utan den brydde sig enbart om vart modellen var. När hela objektet skulle placeras på marken så hamnade menyknapparna under marken. Det här problemet löstes med att man höjde objektets nya y-position med 0.1m i de fallen när användaren ville placera objektet på marken. Ett problem uppstod från det här då menyknapparna kunde hamna i vägen när objektet skulle placeras. Instruktionerna som är kopplade till respektive menyknapp utfördes istället för att placera objektet. För att åtgärda det här problemet behövde ClickController få behörighet att veta om objektet förflyttades. Om en menyknapp trycktes under tiden som objektet flyttades så skulle det ignoreras, och objektet skulle istället placeras.

4.5.5 Röstkommandon

Röstkommandon är ett smidigt sätt att interagera med en HoloLens-applikation. För FoldableVR valdes det att lägga till röstkommandon som hade samma funktionalitet som menyknapparna.

- Next step - Nästa steg av instruktionen
- Previous step - Gå tillbaks ett steg av instruktionen
- Play demo - Återgå till startposition och spela därefter upp instruktionerna i en slinga
- Reset - Gå tillbaks till första instruktionen
- Move here - Placera modellen framför användaren och starta flyttningsmetoden

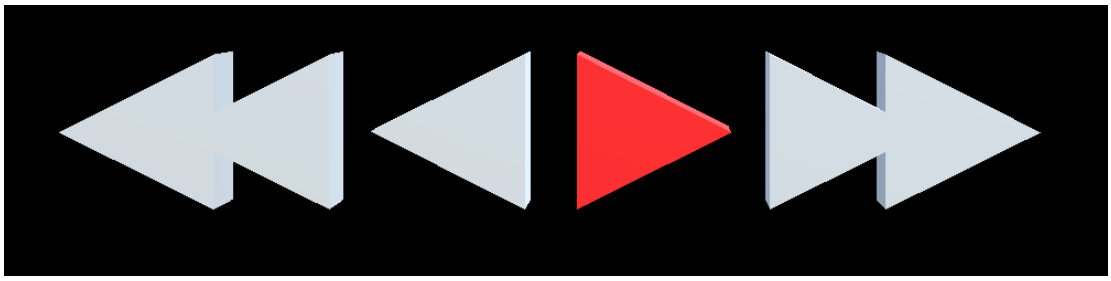
Skapandet av röstkommandon för HoloLens görs med hjälp av en klass från HoloToolKit som heter KeywordManager. Det enda som utvecklaren behöver göra är att koppla ihop ett nyckelord med en metod. En inställning behövde också ställas in i Unity för att meddela programmet att den ska använda mikrofonen. Den inställningen hittas genom att gå till Edit-> Project Settings -> Player -> Publishing Settings -> Capabilities och därefter se till att Microphone är aktiv.

Applikationen testades av flera personer för att undersöka vad som kunde förbättras. Det visade sig snabbt att applikationen kändes otydlig då instruktionerna inte var tillräckligt intuitiva. Det var svårt att veta när ett specifikt föremål var markerat och det var inte tydligt vad de olika knapparna styrde. Det var även svårt att flytta modellen. Efter diskussion om hur de här problemen kunde lösas så bestämdes det

att en markeringsmetod skulle läggas till.

4.5.6 Markering

Markeringsmetoden ska ändra färg på objektet som är i fokus samt visa upp en förklarande text, när objektet inte längre är markerat så ska det återgå till hur det såg ut ursprungligen. Processen för att skapa en markeringsmetod delades upp i flera mindre delar. Första delen som implementerades var att byta färg på ett objekt genom att enbart kolla på det. För att märka när ett objekt är fokuserat så behöver det kopplas ett skript med metoden onMouseEnter från MonoBehaviour till objektet[36][35]. Monobehaviour är den basklass som alla Unity-skript ärver ifrån. Metoden onMouseEnter kallas varje gång användaren tittar på objektet där skriptet är kopplat till. Koden för att byta färg på ett objekt var redan skriven i GameController så det var bara att implementera den här också. Logiken för färgläggandet av objektet lades till i metoden onMouseEnter. Skriptet placerades på alla objekt som skulle kunna bli markerade. Efter att det gick att ändra färg på ett objekt genom att titta på det så var det dags att få tillbaka grundfärgen när användaren slutade titta på objektet. För att få tillbaka färgen så sparades grundfärgen i en privat variabel som sedan användes i metoden onMouseExit för återställning[37].



Figur 4.6: Menyn när en knapp är fokuserad.

När den grundläggande logiken för metoden var skapad testades metoden på de olika objekten som skulle kunna markeras. Metoden behövde modifieras så att den ändrade färgen på objektet där skriptet låg, samt alla barnobjekt. Den behöver markera barnobjekten också eftersom att ett objekt kan vara en samling av andra objekt. För att hitta alla underliggande objekt så användes metoden findParents som skapades i GameController och därefter placerades alla GameObjects i en lista. När objektet ska ändra färg så ska skriptet gå igenom hela listan.

Den nya metoden testades och den fungerade bra på menyobjekten, men det fungerade inte på Glasses. Felsökning visade att det måste finnas en Collider hos objektet för att skriptet ska fungera korrekt. Om objektet inte har en Collider så kommer blicken aldrig fångas upp hos objektet. Det räckte inte att barnen till objektet hade en Collider. Att samla ihop Colliders från sina barn fungerade inte heller, så problemet fick lösas på annat sätt. Skriptet modifierades så att det kunde placeras på ett barnobjekt istället. Problemet med Collider:s löstes men på grund av strukturen på Glasses så var den här ändringen inte tillräcklig. Eftersom att strukturen på Glasses var så unik så fanns det inget smidigt sätt att lösa det här problemet så det skapades

ett undantagsfall. Skriptet modifierades så att om man angav ett GameObject i en publik variabel markerades det objektet med barn istället.

När markeringen fungerade för alla objekt som krävdes så var det dags att finslipa metoden så att den är lättare att implementera på nya objekt. Det första som ändrades var att en publik variabel lades till vars ansvar är att bestämma vilken färg som objektet skulle få när det iaktogs. Nästa modifiering som gjordes var att förhindra markering under tiden som objektet förflyttas. Det här problemet löstes genom att ClickHandler importerades från ObjectCollection och därifrån kallades metoden getIsMoving för att ta reda på om objektet rör sig eller ej. Därefter modifierades metoden så att markeringen inte aktiveras om getIsMoving är sann.

Slutligen så skulle programmet testas på HoloLens:en, men då visade det sig att användarens blick inte fångades upp. Problemet var att onMouseEnter och onMouseExit ej fungerade för HoloLens:en då pekaren för HoloLens:en inte räknas som en mus. För att lösa problemet så behövde metoden omstruktureras så att den använde sig av två Interface för HoloLens istället, IPointerEnterHandler och IPointerExitHandler[38][39]. OnPointerEnter ersätter onMouseEnter och OnPointerExit ersätter onMouseExit.

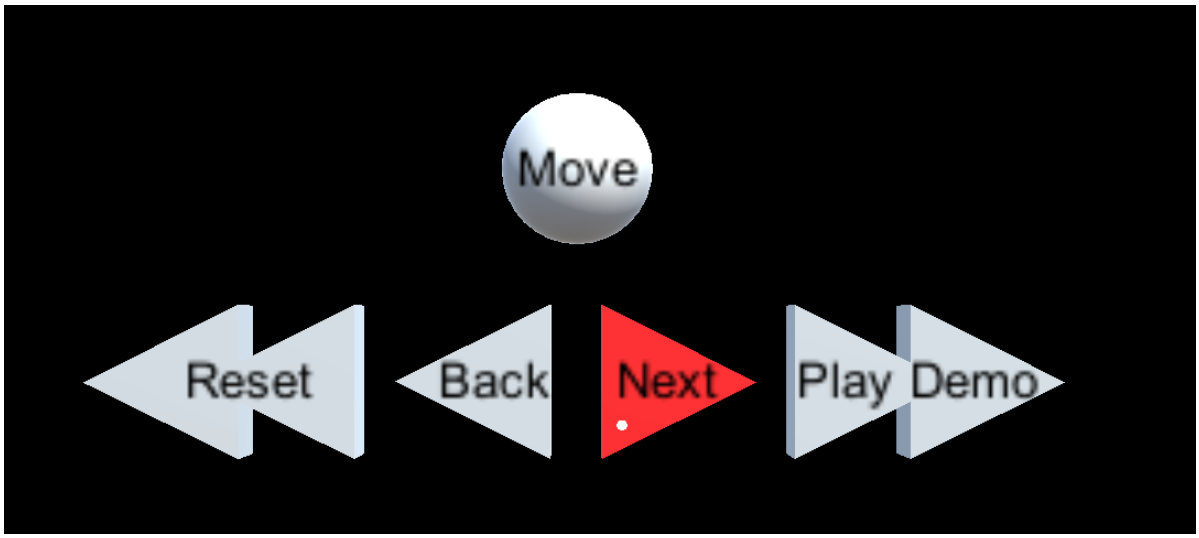
5
Resultat

En AR-applikation har utvecklats som instruerar användaren hur en kartong ska vikas ihop till ett par VR-glasögon. Med interaktiva moment, såsom handgester och röstkommandon, kan användaren stegvis se hela vikningsprocessen. Applikationen är konstruerad för AR-plattformen HoloLens.

Användaren kan navigera i vikningsinstruktionerna på två olika sätt. Det första sättet är att klicka på någon av de olika menyknapparna:

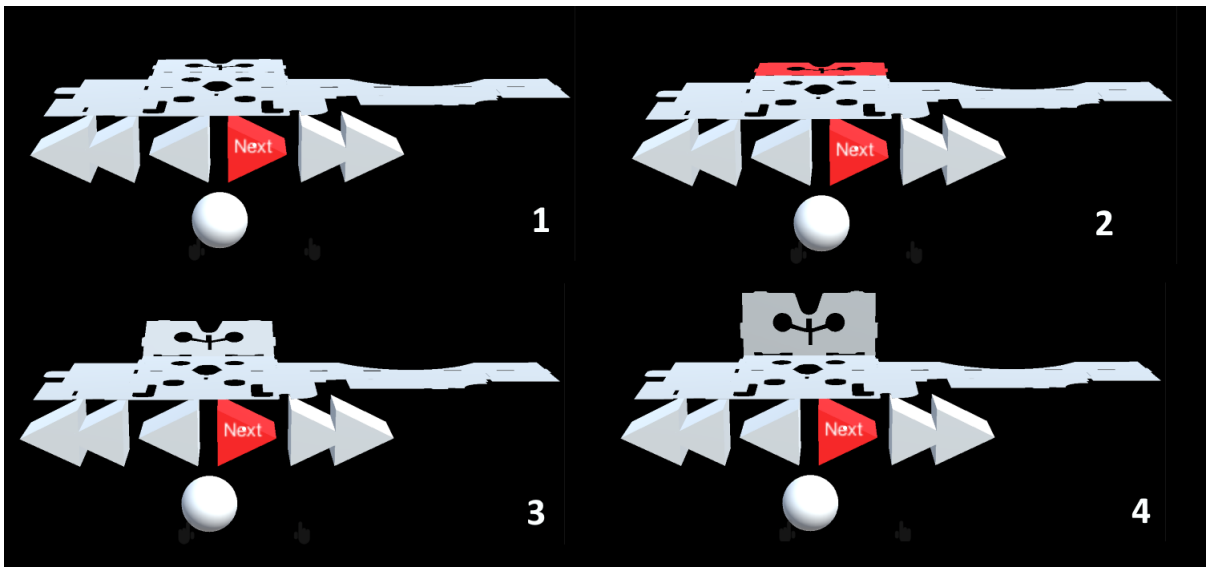
- **Next** - Nästa steg i instruktionerna
- **Back** - Gå tillbaka ett steg i instruktionerna
- **Play demo** - Återgå till startposition och spela därefter upp instruktionerna i en oavbruten slinga
- **Reset** - Gå tillbaka till första instruktionen
- **Move** - Flytta modellen av glasögonen till önskad position.

För att göra det tydligare för användaren så visas en text som förklarar vad knappen gör när användaren fäster sin blick på den.



Figur 5.1: Bilden visar menyn med en knapp markerad. Observera att, i applikationen, visas texten enbart när användaren fäster sin blick på menyknappen.

Det andra sättet som användaren kan navigera på är genom röstkommandon. Varje menyknapp har ett motsvarande röstkommando som utför knappens handling.



Figur 5.2: Bilden visar vad som sker när det första steget av instruktionen utförs

Applikationen markerar visuellt, i en blinkande stil, den del som härnäst ska vikas. Vid oklarheter kan användaren gå tillbaka till föregående steg för att se instruktionen igen. Programmet har även möjlighet att visa upp hela vikningsprocessen i en omgång.

6
Slutsats

Med hjälp av AR och den utvecklade applikationen är det full möjligt att förenkla företags bruksanvisningar. Tester som har gjorts av testpersoner med FoldableVR visar att vikningsinstruktionerna i applikationen är tillräckligt enkla att följa.

Applikationens funktionalitet har medvetet hållits minimal, allt för att användaren ska kunna fokusera på det viktiga, i detta fall att vika den fysiska produkten. För att ta tillvara på de naturliga sätten att interagera med HoloLens, har både röstkommandon och handgester använts.

Tyvärr så har tiden inte räckt till för att göra mer omfattande studier kring hur instruktionerna upplevs för en bredare användargrupp. Detta kommer att diskuteras vidare i avsnitt 6.1.

6.1 Tydliga instruktioner med AR

AR är ett verktyg som vi anser förtydligar monteringsinstruktioner. Med verkligheten som bakgrund, kan användaren jämföra den fysiska enheten med ett hologram. Objektet kan betraktas från vilken vinkel som helst, genom att användaren rör sig runt det. Interaktiva moment, såsom röststyrning, gör också användaren mer involverad i instruktionerna.

Därav anser vi att AR har potential att bli ett komplement till de traditionella bruksanvisningarna i pappersform. Detta är helt baserat på våran erfarenhet av AR, och ska inte ses som någon form av fakta. Tyvärr har tiden inte räckt till mer omfattande studier som skulle behövas för att kunna dra några objektiva slutsatser. Genom att låta ett större urval av användare jämföra hur de upplever instruktionerna i FoldableVR, gentemot traditionella, skulle det kunna framgå vad som är det tydligaste sättet att presentera instruktioner på. Det skulle även behöva undersökas på ett mer akademiskt plan vad exakt som gör instruktioner tydliga, vilket är utanför detta projektets ramar.

Däremot tycker vi inte att våra egna slutsatser ska förkastas. De testpersoner som har fått prova applikationen har alla varit eniga om att instruktionerna är rättframma och tydliga. En video på våra instruktioner har använts i en mobilapplikation som utvecklats i projektet Smarta fabriker, och därav har alltså instruktionerna även använts i professionellt syfte. Denna video valdes istället för klassiska ritningsinstruk-

tioner, vilket vi tycker är en stark indikation på att traditionella bruksanvisningar har sina begränsningar. Även andra inblandade parter, såsom våra handledare har gett en positiv bedömning av FoldableVR.

Det finns flera olika undersökningar som vi skulle vilja utföra för att ta reda på om instruktionerna är tillräckligt tydliga. En sådan skulle kunna vara att samla in information från användare när de utför de olika vikningsstegen. Detta skulle kunna användas för att se om användarna kollade på samma instruktion flera gånger.

Att skapa tydliga instruktioner i AR innebär enligt oss att fullt ut utnyttja de egenskaper som teknologin besitter. En utav egenskaperna som HoloLens är utrustad med är de stereoskopiska kameror som krävs för att skanna av verkligheten. Detta innebär för FoldableVR att användaren kan placera den virtuella modellen av glasögonen var som helst i verkligheten, t.ex. på ett bord. Detta gör upplevelsen mer verklighetstrogen.

En annan egenskap som HoloLens besitter är möjligheten att projicera hologram för användaren. Med dessa hologram kan en virtuell modell av glasögonen visas, som med animationer ”viker” glasögonen. Den del av modellen som ska vikas härnäst, blinkar i ett rött ljus, allt för att göra det tydligt vilken del av glasögonen användaren ska arbeta med.

Två olika sätt att använda applikationen på har implementerats, med röststyrning eller en klickbar meny. HoloLens har stöd för röstkommandon, och det visade sig vara förvånansvärt enkelt att implementera dessa.

6.2 Vad som kunde gjorts annorlunda

För att lättare hantera animationerna i programkoden har en speciell trädstruktur använts, som i Unity representerar glasögonens delar. Nu i efterhand kan strukturen ifrågasättas eftersom den, vid vissa omständigheter, kräver omstrukturering när fler steg i animationerna läggs till. Den är alltså svår att bygga vidare på och dessutom kan det vara svårt att förstå strukturen eftersom den kan tyckas vara komplex. Ett alternativ skulle kunna vara att undvika en trädstruktur och istället låta glasögonens delar vara fristående objekt. Med fristående objekt avses här objekt som saknar en *förälder-barn*-relation. Denna relation gör det möjligt att animera flera objekt på samma sätt med en animation. Att undvika sådana relationer skulle alltså innebära ett behov av fler animationer, och exakt hur dessa skulle implementeras vet vi inte. Det är hursomhelst något som skulle behöva undersökas vidare.

Att programmera glasögonens animationer för hand visade sig vara en onödigt invecklad lösning. Istället användes Unity:s verktyg för animationer, vilket underlättade arbetet. Anledningen till att dessa verktyg inte användes redan från början var att vi saknade erfarenhet av Unity och därmed inte visste att sådan funktionalitet existerade. Det tog tid innan vi insåg att koden för animationerna inte gick att utveckla vidare. Om projektet hade utförts idag så hade vi handlat annorlunda, med

större fokus på att förstå hur Unity och HoloLens-applikationer fungerar.

En annan intressant fråga att beakta är varför vi först valde att modellera glasögonen själva, istället för att omvandla ritningen till en 3D-modell med den metod som presenterades i avsnitt 4.1.2. Svaret på detta är helt enkelt att den senare metoden inte var bekant för oss vid tidpunkten då modellerna skulle skapas. Det är viktigt att förstå att vi aldrig tidigare berört 3D-modellering i våran utbildning, och därmed inte visste så mycket om tillgängliga alternativ. Det var först efter att en bekant berättat för oss hur de hade löst ett liknande problem som vi blev medvetna om alternativet. Däremot ångrar vi inte att vi angrep problemet på två olika sätt, det har givit oss grundläggande kunskap om modelleringteknik, vilket vi är tacksamma för.

6.3 Vidareutveckling

Vidareutveckling av projektet kan handla om att skapa en bättre meny. Den menyn skulle kunna ta inspiration från en inbyggd applikation i HoloLens som heter Holograms. Där är alla 3D-objekt inramade med interaktiva ramar som man greppa tag i för att flytta objekten. Rotering av objekten är möjligt genom att först greppa tag i ett klot som sitter på ramarna, och därefter flytta handen i den riktning som objektet ska roteras. Det finns även en möjlighet att ändra storleken på objektet, vilket skulle kunna vara av intresse om en användare vill se modellen av glasögonen närmare.

Det skulle även vara intressant att skapa instruktioner för andra byggsatser än kartongglasögon.

En instruktionsbok har fördelen att flera personer kan läsa den samtidigt, medan den applikation som vi har utvecklat enbart kan visas för en person samtidigt. Detta gör det svårt för samarbete, om flera personer skulle vilja hjälpas åt att montera en produkt. I FoldableVR så är det här inte ett problem, eftersom att alla steg i instruktionerna är kopplade till varandra. Om användaren utför ett steg, så bygger nästa steg på att föregående steg är utfört osv. För att möjliggöra för två användare att se samma virtuella modell framför sig, och därmed kunna sammarbeta i vikningsinstruktionerna, skulle HoloLens:ens nätverkegenskaper behöva utnyttjas. Detta ligger utanför projektets omfång, men är intressant i utvecklingsperspektiv.

6.4 Hållbar utveckling

En viktig aspekt att beakta är huruvida den tekniska innovationen främjar hållbar utveckling. Om virtuella bruksanvisningar skulle bli vanligare så skulle kanske pappersanvändningen att minska. Detta har dock inte bara med AR att göra, digitala kopior av befintliga bruksanvisningar skulle ha samma effekt. Men vi anser att det är ett steg i rätt riktning.

Litteraturförteckning

[1] Cybercom Group AB, "Cybercom Group", 2017. [Online]. Tillgänglig: http://www.cybercom.com/sv/. Hämtad: 3 apr, 2017.

[2] W. L. Hosch, "Augmented Reality", i *Encyclopædia Britannica*, 2017. [Online]. Tillgänglig: https://www.britannica.com/technology/augmented-reality. Hämtad: 3 apr, 2017.

[3] Microsoft, "Microsoft HoloLens", 2017. [Online]. Tillgänglig: https://www.microsoft.com/en-us/hololens. Hämtad 3 apr, 2017.

[4] H. E. Lowood, "Virtual Reality", i *Encyclopædia Britannica*. [Online]. Tillgänglig: https://www.britannica.com/technology/virtual-reality. Hämtad: 3 apr, 2017.

[5] Microsoft, "HoloLens hardware details", 2017. [Online]. Tillgänglig: https://developer.microsoft.com/en-us/windows/mixed-reality/hololens_hardware_details. Hämtad 13 apr, 2017.

[6] Microsoft, "Mixed Reality", 2017. [Online]. Tillgänglig: https://developer.microsoft.com/en-us/windows/mixed-reality/mixed_reality. Hämtad 13 apr, 2017.

[7] Microsoft, "Hologram Stability", 2017. [Online]. Tillgänglig: https://developer.microsoft.com/en-us/windows/mixed-reality/hologram_stability. Hämtad 13 apr, 2017.

[8] Blender Foundation, "Blender", 2017. [Online]. Tillgänglig: https://www.blender.org/about/. Hämtad 17 apr, 2017.

[9] Encyclopædia Britannica, "Virtual Reality", i *Encyclopædia Britannica*. [Online]. Tillgänglig: https://www.britannica.com/topic/vector-graphics Hämtad: 26 apr, 2017.

[10] Unity Technologies, "Unity - Game engine, tools", 2017. [Online]. Tillgänglig: https://www.blender.org/about/. Hämtad 17 apr, 2017.

[11] Unity Technologies, "Unity - Manual: Scenes", 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/CreatingScenes.html. Hämtad 17 apr, 2017.

[12] Unity Technologies, "Unity - Manual: GameObject", 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/class-GameObject.html. Hämtad 17 apr, 2017.

[13] Unity Technologies, "Unity - Manual: Colliders", 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/CollidersOverview.html. Hämtad 6 juli, 2017.

[14] Unity Technologies, "Unity - Scripting API: Time.deltaTime", 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/Time-deltaTime.html. Hämtad 17 apr, 2017.

[15] Unity Technologies, “Unity - Scripting API: MonoBehaviour.Update()”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html. Hämtad 17 apr, 2017.

[16] Unity Technologies, “Unity - Manual: Coroutines”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/Coroutines.html. Hämtad 17 apr, 2017.

[17] Unity Technologies, “Unity - Manual: Coroutines”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/351/Documentation/ScriptReference/index.Coroutines_26_Yield.ht
Hämtad 17 apr, 2017.

[18] Unity Technologies, “Unity - Scripting API: MonoBehaviour.StartCoroutine()”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html. Hämtad 17 apr, 2017.

[19] Github, Inc., “Microsoft/HoloToolkit - Unity”, 2017. [Online]. Tillgänglig: https://github.com/Microsoft/HoloToolkit-Unity. Hämtad 17 apr, 2017.

[20] Unity Technologies, “Unity - Manual: Animation Clip”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/class-AnimationClip.html. Hämtad 26 apr, 2017.

[21] Unity Technologies, “Unity - Manual: Animator Controller”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/class-AnimatorController.html. Hämtad 26 apr, 2017.

[22] Unity Technologies, “Unity - Manual: Animation Parameters”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/AnimationParameters.html. Hämtad 26 apr, 2017.

[23] Unity Technologies, “Unity - Manual: Animator Controller”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/class-Animator.html. Hämtad 26 apr, 2017.

[24] Unity Technologies, “Unity - Manual: Asset Server”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/AssetServer.html. Hämtad 17 apr, 2017.

[25] Unity Technologies, “Deprecating Asset Server support”, Unity Blogs, 2017. [Online]. Tillgänglig: https://blogs.unity3d.com/2017/02/06/deprecating-asset-server-support/. Hämtad 17 apr, 2017.

[26] Unity Technologies, “Unity Services - Collaborate”, 2017. [Online]. Tillgänglig: https://unity3d.com/services/collaborate. Hämtad 17 apr, 2017.

[27] Github, Inc., “Unity.gitignore”, 2017. [Online]. Tillgänglig: https://github.com/github/gitignore/blob/master/Unity.gitignore. Hämtad 17 apr, 2017.

[28] Unity Technologies, “Unity - Manual: Version control integration”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/Manual/Versioncontrolintegration.html. Hämtad 17 apr, 2017.

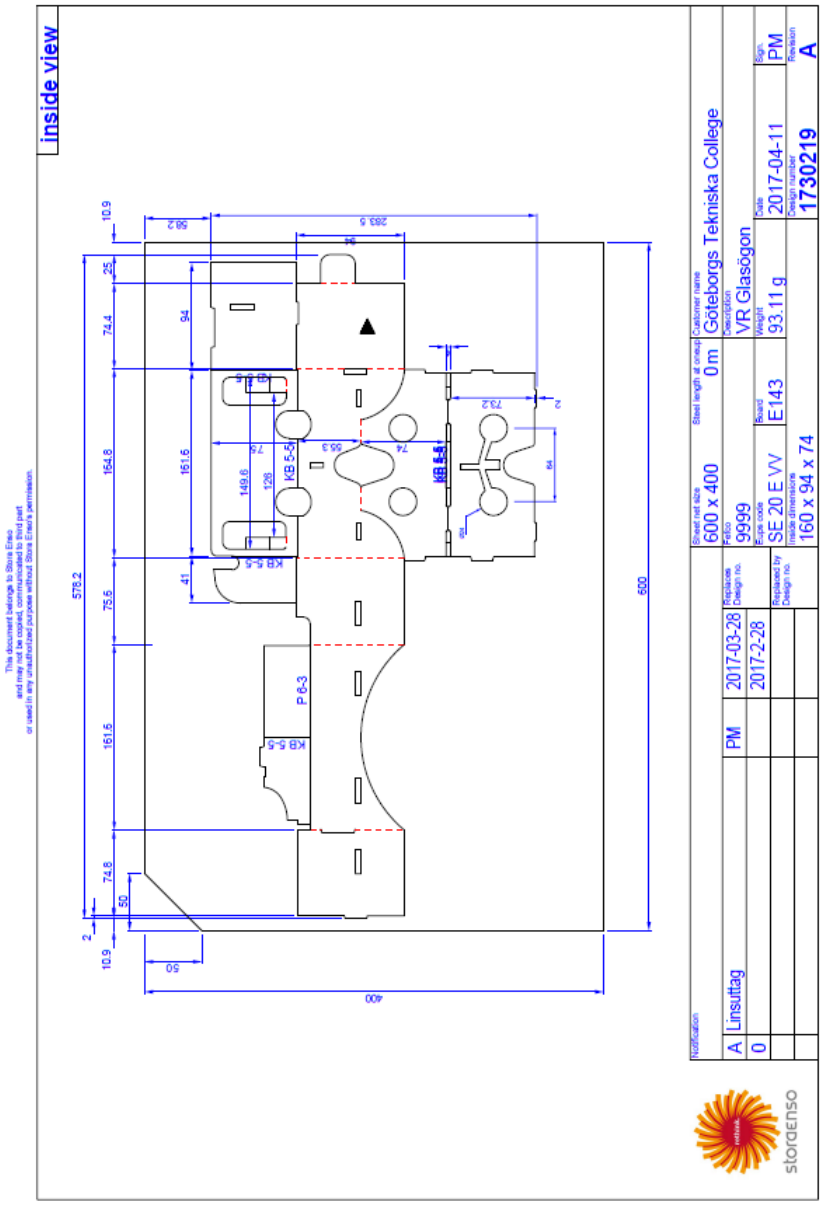
[29] Microsoft, “Microsoft - Visual Studio IDE”, 2017. [Online]. Tillgänglig: https://www.visualstudio.com/vs/ide/. Hämtad 16 maj, 2017.

[30] Convertio, “Convertio – File Converter”, 2017. [Online]. Tillgänglig: https://convertio.co/. Hämtad 26 apr, 2017.

[31] Unity Technologies, “Unity - Scripting API: WaitForSeconds”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/WaitForSeconds.html. Hämtad 26 apr, 2017.
[32] Microsoft, “Microsoft - Mixed Reality Academy”, 2017. [Online]. Tillgänglig: https://developer.microsoft.com/en-us/windows/mixed-reality/academy. Hämtad 17 maj, 2017.
[33] Microsoft, “Microsoft - Spatial mapping”, 2017. [Online]. Tillgänglig: https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping. Hämtad 26 apr, 2017
[34] Unity Technologies, “Unity - Scripting API: IPointerClickHandler”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/EventSystems.IPointerClickHandler.html. Hämtad 6 juli, 2017.
[35] Unity Technologies, “Unity - Scripting API: MonoBehaviour”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html. Hämtad 6 juli, 2017.
[36] Unity Technologies, “Unity - Scripting API: OnMouseEnter”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseEnter.html. Hämtad 6 juli, 2017.
[37] Unity Technologies, “Unity - Scripting API: OnMouseExit”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseExit.html. Hämtad 6 juli, 2017.
[38] Unity Technologies, “Unity - Scripting API: IPointerEnterHandler”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/EventSystems.IPointerEnterHandler.html. Hämtad 6 juli, 2017.
[39] Unity Technologies, “Unity - Scripting API: IPointerExitHandler”, 2017. [Online]. Tillgänglig: https://docs.unity3d.com/ScriptReference/EventSystems.IPointerExitHandler.html. Hämtad 6 juli, 2017.

A

Appendix A: Ritning



Figur A.1: Ritning av VR-glasögonen.

B

Appendix B: Flyttningsmetod

Kodavsnitt B.1: Metoden för att flytta på objekt

```
1     /// <summary>
2     /// Moves the object to where the user is looking. If the user
          gazes at a wall, it will be placed by the wall. If the gaze
          hits the floor
3     /// the object will be placed 0.1m above the floor. If the user
          gazes at a roof, it will be placed under the roof. If the gaze
          does not
4     /// hit anything the object will be placed 2 meters in front of the
           user.
5     /// The object will rotate so that it always faces the user. The x
          and z angle is locked, only the y-angle is modifiable.
6     /// </summary>
7     private void move()
8     {
9         //Debug.Log("Move");
10        //find position gazing at
11        var headPosition = Camera.main.transform.position;
12        var gazeDirection = Camera.main.transform.forward;
13        Vector3 position;
14        Quaternion orientation;
15        RaycastHit hitInfo;
16        var layermask = 1 << 31; // 31: Physics layer
17        //If the gaze hits anything within 3.5 meters, see what surface
               it hit
18        if (Physics.Raycast(headPosition, gazeDirection, out hitInfo,
              3.5f, layermask))
19        {
20            position = hitInfo.point;
21            orientation = Quaternion.LookRotation(-hitInfo.normal);
22            Debug.Log(position);
23            float wallInclinationThreshold = 10.0F;
24
25            Vector3 normalVerticalProjection = new Vector3(0.0F,
                  hitInfo.normal.y, 0.0F);
26            Vector3 normalHorizontalProjection = new Vector3(hitInfo.
                  normal.x, 0.0F, hitInfo.normal.z);
27
28            bool isVertical = normalHorizontalProjection.magnitude /
                  normalVerticalProjection.magnitude >
                  wallInclinationThreshold;
29            if (isVertical)
30            {
```

```
31                  //Debug.Log("Wall");
32                  orientation = Quaternion.LookRotation(-hitInfo.normal,
                        Camera.main.transform.forward);
33              }
34              else
35              {
36                  bool isHorizontal = normalVerticalProjection.magnitude
                        / normalHorizontalProjection.magnitude >
                        wallInclinationThreshold;
37                  if (isHorizontal)
38                  {
39                      //If the gaze hits the floor, place the object 0.1m
                            over the floor
40                      if (hitInfo.normal.y > 0)
41                      {
42                          //Debug.Log("Floor");
43                          orientation = Quaternion.LookRotation(-hitInfo.
                                normal, Camera.main.transform.forward);
44                          position.y += 0.1f;
45                      }
46                      //If the gaze hits the roof, place the object under
                            the roof
47                      else
48                      {
49                          //Debug.Log("Roof");
50                          orientation = Quaternion.LookRotation(-hitInfo.
                                normal, -Camera.main.transform.forward);
51                      }
52                  }
53              }
54          }
55          //If the gaze does not hit anything, place the object 2 meters
                in front of the user
56          else
57          {
58              //Debug.Log("Floating object");
59              position = headPosition + 2f * gazeDirection; //2m in front
60              orientation = Camera.main.transform.localRotation; //
                    facing user
61          }
62          orientation.x = 0;
63          orientation.z = 0;
64          if (moveParent)
65          {
66              transform.parent.position = Vector3.Lerp(transform.parent.
                    position, position, 0.1f);
67              transform.parent.rotation = Quaternion.Lerp(transform.
                    parent.rotation, orientation, 0.1f);
68          }
69          else
70          {
71              transform.position = Vector3.Lerp(transform.position,
                    position, 0.1f);
72              transform.rotation = Quaternion.Lerp(transform.rotation,
                    orientation, 0.1f);
73          }
```

```
74
75      }
```