

# DomeDagen

Immersiv och skalbar medieutveckling för en domskärm

**Frans Johansson  
Josefine Klintberg  
Iris Kotsinas  
Erik Larsson  
David Robín Karlsson  
Algöt Sandahl**

Examinator: Daniel Jönsson

# Sammanfattning

Denna rapport undersöker hur en immersiv och skalbar medieproduktion utformas optimalt för en domskärm där flertalet användare deltar samtidigt. Undersökningen har fokuserat på att besvara hur designen av kontroller görs för att optimera förståelse för ett medie på domskärm, hur nätverksinfrastruktur på bästa sätt hanterar flertalet simultana användare samt hur det går att säkerhetsställa ett immersivt och skalbart system genom tydlighet och underhållningsvärde.

Undersökningen har utförts genom att utveckla ett fleranvändarspel för 5 till 50 användare där spel-skärmen består av en domskärm och varje spelare styr sin karaktär med hjälp av en mobil, handhållen enhet. Spelapplikationen har utformats i samråd med kund på *Visualiseringscenter C* i Norrköping som avser introducera ett fleranvändarspel för deras domskärm som kan köras inför ordinarie filmföreställningar i domen och underhålla besökare under väntetiden. Kunden krävde även att spelets tema riktade sig mot en miljömässig utmaning i samhället varpå spelidén bygger på en undervattensmiljö där varje spelare är en dykare som hjälper till att rensa havet på plast. Spelapplikationen utvecklades med hjälp av C++ och OpenGL och det interna verktyget *Simple Graphics Clustering Toolkit* (SGCT) låg till stor grund för att integrera spelapplikationen med dombiografens skärmar. Nätverkskommunikationen sköttes med hjälp av kommunikationsprotokollet *WebSocket* samt biblioteket *Node.js* och webbapplikationen programmerades med HTML och CSS. Designrelaterade frågeställningar har besvärats genom en användarundersökning.

Projektgruppen bestod av sex utvecklare och arbetet delades in i tre olika områden: modellering och grafisk design, spelutveckling samt nätverks- och serverhantering. Gruppen har följt en agil utvecklingsmetodik baserad på *Scrum*, där samtliga dokument samlades på *GitHub Projects* och *Google Drive*. Utvecklingsarbetets delades in i sprintar och bröts ned i delmål i form av leverabler och milstolpar. Regelbundna möten hölls för att diskutera framsteg och svårigheter.

Undersökningen i detta projekt kom fram till att kontrollerna för ett spel i en dombiograf bör utformas med hjälp av ett fåtal, och enkla, knappar som gör att användaren konstant kan ha sina fingrar på sin mobila enhet. Det ger också en enkelhet då användaren kan fokusera på domskärmen och hålla blicken lyft uppåt istället för att titta ned på sin mobila enhet.

En nätverksinfrastruktur som hanterar såväl kommunikationsvägar gentemot ett varierande antal simultana användare och eventuella temporära bortfall av användare bör utformas på ett sådant sätt så att datakommunikationen är både skalbar och icke-blockerande genom att se till att spelet inte behöver stanna upp för att skicka eller ta emot data utan att detta sker vid sidan av resterande spellogik. Spelare som av någon anledning har tappat uppkopplingen till spelet bör kunna återansluta sig med en ny spelarkaraktär. Den gamla spelarkaraktären bör i dessa fall tas bort.

De tekniker och strategier som kan utnyttjas för att säkerhetsställa ett dynamiskt spelsystem för allt mellan 5 och 50 simultana användare är framför allt en skalbar spellogik där antalet simultana objekt som ska samlas av spelaren är olika i antalet beroende på hur många spelare som är anslutna. Ytterligare strategier som med fördel kan utnyttjas är tydlighet gällande färger på spelarkaraktärer som kombineras med hjälp av primära och sekundära färger, vilket minimerar risken för förvirring gällande vilken karaktär som hör till vilken spelare.

Noterbart är att detta projekt ej har fokuserat på optimeringar utav lösningen då den globala virus-spridningen av COVID-19 bröt ut i samband med starten av detta projekt vilket tvingade till distansarbete för utvecklingsgruppen. Dessutom blev antalet tester i dombiografen kraftigt minskade och alla fysiska användartester inställda.

Utvecklingsgruppen har trots detta kunnat utföra och besvara de frågeställningar som definierats. Dessutom har ett antal utmaningar gällande hur immersiva och skalbara lösningar för en domskärm utformas optimalt kunnat urskiljas. Formen på en domskärm ställer högre krav på projektion samt spelstyrning då mycket av den intuitiva känsla som finns för vanliga skärmar försätter. Även om en enkel design av kontroller är att sträva efter så noteras det att ytterligare alternativ finns som kan vara värda att utforska i liknande framtida arbeten. På samma sätt finns det även optimeringar som kan undersökas gällande spellogik och nätverkskommunikation som tillåter en än mer immersiv upplevelse.

Slutligen kan projektgruppen dra slutsatsen att ett fleranvändarspel som styrs av handhållna enheter och är avsedd för en dombiograf kan skapa en immersiv och underhållande upplevelse genom en tydlig spellogik och effektiv nätverkskommunikation.

# Innehåll

|   |             |
|---|-------------|
| <b>Sammanfattning</b>                               | <b>i</b>    |
| <b>Figurer</b>                                      | <b>vi</b>   |
| <b>Tabeller</b>                                     | <b>viii</b> |
| <b>Konventioner och ordlista</b>                    | <b>ix</b>   |
| <b>1 Inledning</b>                                  | <b>1</b>    |
| 1.1 Syfte . . . . .                                 | 1           |
| 1.2 Frågeställningar . . . . .                      | 1           |
| 1.3 Avgränsningar . . . . .                         | 2           |
| 1.4 Spelbeskrivning . . . . .                       | 2           |
| <b>2 Relaterat arbete</b>                           | <b>3</b>    |
| 2.1 Immersiva spelupplevelser . . . . .             | 3           |
| 2.2 Utveckling för domskärmar . . . . .             | 3           |
| <b>3 Projektmetodisk redogörelse</b>                | <b>5</b>    |
| 3.1 Utvecklingsmetodik . . . . .                    | 5           |
| 3.2 Organisation . . . . .                          | 5           |
| 3.2.1 Arbetsgrupper och ansvarsfördelning . . . . . | 6           |
| 3.2.2 Möten . . . . .                               | 7           |
| 3.3 Tidsplan . . . . .                              | 8           |
| 3.3.1 Leverabler . . . . .                          | 9           |
| 3.4 Kravhantering . . . . .                         | 9           |
| 3.5 Dokumentationsprinciper . . . . .               | 10          |
| 3.6 Kvalitetssäkring . . . . .                      | 10          |
| 3.6.1 Kodgranskning . . . . .                       | 10          |
| 3.6.2 Användartester . . . . .                      | 10          |
| 3.7 Versionshanteringssystem . . . . .              | 11          |

|  |           |
|--|-----------|
| <b>4 Teknisk redogörelse</b>                                       | <b>12</b> |
| 4.1 Grundläggande, initiala krav och systembegränsningar . . . . . | 12        |
| 4.1.1 Grundläggande krav . . . . .                                 | 12        |
| 4.1.2 Systembegränsningar . . . . .                                | 12        |
| 4.2 Systemarkitektur . . . . .                                     | 13        |
| 4.2.1 Övergripande struktur . . . . .                              | 13        |
| 4.2.2 Delsystem . . . . .  | 13        |
| 4.2.3 Komponenter . . . . .  | 15        |
| 4.3 Tredjepartsbibliotek och -verktyg . . . . .                    | 16        |
| 4.3.1 Node.js . . . . .  | 16        |
| 4.3.2 Libwebsockets . . . . .                                      | 16        |
| 4.3.3 OpenGL . . . . .   | 16        |
| 4.3.4 GLM . . . . .  | 16        |
| 4.3.5 Assimp . . . . .   | 16        |
| 4.3.6 SGCT . . . . .   | 17        |
| 4.4 Utvecklingsmiljö . . . . .                                     | 17        |
| 4.4.1 IDE och externa verktyg . . . . .                            | 17        |
| 4.4.2 Mjukvara för grafisk design och modellering . . . . .        | 17        |
| 4.5 Grafisk design och modellering . . . . .                       | 18        |
| 4.5.1 Hemsidans användargränssnitt . . . . .                       | 18        |
| 4.5.2 3D-modeller . . . . .  | 21        |
| 4.6 Spelapplikation . . . . .                                      | 24        |
| 4.6.1 Grafikprogrammering . . . . .                                | 24        |
| 4.6.2 Spelarkaraktären . . . . .                                   | 25        |
| 4.6.3 Spelsimulering och dataorientering . . . . .                 | 26        |
| 4.6.4 Föroringar i objektpooler . . . . .                          | 27        |
| 4.6.5 Nodsynkronisering . . . . .                                  | 28        |
| 4.6.6 Modellhantering i spelet . . . . .                           | 28        |
| 4.6.7 Position i domen . . . . .                                   | 28        |
| 4.6.8 Kollisionsdetekttering och poängutdelning . . . . .          | 30        |
| 4.6.9 Ekvirektangulär mappning . . . . .                           | 31        |
| 4.7 Nätverkskommunikation . . . . .                                | 33        |
| <b>5 Resultat</b>  | <b>35</b> |
| 5.1 Grafik och modeller . . . . .                                  | 35        |
| 5.1.1 3D-modeller i Blender . . . . .                              | 35        |
| 5.1.2 Spelarmodellens färger . . . . .                             | 37        |
| 5.2 Användartester . . . . .                                       | 38        |

|          |   |           |
|----------|---|-----------|
| 5.3      | Nätverkskommunikation . . . . .   | 38        |
| 5.4      | Spellogik och prestanda . . . . .   | 39        |
| 5.5      | Spelapplikationen . . . . .   | 40        |
| 5.5.1    | Användargränssnitt . . . . .  | 41        |
| 5.5.2    | Fullständig spelupplevelse . . . . .  | 41        |
| <b>6</b> | <b>Analys och diskussion</b>  | <b>43</b> |
| 6.1      | Metod . . . . .   | 43        |
| 6.1.1    | Källkritik . . . . .  | 44        |
| 6.2      | Resultat . . . . .  | 45        |
| 6.3      | Etisk och samhällelig reflektion . . . . .                                    | 46        |
| <b>7</b> | <b>Slutsatser</b>   | <b>47</b> |
| 7.1      | Vidareutveckling . . . . .  | 48        |
| <b>A</b> | <b>Utvecklarnas individuella ansvarsområden och bidrag till slutprodukten</b> | <b>51</b> |
| <b>B</b> | <b>Användarundersökning av styrningsgränssnitt</b>                            | <b>52</b> |
| B.1      | Formulärets frågor . . . . .  | 53        |
| B.2      | Sammanställning av svarsdata . . . . .  | 54        |
| B.2.1    | Sammanställning av fria svar och allmän återkoppling . . . . .                | 55        |
| <b>C</b> | <b>Grafiska element i användargränssnittet</b>                                | <b>56</b> |

# Figurer

|   |    |
|---|----|
| 2.1 En bild från dombiografen på Visualiseringsscenter C när en visualisering från OpenSpace-projektet visas. Bild hämtad från [1]. . . . .   | 4  |
| 3.1 Gantt-schema över utvecklingsteamets initiala plan för arbetets fortgång, sprintar, leverabler och tester. . . . .  | 8  |
| 4.1 Systemets övergripande struktur med tre huvudsakliga komponenter: en hemsida på användarens mobila enhet, en webbserver och dommiljöns huvuddator vilken kommunicerar med projektörernas styrdatorer. . . . .   | 14 |
| 4.2 Vidareutvecklad redogörelse för systemets struktur. . . . .   | 14 |
| 4.3 Modell över de olika klasstrukturerna i spelet. . . . .   | 15 |
| 4.4 Två varianter av en mer utarbetad prototyp av hemsidans skärm för anslutning till spelet. 4.4a visar en mörkare version utifrån en undervattensmiljö och 4.4b visar en ljusare version utifrån en strand mot öppet hav. . . . .   | 19 |
| 4.5 Illustrationer av spelarkarakturen i ett antal olika färgkombinationer. . . . .   | 20 |
| 4.6 Illustration av syrgastuben med tidsindikator i mitten. . . . .   | 21 |
| 4.7 En delmodell av dykarmodellen efter att ha behandlats med hjälp av skulpteringsverktyg i Blender med standardmaterial och ljussättning. . . . .   | 22 |
| 4.8 En modell av en fisk som har texturerats utan korrekt UV-unwrapping till vänster som då har synliga artefakter, speciellt kring nosen. Till höger är samma fisk men med korrekt UV-mappning. Bilden är ett utsnitt ur arbetsytan i Blender. . . . .                           | 22 |
| 4.9 En bild från Blender där modelleringsvyn syns till höger med markerad yta som ska textureras. Till vänster syns fönstret för <i>UV-unwrap</i> där 3D-ytan har projicerats på ett 2D-plan och kan transformeras till hur användaren avser att UV-mappningen ska se ut. . . . . | 23 |
| 4.10 En illustration över hur ett skelett har skapats och adderats inuti en fisk- <i>mesh</i> . Varje ben och led i skelettet kan modifieras och rör sig med hjälp av fördefinierade restriktioner. Bilden är ett utsnitt ur arbetsytan i Blender. . . . .                        | 23 |
| 4.11 Dykarmodell med ett skelett adderat som möjliggör för animering av karaktären. Bilden är ett utsnitt ur arbetsytan i Blender. . . . .  | 24 |
| 4.12 Den maskeringstextur som utnyttjas för att särskilja dykarmodellens kroppsdelar, skapad med hjälp av Blender. . . . .  | 25 |
| 4.13 Illustration över hur nästa färgkombination ges av positionerna 1 och 7 för primär- respektive sekundärfärg. . . . .   | 26 |
| 4.14 Från figur 4.13 inkrementeras sekundärpositionen förbi slutet av dess lista och nollställs. Istället inkrementeras primärpositionen vilket ger nästa färgkombination av positionerna 2 och 0 för primär- respektive sekundärfärg. . . . .                                    | 26 |

|  |    |
|--|----|
| 4.15 Modell av hur unika objekt som lagras konkret i tre olika <code>std::vector</code> kan se ut i datorns minne. De färgade områdena representerar objekten, ofärgade områdena fritt minne. . . . .  | 27 |
| 4.16 Modell av hur minnesallokering med polymorfisk struktur kan se ut i datorns minne. De färgade områdena representerar objekten, ofärgade områdena fritt minne. . . . .   | 27 |
| 4.17 Visualisering av objekts position i domen. Figuren är skapad i Inkscape. . . . .  | 28 |
| 4.18 Hur det tillåtna området på sfären definieras. Det tillåtna området är markerat i grön färg. Figuren är skapad i Inkscape. . . . .  | 29 |
| 4.19 Hjälpvektorer vid beräknandet av korrigerad position. Figuren är skapad i Inkscape. .   | 30 |
| 4.20 Övergripande illustration över ekvirektangulär mappning mellan longitud och latitud $\theta, \phi$ och den horisontella och vertikala texturcoordinaten, $u, v$ . I (a) visas de sfäriska koordinaterna och i (b) de ekvirektangulära texturkoordinaterna. Bild hämtad från [18]. | 31 |
| 4.21 En testscen skapad i Unity med hjälp av en ekvirektangulärmappning och där kameran fångar en $360^\circ$ sfärisk panorama av scenen. . . . .  | 32 |
| 4.22 Till vänster syns ett av bakgrundskoncepten skapade i projektet och till höger är resultatet efter att ha texturerat denna bild på insidan av en sfär i Unity. . . . .  | 32 |
| <br>5.1 Alla de plastmodeller som skapats och texturerats i Blender för att samla i spelet. . .  | 36 |
| 5.2 Den färdiga dykarmodellen som har skapats i Blender visas med grundläggande ljussättning och material till vänster, som wireframe-modell i mitten och med texturering till höger. . . . .  | 36 |
| 5.3 Dykarmodell med texturer och adderad ljussättning samt ett bakgrundsplan texturerat med ett av de framtagna bakgrundskonceptet som sedan har renderats i Blender med renderaren <i>Cycles</i> . . . . .  | 37 |
| 5.4 64 dykare i spelet samtidigt, alla med en egen färgkombination. . . . .  | 37 |
| 5.5 Nodsynkronisering med sex noder utan spelobjekt. . . . .   | 39 |
| 5.6 Nodsynkronisering med sex noder och 402 spelobjekt. . . . .  | 40 |
| 5.7 Hemsidans olika skärmar i dess form vid rapportens skrivande. . . . .  | 41 |
| 5.8 Bild på spelapplikationen i dombiografen på Visualiseringscenter C. . . . .  | 42 |
| 5.9 Bild på spelapplikationen i dombiografen på Visualiseringscenter C. . . . .  | 42 |
| <br>B.1 Prototyp av grafiskt gränssnitt för styrning med två knappar. . . . .  | 52 |
| B.2 Prototyp av grafiskt gränssnitt för styrning med en <i>slider</i> . . . . .  | 52 |
| <br>C.1 Färgpalett för utvecklingsprojektets användargränssnitt. . . . .   | 56 |
| C.2 Färgpalett för bakgrundsfärger gällande framtagandet av konceptbilder. . . . .   | 56 |
| C.3 Färgpalett för förgrundsdetaljer gällande framtagandet av konceptbilder. . . . .   | 56 |
| C.4 Till vänster visas en första prototyp av hemsidans olika skärmar. Till höger visas uppmaningen till användaren att hålla sin <i>smartphone</i> i landskapsläge då styrningsgränsnittet ej är utformat för porträtläge. . . . .   | 57 |

# Tabeller

|     |   |    |
|-----|---|----|
| 3.1 | Ansvarsfördelning av utvecklingsteamets medlemmar mellan de olika arbetsgrupperna samt ytterligare organisatoriska ansvarsroller. . . . .             | 6  |
| 4.1 | Strukturering av kommunikation från webbserver till spelapplikation . . . . .   | 34 |
| 4.2 | Strukturering av kommunikation från spelapplikation till webbserver . . . . .   | 34 |
| B.1 | Svarsdata från användarundersökningen avseende målgruppstillhörighet av de svarande.  | 54 |
| B.2 | Svarsdata från fråga 1 från del 2 samt del 3 ur formuläret avseende förståelse av de två styrningsgränssnitten. . . . .                               | 54 |
| B.3 | Svarsdata från fråga 2 från del 2 samt del 3 ur formuläret avseende att få sin karaktär att svänga med de två styrningsgränssnitten. . . . .          | 55 |
| B.4 | Svarsdata från fråga 3 från del 2 samt del 3 ur formuläret avseende att få sin karaktär att simma rakt fram med de två styrningsgränssnitten. . . . . | 55 |
| B.5 | Svarsdata från samtliga frågor från del 4 ur formuläret avseende en direkt jämförelse mellan de två styrningsgränssnitten. . . . .                    | 55 |

# Konventioner och ordlista

I denna rapport förekommer ett stort antal engelska ord när det kommer till programmeringstermer då en översättning till svenska innebär en otydlig förklaring eller förlorad mening i ordet. Dessa ord markeras i texten som *kursiva* och böjs enligt svenska konventioner, ett antal av dessa ord tas även upp med förklaring i ordlistan nedan.

Ytterligare konventioner är att första gången ett egennamn benämns i texten är denna markerad som *kursiv* och de ord som relateras direkt till kodimplementationen benämns i texten enligt *typewriter*.

## Ordlista

Följande tekniska termer förekommer i rapporten och förklaras här för att ge läsaren en snabb insikt och enkelhet i att använda denna sida för att slå upp specifika termer.

**Agil** – Ett ord som kommer från engelskans ord *agile* som betyder rörlig. Med en agil utvecklingsprocess menas en flexibel och iterativ arbetsmetodik.

**API** – Applikationsprogrammeringsgränssnitt, specificerar hur kommunikation mellan en programvara och olika applikationer.

**Assimp** – *Open Asset Import Library*, ett bibliotek som är *open source* och hanterar inläsning av 3D-modeller.

**Dombiograf** – En biograf med en stor halvcirkelformad sfärisk skärm som omsluter besökaren.

**Fisheye-projektion** – En cirkulär projektion som mappar en plan yta på en sfärisk yta.

**Git repository** – En virtuell lagringsplats för alla dokument och filer som hör till projektet.

**GLSL** – *OpenGL Shading Language*, ett programmeringsspråk som används för grafisk programering i OpenGL.

**Handler class** – En form av klass vars uppgift är att hantera instanser av ett annat objekt genom specialiserad funktionalitet.

**HTTP** – *Hypertext Transfer Protocol*, ett kommunikationsprotokoll som används för att överföra webbsidor över internet.

**HTTPS** – *Hypertext Transfer Protocol Secure*, ett kommunikationsprotokoll för krypterad transport av data för HTTP-protokollet.

**IDE** – *Integrated Development Environment*, är en mjukvaruapplikation med möjlighet att skriva programkod med korrekt syntax, kompilera programmet och utföra felsökning.

**Issue** – Innehåller information om något arbete som ska utföras, problem som ska åtgärdas eller annat som relaterar till kodbasen på GitHub.

**Libwebsocket** – Ett bibliotek som möjliggör integration av *WebSocket*-protokollet i C++.

**Master-gren** – Även kallad *master*. Den gren i versionshanteringssystemet som alltid besitter en funktionellt duglig version av produkten. Utifrån denna skapas sedan förgreningar där arbete med experimentell funktionalitet sker.

**MVP** – *Minimum Viable Product*, är en variant av produkten som precis uppfyller de krav som kunden har ställt på slutprodukten och fungerar som ett sätt att ge återkoppling för fortsatt produktutveckling.

**npm** – *Node Package Manager*, en pakethanterare för *Node.js* som används för inkludering av bland annat websocketfunktionalitet i detta projekt.

**OpenGL** – *Open Graphics Library*, är ett API som hanterar grafikrendering.

**Open source** – En programvara eller källkod som är fritt tillgänglig att använda och modifiera.

**Overhead** – Indirekt och extra processorkraft som används för att genomföra en uppgift. Till exempel fördräjning i grafikkortsdrivrutiner vid rendering av 3D-modeller.

**Produktbacklogg** – En lista som används för att samla alla arbetsuppgifter som behöver utföras i utvecklingsarbetet.

**Pull request** – Skapas på GitHub av en utvecklare som vill sammanfoga en gren med *master*-grenen. Här kan resterande utvecklare granska och diskutera tillhörande ny kod.

**Quaternion** – Benämns även kvaternion vilket är ett matematiskt objekt som kan användas för att representera vinklar.

**README-fil** – En dokumentationsfil som innehåller information om övriga filer och dokument i ett projekt.

**Scrum** – Ett agilt ramverk för systemutveckling och framtagande av produkter.

**SGCT** – *Simple Graphics Clustering Toolkit*, ett internt verktyg framtaget av Linköpings Universitet som används för att skapa innehåll för domskärmar samt möjliggör för projektioner och testning på vanliga datorskärmar.

**Shader** – Ett program som bland annat hanterar pixelpositioner, färgsättning och ljusberäkningar inom en scen i datorgrafik. Skrivas exempelvis i ett programmeringsspråk som GLSL.

**Singleton** – Ett designmönster som begränsar antalet instanser av en klass till ett enskilt objekt.

**Sprintbackloggen** – En spaltad lista där arbetsuppgifter för den nuvarande sprinten samlas och spåras enligt ”ej påbörjade”, ”under utveckling”, ”redo för granskning” och ”färdiga”.

**Visualiseringsscenter C** – En mötesplats kring visualisering och forskning som ligger vid Linköpings Universitet i Norrköping. Har förutom utställningar med inriktning mot visualisering en dombiograf för filmvisning.

**WebSocket** – Ett kommunikationsprotokoll som möjliggör för tvåvägskommunikation över en kopplad server.

# Kapitel 1

## Inledning

Det finns ett pågående samarbete mellan Linköpings Universitet och Norrköpings kommun för att främja universitetets forskningsarbete inom bland annat visualisering och för att väcka allmänhetens intresse till området. Detta samarbete har utmynnlat i Visualiseringscenter C [1] där besökare har möjlighet att utforska både visualisering som virtuell verklighet och andra områden förknippade med medieteknik. Huvudattraktionen på Visualiseringscenter C är domen, där det finns plats för cirka 100 personer att omslutas av den 15 meter i diameter stora kupolen. Sedan 2019 skapas den immersiva bilden av sex stycken laserprojektorer. Varje projektör har en upplösning på  $4096 \times 2160$  bildpunkter och en uppdateringsfrekvens på 120 Hz. Dessa egenskaper tillämpas för domens huvudsakliga användningsområde: den immersiva filmupplevelsen.

Visualiseringscenter C är intresserade av att erbjuda dess besökare en interaktiv upplevelse i samband med domens föreställningar genom att förse dem med något kul att göra tiden innan filmen börjar. Kunden beskrev en idé för ett fleranvändarspel i domen där varje besökare kontrollerar en spelkaraktär på domytan. Kundens huvudkrav var att spelet skulle styras med besökarnas *smartphone* och att spelet skulle ha ett miljötema.

### 1.1 Syfte

Syftet med detta projekt är att undersöka hur en immersiv och skalbar medieproduktion utformas optimalt för en domskärm där flertalet användare engageras samtidigt. Utvecklingsgruppen utgår främst ifrån att utforma ett fleranvändarspel som visas på domskärmen i dombiografen där karaktärer styrs med hjälp av mobila enheter då ett spel på ett gynnsamt sätt väver samman både engagemang hos publiken i dombiografen samt skalbarhet i produktionen. Projektet ska utvärdera hur de underliggande systemen kan utvecklas för att skapa en underhållande och lättbegriplig upplevelse för allt ifrån 5 till 50 användare samt undersöka vad som utgör väl fungerande interaktionsdesign i en dombiograf där användaren ansluter och interagerar med systemet genom en mobil enhet eller *smartphone*.

### 1.2 Frågeställningar

De frågeställningar detta projektarbete avser att besvara redogörs för nedan.

- På vilket sätt bör kontrollerna för ett spel i en dombiograf utformas för att vara lättbegripliga och för att användaren ska kunna behålla sin blick fokuserad uppåt på dombiografens skärm?

- Hur utvecklas en nätverksinfrastruktur för att hantera såväl kommunikationsvägar gentemot ett varierande antal simultana användare – beroende på antal besökare i domen – som eventuella, temporära bortfall av användare – exempelvis på grund av en avbruten nätverksanslutning?
- Vilka tekniker och strategier kan utnyttjas för att säkerställa ett dynamiskt spelsystem – där allt emellan 5 och 50 användare kan interagera simultant – som är väl anpassat för en dombiograf med avseende på tydlighet och underhållningsvärde för användaren?

## 1.3 Avgränsningar

Ett antal avgränsningar har tagits i beaktande. För att minska arbetet för utvecklarna är spelet speciellt utformat för dombiografen på Visualiseringsscenter C i Norrköping. Denna avgränsning togs eftersom utvecklarna hade begränsat med tid och endast tillgång till den ovannämnda domen. Ännu en avgränsning är att spelet är anpassat för att fungera optimalt med minst 5 och maximalt 50 användare, trots att dombiografen på Visualiseringsscenter C kan ta emot 100 besökare. Spelets hemsida är optimerad för de två webbläsarna *Google Chrome* och *Safari*.

På grund av den rådande situationen med den globala virusspridningen av SARS-CoV-2 (COVID-19) begränsades projektgruppen i sitt arbete. Allt arbete, såsom kodutveckling och möten, fick ske på distans. Det togs beslut om att inga användartester skulle genomföras i domen och användartesterna blev därför begränsade till att ske på distans via formulär. Antalet planerade domtester på Visualiseringsscenter C påverkades även av denna globala pandemin och blev därför färre, projektgruppen har därför inte kunnat utföra tester i en högre grad.

## 1.4 Spelbeskrivning

Med kundens givna krav om ett spel med miljöaspekter och tanken att styra en karaktär med en *smartphone* utformades spelidéen. Varje besökare kontrollerar en dykare som samlar poäng genom att rensa upp föroreningar i ett vattendrag. Föroreningarna dyker kontinuerligt upp utspritt på spelplanen och den första spelaren som rör föroreningen rensar upp den. När en spelare rensar upp en förorening tilldelas poäng vilket uppdateras på spelarens *smartphone*.

# Kapitel 2

## Relaterat arbete

I takt med en allt mer digitaliseringad värld skapas fler och fler immersiva upplevelser som syftar till att engagera spelaren på interaktiva sätt genom exempelvis virtuell verklighet eller unika miljöer, så som en dombiograf. Det existerar många studier och projekt som har relevans till detta utvecklingsprojekt, både arbeten mot spelutveckling för immersiva miljöer och användningen av domskärmar för användarupplevelser.

### 2.1 Immersiva spelupplevelser

När spelupplevelser flyttas från en vanlig 2D-skärm till mer immersiva miljöer ställs ett ökat krav på spelprogrammeringen och på spelberättandet. I [2] fastställs vikten av att ta hänsyn till den unika förutsättning som finns i immersiva miljöer och både dra nytta av hur berättande kan göras ännu mer uppslukande, men även se upp med problematik som kan uppstå i denna överföringen genom att den vanliga förståelse för spelmekanik kan gå förlorad. I [3] dras slutsatser kring hur en positiv bild och användarupplevelse kan skapas för en publik genom att de alla har känslan av att vara aktiva i spelet men också har enkelt att förstå det. Detta skapar en grundtanke till hur ett dynamiskt spelsystem som i detta projekt bör utformas med betoning på spelberättande för att vara engagerande för alla spelare. Att använda de strategier som lyfts i [2] och [3] ger goda möjligheter för att undersöka hur ett skalbart och engagerande spelsystem kan skapas i en immersiv miljö.

### 2.2 Utveckling för domskärmar

Interaktiva upplevelser som visas på en domskärm har blivit vanligare i takt med att det blir mer åtkomligt genom att externa verktyg gör att det går att skapa en virtuell utvecklingsmiljö utan tillgång till en faktisk dommiljö. Genom att skapa simuleringar av den projektion som sker i en dombiograf på en vanlig dator börjar kraven och frågorna istället riktas mot hur dessa interaktiva upplevelser kan designas på bästa sätt och vilka begränsningarna är när det kommer till att skapa media för en domskärm [4], vilket har inriktat mer forskning mot dessa typer av immersiva miljöer.

Den bas som Visualiseringcenter C har använt sig av för att kunna skapa domproduktioner är det interna verktyget SGCT som används för att skapa projektioner av en applikation på en domskärm. Detta verktyg är ett av de exempel på verktyg som kan användas för att utveckla applikationer för en domskärm utan tillgång till en domskärm då SGCT kan generera en *fisheye-projektion* av applikationen på en lokal dator. Biblioteket SGCT har tidigare använts i projekt utvecklade för dombiografen på Visualiseringcenter C. Ett exempel på detta är ett projekt som heter *OpenSpace* vilket är ett globalt



Figur 2.1: En bild från dombiografen på Visualiseringsscenter C när en visualisering från OpenSpace-projektet visas. Bild hämtad från [1].

samarbete mellan ett antal forskningsgrupper på olika universitet och museum och syftar till att skapa visualiseringar och förståelse om rymden. Projektet är baserat på öppen källkod från *National Aeronautics and Space Administration* (NASA), vilket är USA:s federala myndighet för luft- och rymdfart, som har samlats in under de rymdfärder som NASA har genomfört<sup>1</sup>. Ett exempel från en visualisering av OpenSpace i Visualiseringsscenter C kan ses i figur 2.1.

OpenSpace används i Visualiseringsscenter C genom att, med hjälp av SGCT, använda domteaterns immersiva miljö för att visualisera astronomiska observationer. Det verktyg som OpenSpace presenterar har designats och utvecklats med förhoppningar om att kunna skapa en språngbräda för nya projekt av interaktiv och datadriven utveckling av interaktiva visualiseringar [5].

Linköpings Universitet är ett av de samarbetande universiteten inom OpenSpace vilket har gett en möjlighet för studenter på Linköpings Universitet att få tillgång till dessa visualiseringssverktyg och dombiografen på Visualiseringsscenter C och därmed kunnat utforma spel för denna plattform. Genom iterativa projekt med hjälp av verktyget SGCT har många undersökningar kring hur infrastrukturen bör utformas för ett spel i dombiografen på Visualiseringsscenter C kunnat utföras. Genom det första projektet under 2014 kunde slutsatsen dras att möjligheten för att skapa spel för en dombiograf är god men att extra tid behöver läggas på att få det att fungera för denna typ av miljö jämfört med mer traditionella plattformar [6]. Genom att dra nytta av att kunna bygga spelutvecklingen på tidigare projekt finns goda förutsättningar för att bygga infrastrukturen på bästa sätt, vilket ses för ett liknande projekt som genomfördes under 2019 där de slutsatser som dras av spelutvecklingen har betoning på hur design görs på bästa sätt för dombiografen [7] och inte enbart hur det ska implementeras för att ens fungera.

Genom det arbete som gjorts i OpenSpace samt de tidigare domproducerade spel inom projekt på Linköpings Universitet fungerar som en grund i detta utvecklingsprojekt för att kunna skapa en nätverksinfrastruktur som hanterar kommunikationsvägar och nätverksanslutning så bra som möjligt.

<sup>1</sup>OpenSpace – ett globalt, öppet projekt för att visualisera universum, <https://www.openspaceproject.com/>

# Kapitel 3

## Projektmetodisk redogörelse

Detta kapitel redogör för projektarbetets val av utvecklingsmetodik, Scrum, samt hur denna har använts och påverkat arbetets ansvarsfördelning, mötesprinciper och allmänna rutiner. Den initiala tidsplanen för projektet samt projektets huvudsakliga milstolpar och leverabler läggs fram. Även rutiner gällande kvalitetssäkring och versionshantering beskrivs.

### 3.1 Utvecklingsmetodik

Projektarbetet använde den agila utvecklingsmetodiken Scrum. En agil metodik ansågs lämplig för detta projekt i och med att en initial förståelse för problemområden saknades, vilket medförde en hög risk för oförutsedda problem under utvecklingsprocessen. Med de principer och rutiner som förespråkas av agila metodiker kunde utvecklingsteamet därmed försäkra sig om en godtagbar och fungerande slutprodukt trots eventuella motgångar.

Slutproduktens krav formulerades först på en hög nivå i form av större och mer diffusa funktionalitetsmål vilka benämndes *epics*. Dessa bröts i sin tur ned till mindre mer hanterbara arbetsuppgifter vilka benämndes *tasks*. Såväl *tasks* som *epics* samlades under utvecklingsprocessens gång i projektets produktbacklogg för att både täcka produktens krav samt hantera eventuella oförutsedda problem. Avsnitt 3.4 redogör mer detaljerat för strukturen och rutinerna kring projektets produktbacklogg.

Då projektarbetet använde Scrum som utvecklingsmetodik följde utvecklingsprocessens huvudsakliga struktur, organisation och möten av de principer som läggs fram i *The Scrum Guide* [8]. Arbetet strukturerades således in i sprintar – perioder om två arbetsveckor med fokuserat arbete mot ett antal förbestämda kortsiktiga mål – vilka resulterade i ett antal mindre men fullt funktionella inkrement av produkten. En redogörelse för utvecklingsarbetet intiala plan för dessa sprintar återfinns i avsnitt 3.3.

### 3.2 Organisation

Utvecklingsteamet för projektarbetet bestod av sex utvecklare. För att utvecklingsarbete skulle hålla en god organisation delades dessa sex utvecklare in i tre arbetsgrupper med specifika ansvarsområden för projektet. Utvecklarna höll även regelbundna möten för att diskutera arbetets fortskridande, eventuella problem samt framtida arbetsuppgifter.

### 3.2.1 Arbetsgrupper och ansvarsfördelning

Utvecklarna delades in i tre huvudsakliga arbetsgrupper, där varje arbetsgrupp befattade sig med en viss del av det planerade utvecklingsarbetet. Utvecklarna som var tilldelad en viss arbetsgrupp ansvarade för att arbetet tillhörande den gruppen blev utfört, men de kunde också arbeta med andra arbetsgruppars uppgifter. Indelningen uppmuntrade således till viss specialisering inom utvecklingsteamet, men utan att helt diktera vem som skulle arbeta med vad. De tre huvudsakliga arbetsgrupperna vilka utvecklingsteamet delades in i var:

- **Modellering och grafisk design:** Arbete rörande framställning av grafiskt innehåll och 3D-modeller samt animationer. Arbete med hemsidans användargränssnitt och design.
- **Spelutveckling:** Arbete med implementation av funktionalitet för spelets logik i C++ samt för rendering i dommiljön med *OpenGL* och GLSL, med hjälp av verktyget SGCT.
- **Nätverks- och serverhantering:** Arbete med kommunikation av all data mellan spel, webbserver och användarens mobila enhet.

Utöver en av de ovannämnda grupperna tilldelades några gruppmedlemmar ett särskilt organisoriskt ansvar. Dessa organisoriska ansvarsroller hade som syfte att projektets utvecklingsmetodik skulle fungera. Nedan följer en beskrivning av dessa ansvarsroller:

- **Scrum master:** Denna hade ansvar för att Scrums principer efterlevdes och att samtliga möten angivna i avsnitt 3.2.2 hölls och genomfördes enligt Scrums ramar.
- **Produktägare:** Denna hade ansvar för att projektets produktbacklogg var välformulerad och ordnad enligt posternas prioritet för projektet. Produktägaren hanterade även all kontakt med projektets kund samt bokade in möten med denna.
- **Sekreterare:** Denna hade ansvar för att föra mötesprotokoll och därmed dokumentera beslut, framsteg och svårigheter under projektarbetets gång.

En sammanställning av utvecklingsteamet med dess tilldelade ansvarsroller och arbetsgrupper återfinns i tabell 3.1. En mer detaljerad redogörelse för varje utvecklares individuella ansvarsområden och arbetsbidrag till slutprodukten återfinns i bilaga A.

Tabell 3.1: Ansvarsfördelning av utvecklingsteamets medlemmar mellan de olika arbetsgrupperna samt ytterligare organisoriska ansvarsroller.

| Namn                 | Arbetsgrupp                    | Organisorisk roll   |
|----------------------|--------------------------------|---------------------|
| Iris Kotsinas        | Nätverks- och serverhantering  | <i>Scrum master</i> |
| David Robín Karlsson | Spelutveckling                 | Produktägare        |
| Frans Johansson      | Modellering och grafisk design | Sekreterare         |
| Josefine Klintberg   | Modellering och grafisk design |                     |
| Erik Larsson         | Nätverks- och serverhantering  |                     |
| Algot Sandahl        | Spelutveckling                 |                     |

### 3.2.2 Möten

Samtliga av utvecklingsarbetets organisationsmässiga möten var planerade att hållas i en förbokad lokal på Campus Norrköping – detta för att undvika externa distraktioner under mötet. Då rådande omständigheter under vårterminen 2020 förhindrade detta hölls istället möten på distans med hjälp av *Microsoft Teams*.

Utvecklarna hade inte samordnade scheman utöver projektarbetet. Det var alltså inte möjligt att fastställa bestämda mötestider för samtliga arbetsveckor; varje ny arbetsvecka medförde nya mötestider. För att således säkerställa att så många av utvecklingsteamets medlemmar kunde delta under den kommande veckans möten planerades dessa in vid slutet av varje arbetsvecka i samråd med samtliga medlemmar. Nedan följer en redogörelse för projektarbetets möten och dess rutiner.

#### Sprintplanering

Varje sprint inleddes med ett planeringsmöte där sprintens huvudsakliga mål formulerades. Projektets produktbacklogg utvärderades, dess *epics* och *tasks* prioriterades och förtydligades varefter en separat sprintbacklogg skapades med de *tasks* som var planerade att utföras under den kommande sprinten. Dessa möten bokades in av *scrum mastern*.

#### Dagliga möten

Varje arbetsdag inleddes med ett möte där varje utvecklare redogjorde för utfört arbete samt eventuella problem denne fastnat med. Därefter reflekterade utvecklingsteamet över vad som borde utföras enligt uppgifter i rådande sprintbacklogg. Arbetsuppgifter för den kommande arbetsdagen delades sedan ut till samtliga utvecklare. Dessa möten bokades in av *scrum mastern*.

#### Sprintåterblick och -utvärdering

Vid slutet av varje sprint hölls ett utvärderingsmöte där projektets framsteg och eventuella utvecklingsmetodiska problem diskuterades. Förslagna justeringar gällande utvecklingsmetodiken, arbetets tidsplan och dylikt kunde därefter inkorporeras i nästkommande sprint, således försäkrades en väl fungerande arbetsdynamik inom utvecklingsteamet. Dessa möten bokades in av *scrum mastern*.

#### Kundmöten

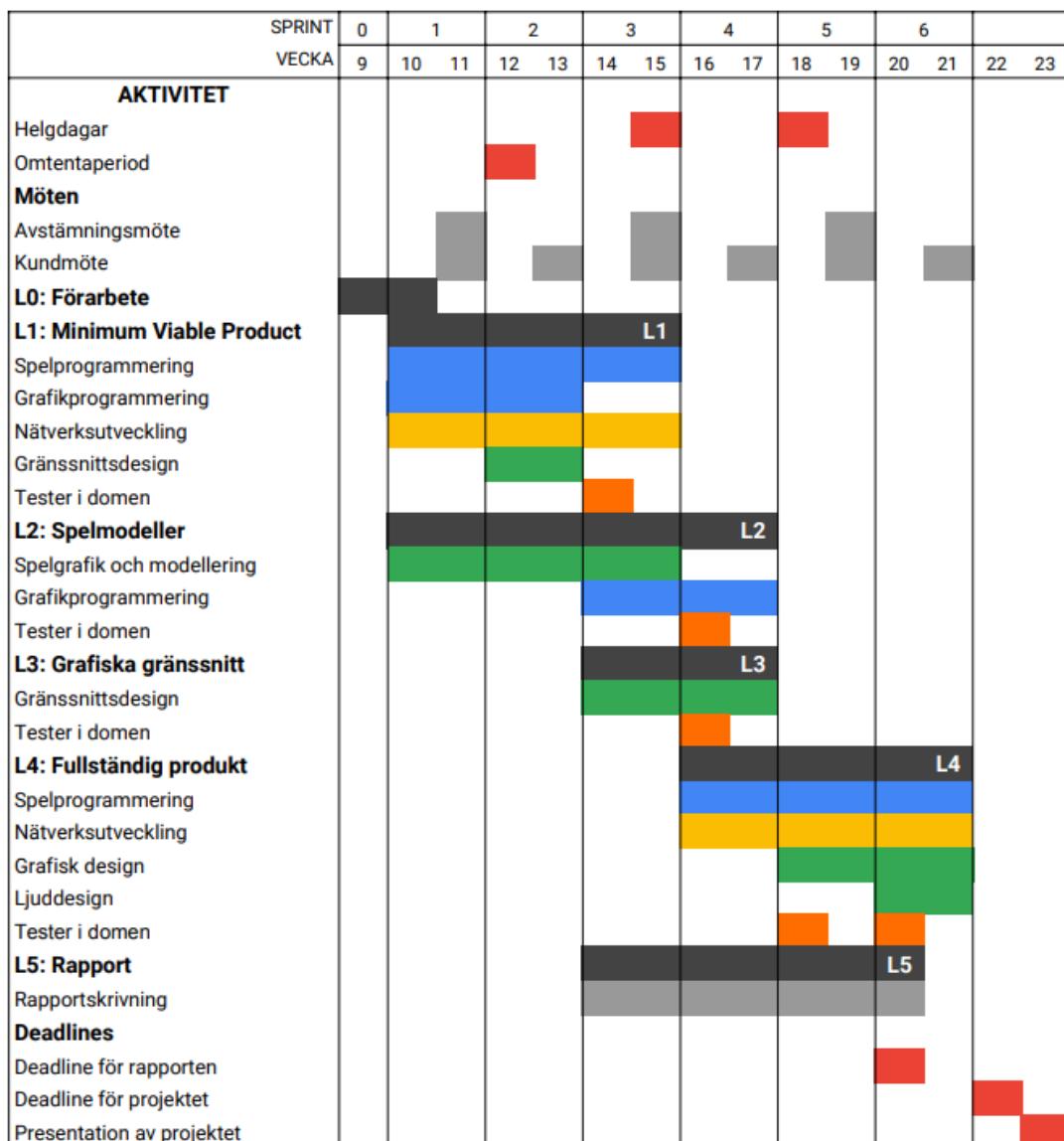
Då utvecklingsarbetet hade uppnått betydliga framsteg i slutproduktens funktionalitet bokades möten in med kunden. Under dessa möten hade utvecklingsteamet möjlighet att testa det utvecklade systemet i dommiljön och stämma av arbetets framsteg med kunden. Kunden hade här möjlighet att komma med återkoppling och förslag till utvecklingsteamet för arbetets fortgång. Dessa möten bokades in av produktägaren.

Utvecklingsteamet hade dessutom möten med kunden i projektets initiala skeden för att diskutera kravspecifikation, speldéer och teknologin tillhörande dommiljön.

### 3.3 Tidsplan

Utvecklingsarbetet utfördes under 12 veckor med start den 2 mars 2020 och slutlig deadline den 27 maj 2020. Arbetet planerades därav grovt in över 6 sprintar à 2 arbetsveckor. Med en arbetsvecka, för detta projekt, menades 3 arbetsdagar då utvecklingsteamets medlemmar hade diverse parallella arbeten att utföra under denna period. Antalet sprintar samt dess längd var initialt sett lösa för att kunna preciseras eller förändras till följd av diskussioner under utvärderingsmöten. Se avsnitt 3.2.2 för en redogörelse för arbetets mötesrutiner.

En översiktig plan för projektets fortgång och sprintar sammanställdes under projektets tidiga skeenden och återfinns i Gantt-schemat i figur 3.1. Denna redogjorde för utvecklingsteamets initiala plan angående parallelisering av arbete, tider då leverabler planerades vara uppnådda samt när tester planerades utföras. Notera dock återigen, enligt föregående stycke, att denna plan kom att preciseras och förändras under projektets gång.



Figur 3.1: Gantt-schema över utvecklingsteamets initiala plan för arbetets fortgång, sprintar, leverabler och tester. Schemat skapades med hjälp av *Google Sheets*.

### 3.3.1 Leverabler

Utvecklingsarbetets framsteg gentemot slutprodukten bröts ned i ett antal delmål i form av leverabler. Dessa motsvarade någon samling avklara milstolpar gällande dokumentation, implementerad funktionalitet eller liknande presenterbara framsteg. Projektets leverabler, benämnda L0–L5 i Gantt-schemat i figur 3.1, redogörs för kort nedan.

**L0 Förarbete:** Projektets plan fastställdes tillsammans med en beskrivning av utvecklingsteamets vision för slutprodukten samt en initial kravspecifikation utifrån kundens önskemål.

**L1 Minimum Viable Product (MVP):** Projektet uppnådde den minsta uppsättningen av implementerad funktionalitet som krävdes för att presentera en godtagbar produkt. Till detta hörde grundläggande spellogik, rendering och nätverkskommunikation mot användarens mobila enhet för att tillåta styrning via en hemsida. Spelets karakterär bestod av enkla geometriska former. All denna funktionalitet testades i dommiljön.

**L2 Spelmodeller:** Färdiga modeller för spelets karakterärer och andra objekt färdigställdes och ersatte de tidigare enkla geometriska formerna. Spelarkarakterärens modell var utformad för att stödja en visuell variation för upp till 50 unika instanser.

**L3 Grafiska gränssnitt:** Spelets grafiska gränssnitt, såväl i användarens mobila enhet som på domens skärm, utformades i färdig form och ersatte tidigare prototyper. Spelets principer, styrning, mål, poängsättning samt den egna spelarkarakterären presenterades för användaren i och med dessa gränssnitt.

**L4 Fullständig produkt:** Produkten uppnådde alla krav från kunden samt var förenligt med utvecklingsteamets vision och spelidé givna ur L0. Spelets parametrar för funktionalitet som styrning, poängsystem och ljudeffekter finjusterades och en slutgiltig produkt presenterades för kunden.

**L5 Rapport:** En rapport med redogörelse för projektets syfte, metodik, tekniska lösningar och resultat sammanställdes av utvecklingsteamet.

## 3.4 Kravhantering

Till följd av Scrums principer utnyttjade utvecklingsarbetet ett system av anslagstavlor där arbetsuppgifter i form av poster samlades och dess framsteg spårades. Dessa kom i två former: en projektmässig produktbacklogg och en sprintmässig sprintbacklogg. Den förstnämnda, enligt beskrivning i avsnitt 3.1, utgjorde en uttömnande lista av planerad funktionalitet för att uppnå projektets krav. Den sistnämnda bestod av de specifika uppgifter som planerades implementeras under en viss sprint. Denna bestod av kolumner vilka benämnde uppgifter som ”ej påbörjade”, ”under utveckling”, ”redo för granskning” och ”färdiga”. Allt eftersom arbete med en viss uppgift fortskred förflyttades den tillhörande posten genom kolumnerna. Utvecklingsteamet erhöll således en god överblick över vilket arbete som var utfört, vilket arbete som stod stilla samt vilket arbete som ej påbörjats.

Då utvecklingsarbetet redan använde GitHub för kodelinering och versionshantering, se avsnitt 3.7, föreföll de tillhörande verktygen *GitHub Issues* och *GitHub Projects* som goda val för kravhantering. Med *issues* kunde utvecklarna sammanställa uppgifter såväl *tasks* som epics, kopplade till specifika milstolpar och kategoriserade utifrån arbetsgrupp och prioritet, i en lista. Denna lista utgjorde projektets produktbacklogg. Med *Projects* kunde utvecklarna skapa digitala anslagstavlor med kolumner enligt beskrivning i föregående stycke. Till dessa anslagstavlor flyttades uppgifter, *issues*, från projektets produktbacklogg för att skapa en sprintbacklogg.

## 3.5 Dokumentationsprinciper

Under utvecklingsarbetets fortgång dokumenterades projektets beslut, framsteg och svårigheter kontinuerligt i och med mötesprotokollen. Syftet med dokumentationen var att etablera god kommunikation inom utvecklingsteamet samt ge en genomgående översikt för projektet. De tre arbetsgrupperna kunde i och med god dokumentation om sitt arbete på så vis förmedla kunskap om dess domäner — spelutveckling, grafikprogrammering, nätverkskommunikation, användargränssnitt o.s.v. — till övriga utvecklare.

Molnplattformen Google Drive användes för att sammanställa dokumentation som inte direkt berörde kodbasen. Detta medförde att hela utvecklingsteamet hade ständig och gratis tillgång till projektets dokumentation.

Dokumentation tillhörande kodbasen sköttes antingen direkt i koden — i form av kommentarer vilka redogjorde för metoders syfte, funktionalitet och implementation — eller i *README*-filer belägna i närhet till kodbasen. För att undvika överflödig dokumentation av kodbasen placerades hög vikt på välformulerad och självbeskrivande kod enligt princip P.3. i [9]: "Say what should be done, rather than just how it should be done".

## 3.6 Kvalitetssäkring

För att säkerställa sig om en slutprodukt av hög standard tillämpade utvecklingsteamet kvalitetssäkrande rutiner. Dessa bestod framför allt av att all ny kod granskades med avseende på god kodstil, funktionalitet och effektivitet innan den integrerades med kodbasen. Regelbundna tester planerades att hållas in i domen för att verifiera att produkten fungerade i dess målmiljö.

### 3.6.1 Kodgranskning

Då utvecklarna i hög grad arbetade självständigt med att implementera ny funktionalitet granskades all ny kod av andra utvecklare inom gruppen. Åtminstone två kodgranskare tilldelades varje ny komponent eller modul. Granskarna hade i uppgift att kontrollera kodens läsbarhet, förenlighet med etablerad kodstil, effektivitet och funktionalitet. Endast när båda granskarna givit godkänt tilläts den nya komponenten eller modulen integreras i resterande kodbas. Återkoppling på skriven kod skedde främst på GitHub i form av kommentarer i aktuell *pull request* vilket beskrivs vidare i avsnitt 3.7.

Utöver att medföra en välskriven och ändamålsenlig kodbas innebar detta att utvecklingsteamet erhöll en mer omfattande förståelse för spelets system. Detta bidrog överlag till smidigare kommunikation inom utvecklingsteamet då alla var på samma sida gällande kodbasens diverse beståndsdelar.

### 3.6.2 Användartester

Under utvecklingsprocessen utfördes en användarundersökning som var avsedd för att undersöka hur potentiella användare upplevde styrningsgränssnittet. Detta med avsikt att vägleda utvecklingsgruppen mot en så intuitiv, användarvänlig design som möjligt.

Användarundersökningen utformades med ett användarcentrerat förhållningssätt och bestod av både kvantitativa och kvalitativa frågor. Den kvantitativa datan samlades in för att direkt få en mätbar siffra på hur förståelsen för designen såg ut och den kvalitativa datan samlades in för att kunna få höra åsikter och jämförelser mellan de olika designerna och vilken design som användaren helst hade velat använda sig av.

Användarundersökningen utfördes anonymt via ett formulär över internet och data samlades in under 4 arbetsdagar. Alla frågor samt en sammanställning av den insamlade datan kan ses i appendix B. Eftersom undersökningen utfördes helt på distans kunde utvecklingsgruppen arbeta med en användarcentrerad design trots rådande avgränsningar enligt avsnitt 1.3.

## Utvärdering av användargränssnitt

Under utvecklingsprocessen utvärderades ett antal möjligheter gällande hur användaren skulle styra sin karaktär från hemsidan. Dessa redogörs för i mer detalj i avsnitt 4.5.1. Ett anonymt formulär skapades och distribuerades över sociala medier där potentiella användare av gränssnittet fick evaluera prototyper på såväl styrningsgränssnitt som andra komponenter av hemsidan. Frågorna samt svaren från detta formulär återfinns i bilaga B. Utvecklarna utgick från återkopplingen från undersökningen för att justera det slutliga gränssnittet vilket presenteras i avsnitt 4.5.1.

## Tester i dommiljön

För att testa systemets kapacitet för simultana användare planerades det in ett större användartest i dommiljön. Detta test kunde inte genomföras på grund av de rådande omständigheterna med den globala pandemin, som nämnt i avsnitt 1.3. För att trots detta säkerställa att nätverkskommunikationen fungerade utan komplikationer i domen planerades det in domtester för endast utvecklingsteamet. Dessa tester kunde inte ske lika kontinuerligt som planerat och projektgruppen fick därför anpassa sig efter situationen och planera in färre domtester.

## 3.7 Versionshanteringssystem

För att samtliga utvecklare skulle ha tillgång till den senaste versionen av all kod såväl integrerad som under utveckling användes *Git* som versionshanteringssystem med webbtjänsten GitHub som nav. Huruvida en applikation med grafiskt gränssnitt, exempelvis *GitHub Desktop*, eller kommandotolkten användes för att lokalt hantera Git lämnades till varje utvecklare att bestämma.

Projektet bibehöll kontinuerligt en fungerande version av produkten i en *master*-gren på GitHub. Denna version av produkten eftersträvades alltid vara i ett fullt funktionellt stadi med så få problem som möjligt. Under utvecklingens gång kunde därför utvecklarna skapa egna grenar utifrån *master* för att implementera ny funktionalitet, testa en ny idé eller liknande. Enligt rutinerna i avsnitt 3.6.1 tilläts dessa ej sammanfogas med *master* såvida koden inte hade granskats av två övriga utvecklare.

När väl tiden var kommen för att sammanfoga en gren till *master*-grenen skapade den ansvarige utvecklaren en *pull request* på GitHub. I denna refererade utvecklaren till vilka *issues* – poster i relevant sprintbacklogg – arbetet på grenen behandlade, samt tilldelade kodgranskare. Eftersom GitHub *projects* användes gav detta upphov till ett smidigt arbetsflöde där de angivna posterna automatiskt flyttades genom relevant sprintbackloggs kolumner, se avsnitt 3.4, allt eftersom koden granskades, godkändes och sammanfogades med *master*-grenen.

# Kapitel 4

## Teknisk redogörelse

I detta kapitel redogörs för alla tekniska aspekter av projektet som systemets krav och begränsningar, nätverkskommunikation, systemarkitektur och spelets delsystem. Det redovisas även vilka verktyg, tredjepartsbibliotek och utvecklingsmiljöer som har använts.

### 4.1 Grundläggande, initiala krav och systembegränsningar

Utifrån slutproduktens syfte att utformas som kort underhållning i form av ett fleranvändarspel för kundens besökare inför ordinarie föreställningar i domen formuleras systemets grundläggande krav och begränsningar.

#### 4.1.1 Grundläggande krav

För att uppnå sitt syfte måste spelets principer vara lättbegripliga för en bred målgrupp av användare och dess användargränssnitt måste vara väl anpassade för en dombiograf. Spelet måste även kunna hantera ett varierande antal simultana användare till följd av ett varierande antal besökare i domen.

De grundläggande krav som ställs på slutprodukten är därmed att:

- Spelet ska vara nägesvärt och lättbegripligt oavsett om antal spelare är så få som 5 eller upp till 50 personer.
- Spelets principer, mål och design ska anpassas för en bred målgrupp samt för att fungera väl som tidsbegränsad underhållning. Spelets styrning och gränssnitt ska anpassas för en dommiljö.
- Den mobilanpassade hemsidan ska fungera likvärdigt för båda de två webbläsarna Google Chrome och Safari för *smartphones*.

#### 4.1.2 Systembegränsningar

För att leverera bästa möjliga visuella upplevelse till användaren krävs det att spelets system kan rendera som minst 120 bildrutor per sekund. Detta för att maximalt utnyttja dommiljöns projektors uppdateringsfrekvens på 120 Hz. Spelets logik, nätverkskommunikation och rendering nödvändig för varje bildruta måste därmed ske under tidsintervallet  $1/120s \approx 8.33ms$ . Detta krav ska gälla för upp till 50 spelare samtidigt enligt det förstnämnda kravet i föregående avsnitt. Systemet verkar således under följande begränsning:

- För att inte orsaka födröjningar i spelets grafik måste all bakomliggande logik och nätverks-kommunikation för en bildruta ske inom tidsintervallet 8.33 ms.

Då datorerna som renderar bildrutorna i domen använder kraftfulla grafikkort kommer höga antal vertiser inte vara något problem. Därmed är kravet på rendering inriktat på effektiv användning av utvald *API* för datorgrafikrendering.

## 4.2 Systemarkitektur

Utveckling av applikationer för den givna dommiljön medför en del krav på hur det bakomliggande systemet bör struktureras. För att underlätta uppstarten av projektet försåg kunden teamet med en mall<sup>1</sup> innehållande en minimal setup för kommunikation hemsida–webbserver–spelapplikation som låg som grund till projektet. Detta avsnitt redogör för och motiverar den övergripande strukturen av slutproduktens system samt dess aktörer, komponenter, delsystem och interaktionsvägar dåremellan.

### 4.2.1 Övergripande struktur

Den huvudsakliga delen av spelets logik och data hanteras av en applikation som körs direkt i dommiljön. Denna spelapplikation ansvarar för att uppdatera spelets interna tillstånd utifrån inkommande användardata, såsom styrningsdata, och att rendera de bilder som projiceras på domens skärm. Kundens hårdvarusystem för att hantera detta består av ett datorkluster där en huvuddator kommunicerar med sex styrdatorer vilka kontrollerar var sin projektor i domen.

Då det ej är möjligt för användaren att ansluta sig direkt till det ovannämnda domsystemet introduceras en webbserver som är extern till dommiljön. Denna webbserver hanterar kommunikation av data till och från spelapplikationen gentemot ett användargränssnitt för anslutning, styrning och övrig användarinteraktion.

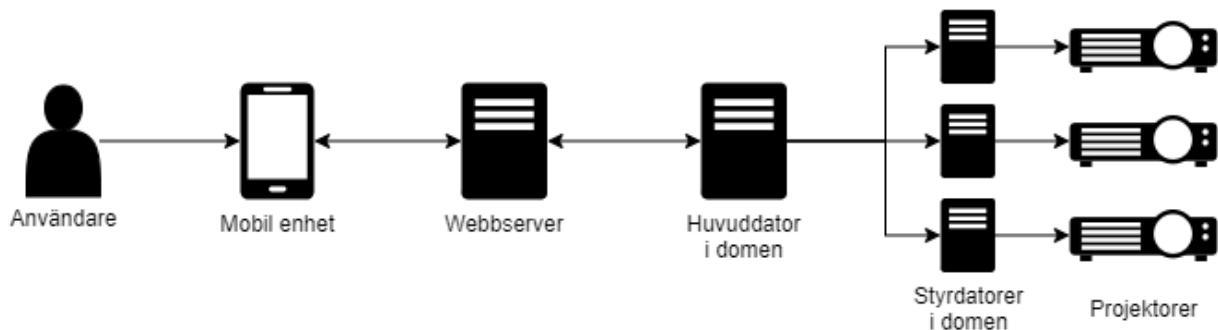
Det krävs att såväl regelbundna som helt nya besökare i domen snabbt kunna ansluta sig till spelet. Av denna anledning består spelets användargränssnitt av en mobilanpassad hemsida vilken användarna utnyttjar både vid anslutning och för att kontrollera sin karaktär. Genom att en hemsida används snarare än en mobilapplikation behöver användaren inte ladda ned något för att kunna delta. Denna hemsida kommunicerar med webbservern vilken i sin tur kommunicerar med huvuddatorn i dommiljön där spelet körs. Utvecklingsarbetet mynnar således ut i ett system med tre huvudsakliga delar. Systemets struktur utifrån beskrivningen i detta avsnitt visas övergripande i figur 4.1.

### 4.2.2 Delsystem

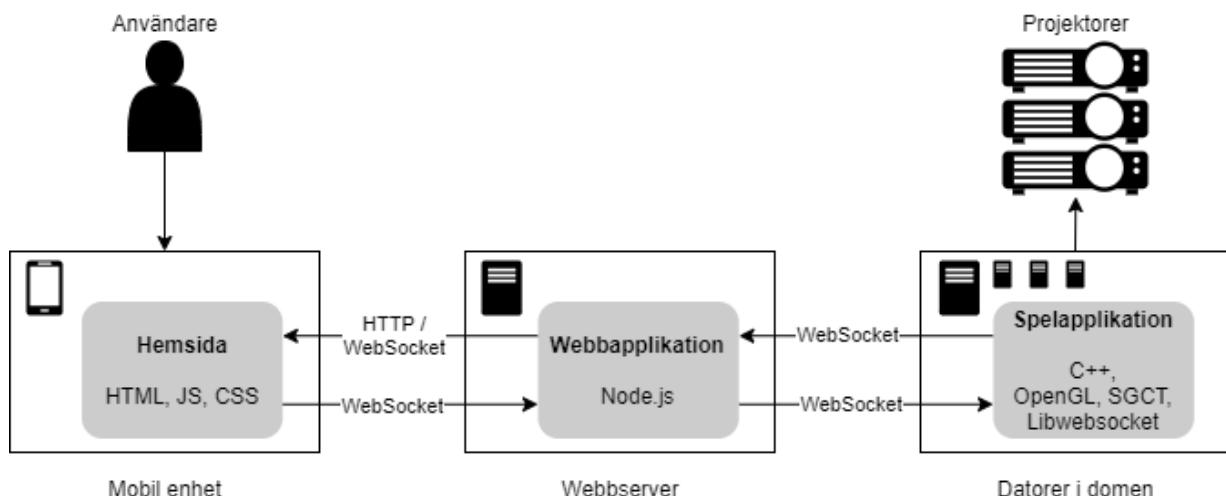
Det utvecklade systemet består enligt beskrivning i föregående avsnitt av tre huvudsakliga delar – en mobilanpassad hemsida, en webbserver och en huvuddator i dommiljön. Systemet har utifrån dessa delar tre huvudsakliga delsystem – hemsidan, en webbapplikation och en spelapplikation – vilka redogörs för vidare i detta avsnitt. En överblick av systemets delar gavs i figur 4.1. Delarnas programmeringsspråk och miljöer samt interaktioner sinsemellan vidareutvecklas i figur 4.2. Tredjepartsbibliotek och dylikt presenteras översiktligt i detta avsnitt och mer detaljerat i avsnitt 4.3.

Den mobilanpassade hemsidan utvecklas med grundläggande tekniker för webbutveckling. Detta anses rimligt i och med dess avsedda funktionalitet att låta användare ansluta med ett eget spelarnamn samt styra sin karaktär. Märkspråket HTML används därmed för att skapa hemsidans statiska innehåll

<sup>1</sup>SGCT-Networked-Application, A. Bock, <https://github.com/alexanderbock/SGCT-Networked-Application>



Figur 4.1: Systemets övergripande struktur med tre huvudsakliga komponenter: en hemsida på användarens mobila enhet, en webbserver och dommiljöns huvuddator vilken kommunicerar med projektörernas styrdatorer. Notera kundens dommiljö har sex av dessa styrdatorer samt projektörer snarare än tre. Diagrammet är skapat med hjälp av *Draw.io*



Figur 4.2: Vidareutvecklad redogörelse för systemets struktur. De tre delsystemen – hemsidan, webbapplikationen och spelapplikationen – visas med dess programmeringsspråk och interaktioner. Diagrammet är skapat i Draw.io.

och struktur, CSS används för *layout* och utseende och JavaScript används för dynamisk funktionalitet. JavaScript används vidare för att etablera kommunikation gentemot webbservern över protokollet *WebSocket*<sup>2</sup>.

Webbapplikationen körs i den externa webbservern. Denna hanterar kommunikation av data mellan användarens mobila enhet och spelapplikationen över protokollet *WebSocket*. Webbapplikationen är skriven i Node.js vilket innebär att både hemsidans och webbapplikationens funktionalitet är utvecklad i samma programmeringsspråk: JavaScript.

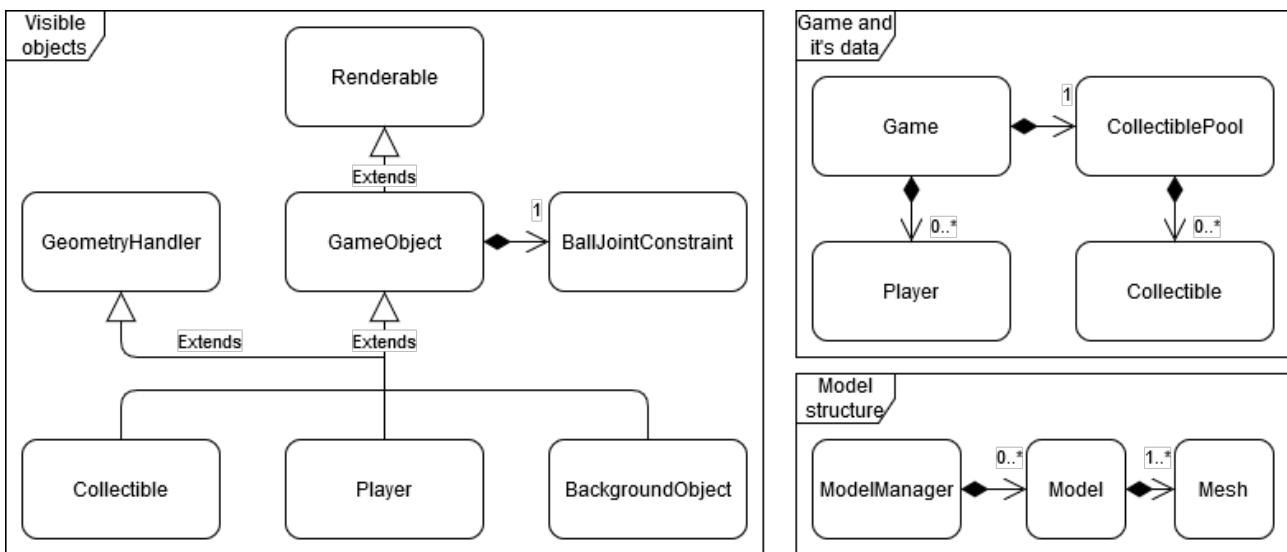
Spelapplikationens logik utvecklas med programmeringsspråket C++. Rendering och projicering på domens skärm hanteras med SGCT som bland annat använder grafik-API:et OpenGL. I och med att OpenGL används skrivs spelapplikationens *shader*-program i GLSL. Spelapplikationen körs enligt figur 4.2 i domens datorer – såväl huvuddatorn som styrdatorerna i figur 4.1 – dessa samverkar med hjälp av SGCT. Endast huvuddatorn hanterar inkommande och utgående data över protokollet *WebSocket* med hjälp av biblioteket *Libwebsocket*<sup>3</sup>.

<sup>2</sup>WebSocket – ett nätverksprotokoll som tillåter tvåvägs-kommunikation mellan sammankopplade parter, <https://tools.ietf.org/html/rfc6455>

<sup>3</sup>Libwebsockets – ett bibliotek för *WebSocket*-funktionalitet i C++, <https://libwebsockets.org/>

### 4.2.3 Komponenter

Den klasstruktur och grundläggande funktionalitetsbeskrivning som systemet byggdes från kan ses i figur 4.3. Spelets mest centrala data och funktionalitet hanteras i en klass Game, implementerad som en explicit singleton [10] då det aldrig bör finnas mer än en instans av denna klass. Denna klass hanterar alla instanser av spelare och förlorenheter och implementerar funktionalitet för spelsimulering och rendering. Denna funktionalitet beskrivs mer utförligt i avsnitt 4.6.3. Denna klass lagrar även den enda instansen av klassen BackgroundObject vilket enbart hanterar bakgrundsbilden i spelet vilket beskrivs mer utförligt i avsnitt 4.6.9.



Figur 4.3: Modell över de olika klasstrukturerna i spelet. Diagrammet är skapat i Draw.io

Alla intergrerbara objekt i spelet ärver dubbelt från `GameObject` och `GeometryHandler`. Klassen `GameObject` hanterar positionsdata och ger ett löfte om positionsuppdatering till dess subklasser. `GameObject` ärver från klassen `Renderable` som endast ger ett löfte om renderingfunktionalitet. `GameObject` lagrar även instans av klassen `BallJointConstraint` som begränsar alla positioner till det tillåtna spelområdet. Denna funktionalitet beskrivs mer utförligt i avsnitt 4.6.7.

`GeometryHandler` hanterar alla data som rör shaders och 3D-modeller och implementerar tillhörande renderingsfunktionalitet. Alla 3D-modeller hämtas från klassen `ModelManager` som lagrar en kopia av varje 3D-modell som spelet använder. Då det bara finns en uppsättning av 3D-modeller för spelet är klassen `ModelManager` implementerad som en explicit singleton. Hur 3D-modeller lagras beskrivs mer utförligt i avsnitt 4.6.6.

Varje spelare representeras som en instans av klassen `Player`. Denna klass lagrar data som rör position (ärvt från `GameObject`), hastighet, poäng, namn och texturfärg. Den icke-triviala funktionaliteten som klassen implementerar består av positionsuppdatering och rendering.

Varje förlorenhet representeras som en instans av klassen `Collectible`. Denna klass lagrar endast data som rör position (ärvt från `GameObject`) och bokföringsdata som används av dess manager klass `CollectiblePool`. Denna klass implementerar funktionalitet för att effektivt skapa och förstöra instanser av `Collectible` vilket beskrivs mer utförligt i avsnitt 4.6.4.

## 4.3 Tredjepartsbibliotek och -verktyg

De externa bibliotek och verktyg som används i projektet är Javascript-plattformen Node.js och biblioteken Libwebsockets, OpenGL, *OpenGL Mathematics* (GLM), *Open Asset Import Library* (Assimp)<sup>4</sup> samt *Simple Graphics Cluster Toolkit* (SGCT). Nedan följer förklaring av dessa verktyg och hur de har använts.

### 4.3.1 Node.js

För kommunikation mellan webbservern och den huvuddatorn i domen används plattformen Node.js. Denna plattform fungerar bra med WebSocket, vilket gynnar kommunikationen med spelapplikationen. Node.js pakethanterare *npm* har ett stort antal paket som smidigt har kunnat inkorporerats i projektet vid behov.

För kommunikation mellan spelapplikationen och webbservern används protokollet WebSocket. Webbservern använder npm-paketet *websocket* för att lägga till WebSocket-funktionaliteten. WebSocket används eftersom det tillåter tvåvägskommunikation på ett användbart sätt där alla parter kan skicka meddelanden till varandra.

### 4.3.2 Libwebsockets

För att huvuddatorn och webbservern ska kunna kommunicera med varandra behöver det finnas en metod i C++-koden som ansluter till Node.js-servern. Detta görs med biblioteket Libwebsockets som är flexibelt och klarar av hög belastning.

### 4.3.3 OpenGL

Implementationen av spelapplikationen görs med hjälp av OpenGL och C++ där valet av programmeringsspråk beror på den interna systemintegrationen med SGCT. OpenGL är ett API för grafikhantering som är oberoende av plattform och operativsystem, och är därför ett effektivt verktyg för att utforma spelet.

### 4.3.4 GLM

GLM är ett matematikkbibliotek för C++ som till stor del är baserat på de typer och operationer som finns i OpenGL:s shaderspråk GLSL. GLM används i projektet för olika typer av beräkningar där de inbyggda typerna i C++ inte är tillräckliga, till exempel beräkning av transformationsmatriser.

### 4.3.5 Assimp

För att underlätta inhämtning av 3D-modeller, texturer och animationer från fil till datorns minne används det öppna biblioteket Assimp. Assimp har syftet att abstrahera bort komplexa detaljer som olika filformatsstrukturer av 3D-modeller från användaren. Det enda biblioteket kräver är sökvägen till filen som ska laddas in, datan lagras sedan i en trädstruktur med noder kan beskriva bland annat vertis- och indexdata, texturer eller animationer, beroende på den inlästa filen.

---

<sup>4</sup>Assimp – länk till biblioteket, <https://www assimp.org/>

### 4.3.6 SGCT

SGCT är ett internt verktyg som utvecklingsgruppen får tillgång till genom ett *repository* på GitHub. Detta verktyg används för att skapa projektioner av en applikation på en domskärm och kan dessutom användas som ett snabbt prototypverktyg för utveckling för domskärmar då SGCT kan generera en *fisheye*-projektion av spelet som kan köras på en lokal dator, med en vanlig 2D-skärm.

Detta bibliotek används för att hantera kommunikationen mellan den interna servern samt de 6 stycken projektorer som finns i dombiografen.

## 4.4 Utvecklingsmiljö

Den utvecklingsmiljö som har använts i detta projekt är utformad för att underlätta arbetet hos utvecklingsgruppen. Den övergripande strukturen är densamma för utvecklarna men valet av IDE har skilt sig åt beroende på vilket operativsystem som utvecklarna har använt sig av.

### 4.4.1 IDE och externa verktyg

I projektet har verktyget *CMake*<sup>5</sup> använts för att bygga, testa och paketera applikationen eftersom detta gör att valet av IDE blir oberoende av operativsystem hos utvecklingsgruppen. Alla utvecklare har därför kunnat välja sin preferens gällande IDE. De IDE som har använts är *Visual Studio 2019*, *XCode* samt *Visual Studio Code* för utvecklare som använder sig av *Windows*, *macOS* respektive *Linux*.

För att snabbt kunna skapa prototyper av hemsidans element och struktur användes webb-verktyget *Figma*. Arbetet med hemsidan utfördes direkt i utvecklarens *smartphone* genom USB-felsökning. Detta lät utvecklarna inspektera hemsidans element som de uppvisades i webbläsaren på en *smartphone* genom att använda utvecklingsverktygen i webbläsaren på utvecklarens dator – detta utnyttjades för utveckling för såväl Google Chrome som Safari.

I spelet används en vanlig bild som bakgrund vilket kräver speciella lösningar för att korrekt rendera på domytan. För att lösa detta användes spelmotorn *Unity* då det finns färdiga verktyg för lösa de speciella problemen som uppstår när en 2D-textur ska visas på en domskärm. Denna teknik för bakgrundsmappning beskrivs vidare i avsnitt 4.6.9.

### 4.4.2 Mjukvara för grafisk design och modellering

För design och modellering har modelleringsprogrammet *Blender* använts. Programmen *Gimp*, *Krita*<sup>6</sup> och *Affinity Designer*<sup>7</sup> har använts för övrig design som behövdes i samband med utformning av hemsidan. Det finns ett flertal modellerings- samt designprogram som hade kunnat användas för detta projekt. Valet föll på program som har en hög industristandard samt med god tillgänglighet för utvecklingsarbete hemifrån – antingen i form av *open source*-mjukvara eller sådan mjukvara som utvecklarna själva hade tillgång till på förhand – i och med de rådande omständigheter projektet utfördes under som lades fram i avsnitt 1.3.

<sup>5</sup>CMake – ett byggverktyg för mjukvaruprojekt över flera plattformar och kompilatorer, <https://cmake.org/>

<sup>6</sup>Krita – ett *open source*-program för digital produktion av målningar och bilder, <https://krita.org/en/>

<sup>7</sup>Affinity Designer – ett program utvecklat av Serif med liknande funktionalitet som Adobe Illustrator, <https://affinity.serif.com/en-us/>

## 4.5 Grafisk design och modellering

Det användargränssnitt och de modeller som har tagits fram i detta projekt har syftat till att samspeла med spelets undervattenstema och spelidé. Designen avser att framhäva en atmosfär av äventyr och förundran, vilket dels grundar spelaren i den givna spelmiljön men dessutom lägger fram en kontrast gentemot förrening av världens hav. Detta val av design avser både motiverar spelarens roll att samla upp förreningarna samt definierat ett äventyrstema som attraherar den breda målgrupp hos publiken som existerar. De olika tekniker, metoder och verktyg som har använts för att åstadkomma dessa användargränssnitt och modeller redogörs för vidare i detta avsnitt.

För att etablera en enhetlighet i designarbetet gällande såväl användargränssnitt som 3D-modellerna formulerades ett grafiskt ramverk tidigt i utvecklingsprocessen. Utvecklingsarbetet gällande spelets grafik, bakgrundskoncept och användargränssnitt skulle i första hand hålla sig till de färgpaletter som redogörs för i figurerna C.1–C.3 i bilaga C. Samtliga paletter sammansättades i webb-verktyget *Colors*. För spelets gränssnittsdesign användes typsnitten *Merriweather* och *Roboto* för rubriks- respektive brödtextselement. Båda dessa typsnitt finns tillhanda via tjänsten *Google Fonts*<sup>8</sup>.

### 4.5.1 Hemsidans användargränssnitt

Användargränssnittet implementeras i den mobilanpassade hemsidan. Genom denna förmedlas i princip all information till användaren gällande spelet. Hemsidan struktureras upp i ett antal skärmar vilka användaren leds genom under spelets gång. Varje skärm uppfyller en specifik funktion som exempelvis att låta spelaren ange sitt namn och ansluta, förmedla information om spelet och sin karaktär samt styra sin karaktär. De första skärmarna användaren bemöts av är utformade med både landskaps- och porträtläge i åtanke. För spelets kontroller ombeds dock användaren att hålla sin *smartphone* i landskapsläge enligt den högra bilden i figur C.4 i bilaga C. Hemsidans samtliga skärmar med dess slutliga funktionalitet anges nedan:

1. **Anslutning:** Användaren bemöts av denna skärm när denne först ansluter sig till hemsidan. Användaren skriver här in sitt namn och ansluter sig till spelet.
2. **Spelinformation:** Användarens spelarkarakter med dess färgkombination presenteras tillsammans med spelets mål och principer. Detta visas innan spelet har börjat.
3. **Spelets kontroller:** Spelets kontroller och spelarens nuvarande poäng visas. I denna skärm implementeras även en tidsmätare för att visa hur mycket tid av spelet som kvarstår.
4. **Slutpoäng:** Spelet är över och användarens slutliga poäng visas.

### Prototyper

De första prototyperna av hemsidans skärmar utvecklades i webb-verktyget Figma. Detta gav utvecklarna en god initial överblick över hur hemsidans olika delar skulle kopplas till varandra samt hur elementen i varje skärm kunde läggas ut. Till vänster i figur C.4 i bilaga C visas prototyperna i Figma där samtliga skärmar ingår med undantag för den gällande återanslutning vilken utvecklades senare.

Utifrån dessa tidiga prototyper formulerades en funktionell struktur för hemsidan i HTML, CSS och JavaScript. Med denna bas väl etablerad kunde även andra delar av utvecklingsarbetet fortgå parallellt

<sup>8</sup>Google Fonts – en webbtjänst för att inkludera diverse typsnitt i en hemsida utan att installera dessa lokalt, <https://fonts.google.com/>

med att gränssnittet färdigställdes – framför allt hämmedes inte arbetet med funktionalitet för anslutning och annan kommunikation till och från gränssnittet, vilket redogörs för i avsnitt 4.7, som krävde att ett grundläggande användargränssnitt var på plats.

Utvecklingen av hemsidan fortgick med att en förfinad prototyp av gränssnittet för en skärm skapades. Arbetet med denna prototyp lät utvecklarna precisera en vision av designen för att hemsidans element skulle samspeла med det övergripande temat. Två varianter av denna prototyp visas i figur 4.4. Bakgrundsbilderna i dessa prototyper, och i det slutliga gränssnittet, är alla skapade av projektets utvecklare. Resterande delar av användargränssnittet förfinades utifrån denna stil. Det slutliga användargränssnittet presenteras i avsnitt 5.5.1.



Figur 4.4: Två varianter av en mer utarbetad prototyp av hemsidans skärm för anslutning till spelet. 4.4a visar en mörkare version utifrån en undervattensmiljö och 4.4b visar en ljusare version utifrån en strand mot öppet hav.

## Styrning

Efter anslutning till spelet är styrningen av spelarkaraktären användarens främsta interaktion genom användargränssnittet. Under utvecklingsprocessen utvärderades totalt fyra olika styrningsgränssnitt. Dessa var ”styrning med den mobila enhetens lutning”, ”joystick-kontroller”, ”slider-kontroller” och ”två-knappsstyrning”. Att låta användaren styra genom att luta sin *smartphone* skulle ha fördelen att minimera mängden distraherande element i användargränssnittet under spelets gång. Användarens fokus riktas därmed mot domens skärm snarare än den mobila enheten. Implementationen av en lutningsbaserad styrning ställde dock säkerhetskrav på anslutningen gentemot spelets webbserver vilket inte skulle kunna ordnas under utvecklingstiden. Därmed undersöktes de övriga tre gränssnitten.

Att använda joystick-likt kontroller, det vill säga att användaren utnyttjar en spak för att ange styrriktning, upplevdes förvirrande på en konkav yta. Problemet beror i huvudsak på att det är svårt att definiera vad som egentligen är uppåt på den sfäriska skärmen då en användare tappar den logiska riktning som fås när en 2D-monitor betraktas. Den enda teknik som utvecklingsgruppen såg som lösning på detta var att ha en diskontinuitet i toppen av domen men detta upplevdes som icke-intuitivt och därmed fick kontrollerna omformas för att optimeras för domskärmen.

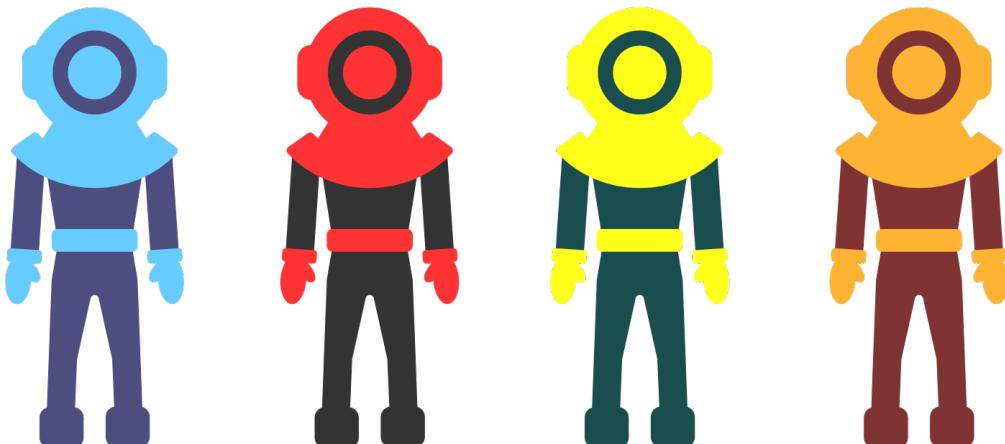
Utvecklingsarbetet fokuserades istället på de två kvarstående gränssnitten ”slider-kontroller” och ”två-knappsstyrning”. För båda dessa gäller det att spelarkaraktären har en konstant hastighet framåt och att användaren styr hur den svänger med hjälp av en *slider* eller höger–vänster-knappar. Prototyper för dessa två utformades och återfinns i figur B.1 samt figur B.2 och en användarundersökning avseende styrningen genomfördes som planerat enligt avsnitt 3.6.2. Detaljerna angående denna användarundersökning återfinns i bilaga B. I avsnitt B.2 till denna bilaga sammanställs de 63 svaren från

undersökningen. Utifrån svaren fastställdes det slutliga styrningsgränssnittet till ”två-knappsstyrning” då denna upplevdes mest intuitiv.

## Dynamiska element

På hemsidan finns ett antal element som utnyttjar dynamisk information från spelapplikationen. En redogöresle för funktionalitetet gällande hur data kommuniceras och från spelapplikationen till hemsidan återfinns i avsnitt 4.7. Det främsta exemplet på ett dynamiskt element är spelarens poäng som uppdateras på användarens mobila enhet. Allt eftersom mer föroreningar samlas upp av spelaren tilldelas denne poäng enligt redogörelse i avsnitt 4.6.8 vilket signaleras genom nätverkskommunikationen till spelarens mobila enhet. Hemsidan använder sedan JavaScript-funktionalitet för att dynamiskt uppdatera poängen som visas. Det relevanta elementet lokaliseras i DOM-trädet via sitt ID med JavaScript-funktionen `querySelector`<sup>9</sup> varpå dess textinnehåll dynamiskt kan uppdateras med den nya poängen.

Spelarens karaktär introduceras för användaren efter den har anslutit sig till spelet. Detta görs genom att en illustration skapad i Affinity Designer av dykarmodellen, som redogörs för i avsnitt 4.5.2, visas för användaren. Några olika färgvarianter av denna illustration visas i figur 4.5 och detaljer angående hur spelarkaraktären tilldelas en färgkombination presenteras i avsnitt 4.6.2. Illustrationen är i SVG-format, vilket innebär att den kan manipuleras med JavaScript som vilket annat DOM-element som helst. Liksom för spelarpoängen tar hemsidan emot information om användarens spelarkaraktärs färger från spelapplikationen och uppdaterar dynamiskt illustrationen som visas för användaren på ett analogt sätt som för spelarpoängen.



Figur 4.5: Illustrationer av spelarkaraktären i ett antal olika färgkombinationer.

Innan spelet har börjat visas informationsmeddelanden angående spelets principer och mål. Dessa visas sekventiellt i en cykel tills spelapplikationen signalerar spelets start enligt beskrivning i avsnitt 4.7. Växlingen mellan de olika meddelandena sköts genom att funktionen i fråga anropas upprepade gånger med ett visst tidsintervall tills att startsignalen mottagits. Detta hanteras i JavaScript med funktionen `setInterval`<sup>10</sup>.

I skärmen där spelets kontroller visas finns även en tidsmätare i form av en syrgastub vilket återkopplar till designelementens syfte att återspegla undervattenstemat. Syrgastuben, som visas i figur 4.6 och är skapad i Affinity Designer, har en hållighet i dess mitt vilket låter elementet på hemsidan under den synas. Således placeras syrgastuben ovanpå en stapel vilken representerar hur mycket tid

<sup>9</sup>querySelector – MDN-dokumentation, <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector>

<sup>10</sup>setInterval – MDN-dokumentation,

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval>

som kvarstår. Denna stapel animeras neråt i takt med speltiden. Denna animation sköts genom att data med den återstående procentsatsen av spelets tid regelbundet kommuniceras över nätverket från spelapplikationen. Med denna procentsats sätts dynamiskt stapelns höjd genom att modifiera dess `style.height`-attribut.



Figur 4.6: Illustration av syrgastubben med tidsindikator i mitten.

## 4.5.2 3D-modeller

Alla de 3D-modeller som används för spelapplikationen har skapats samt animerats med hjälp av modelléringsprogrammet Blender för att sedan exporteras i formatet *FBX* [11]. Ett modelléringsprogram som Blender har otaliga verktyg för att skapa modeller och i detta avsnitt avses enbart de grundläggande processer tas upp som har möjliggjort skapandet av spelmodellerna.

### Modellering

Det första steget till att skapa 3D-modeller handlade om att använda Blenders modelléringsverktyg för att forma objekt. Alla objekt representeras av ett antal *vertex*, *edges* och *faces*.

Blender erbjuder en stor mängd verktyg och de som främst har använts i detta projekt är extruderingsverktyg som utvidgar ytor och spegelverktyg som skapar symmetri. Även så kallade *modifiers* har använts genom *subdivision surfaces* för att dela upp större delar av objektet i mindre och därmed få slätare och mer detaljerade ytor [11].

### Skulptering

Ett ytterligare verktyg som rymmer inom Blender är ett digitalt skulpteringsverktyg. Detta sätt att skapa modeller ger en större kraft och flexibilitet i designen än att enbart använda modelléringsverktyg, men en kombination är den största styrkan. I Blender kan skulpteringsverktyget användas för att förändra ett objekts topologi på ett enkelt sätt och påverka strukturen på en yta, både inåt och utåt, och liknar mer en traditionell skulptering av en figur än vad modellering gör [11].

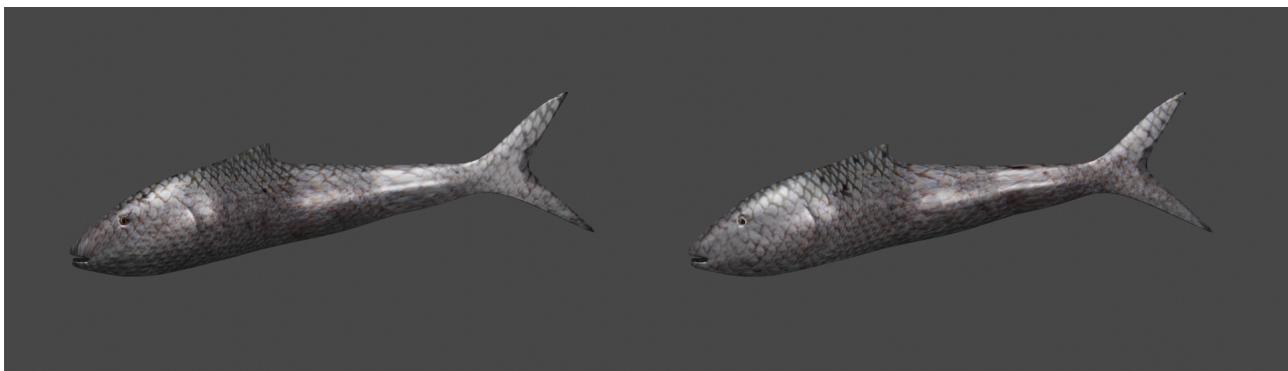
Genom skulpteringsverktyget utformades spelarkaraktären, dykaren. Genom att starta med ett enkelt plan och de modelléringsverktyg som beskrivits i avsnitt 4.5.2 skapades den huvudsakliga formen på dykaren och med hjälp av extruderingsverktyget kunde en volym skapas. Genom spegling och skulpteringsverktyget behandlades sedan denna *mesh* iterativt där detaljer skulpterades ut. Detta ger möjlighet till att skapa en mer realistisk form på exempelvis kläder vilket kan ses i figur 4.7.



Figur 4.7: En delmodell av dykarmmodellen efter att ha behandlats med hjälp av skulpteringsverktyg i Blender med standardmaterial och ljussättning.

## Texturering

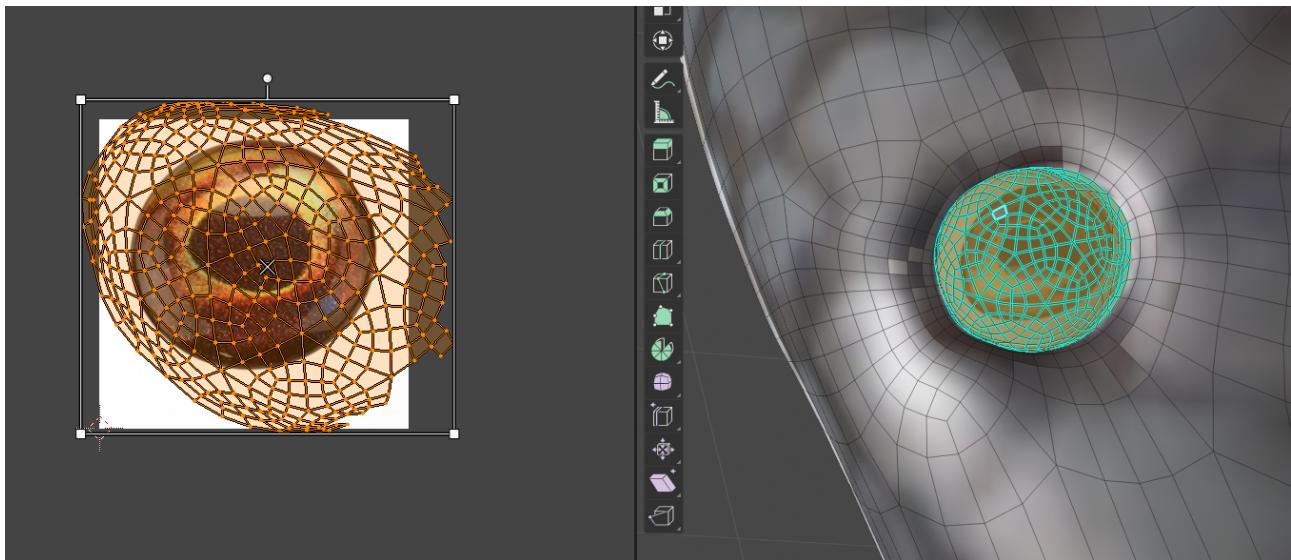
Efter att 3D-modellerna har skapats är nästa steg att lägga till materialegenskaper och texturera objekten. Detta görs med hjälp av så kallad UV-mappning där en 2D-bild projiceras på en 3D-modells yta. För att projiceringen ska bli så korrekt som möjligt behövs det skapas en UV-mapp som avgör vilken texturkoordinat som ska projiceras på vilken modellpunkt i 3D [11]. Denna process kan automatiseras i Blender men om objektens struktur inte enbart består av grundläggande geometriska objekt kan denna automatiserade projiceringen leda till artefakter, så som i den vänstra fisk-modellen i figur 4.8.



Figur 4.8: En modell av en fisk som har texturerats utan korrekt UV-unwrapping till vänster som då har synliga artefakter, speciellt kring nosen. Till höger är samma fisk men med korrekt UV-mappning. Bilden är ett utsnitt ur arbetsytan i Blender.

För att texturmappningen ska stämma överens krävs istället en manuell UV-mappning där verktyget *UV-unwrap* används. Genom att markera en yta på 3D-modellen som sedan projiceras på en 2D-yta kan användaren anpassa hur en bildtextur ska visas på modellen [11]. Ett exempel på detta kan ses i figur 4.9 där ögat på en fisk-modell justeras för att visas i centret av ögat.

Genom manuell UV-mappning kunde modeller skapas som inte lider av artefakter utan istället textureras jämnare och enligt hur speldesignen var tänkt.

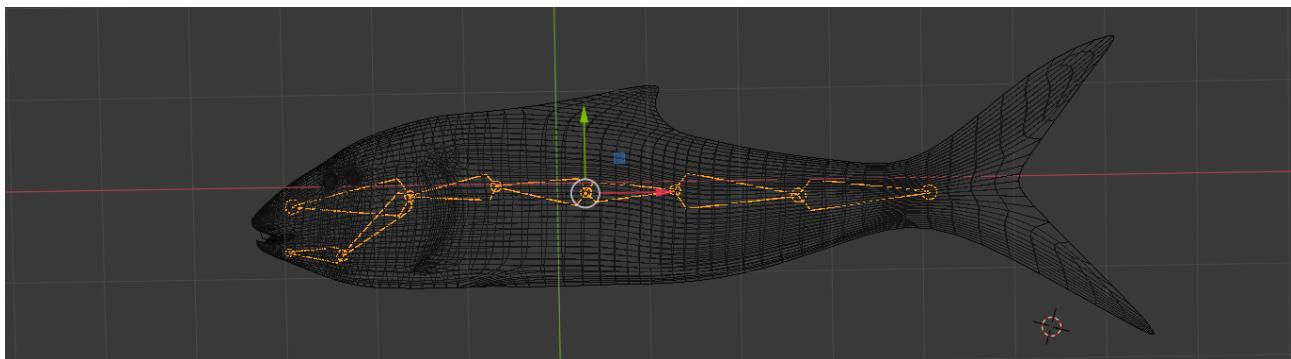


Figur 4.9: En bild från Blender där modelleringsvyn syns till höger med markerad yta som ska textureras. Till vänster syns fönstret för *UV-unwrap* där 3D-ytan har projicerats på ett 2D-plan och kan transformeras till hur användaren avser att UV-mappningen ska se ut.

### Rigging och animering

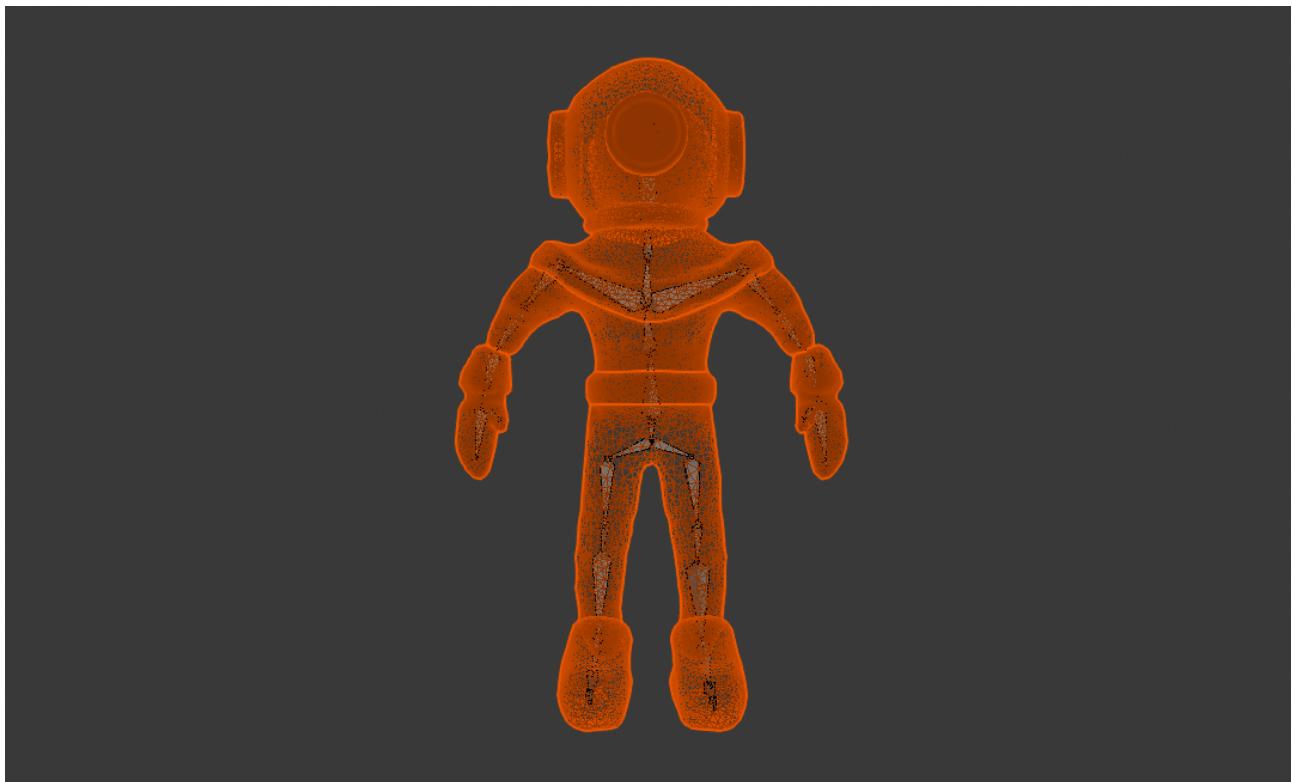
Basen för att skapa en animation för ett objekt i Blender är att rigga objektet. Detta innebär att addera ett så kallat skelett till scenen som sedan sätts som *parent* till det objekt som ska animeras. När lederna i skelettet sedan rör sig och deformeras så kommer objektet att följa med i en interpolerad bana [11].

Ett skelett kan ha varierande komplexitet och animeringarna kan göras extremt detaljerade och addera extra funktionalitet men då detta projekt inte kräver att animeringar betraktas på nära håll utan enbart avser förhöja spelarupplevelsen har enbart grundläggande animationer adderats till objekten. Exempel på skelett som adderas till objekt för animering är fiskmodellen i figur 4.10 och för dykarmodellen i figur 4.11.



Figur 4.10: En illustration över hur ett skelett har skapats och adderas inuti en fisk-mesh. Varje ben och led i skelettet kan modifieras och rör sig med hjälp av fördefinierade restriktioner. Bilden är ett utsnitt ur arbetsytan i Blender.

När skelettet har blivit adderat till modellen kan animeringsarbetet påbörjas genom att definiera *keyframes* för karaktären. Genom att transformera och på olika sätt modifiera poseringen för objektet under influens av skelettet genom att låsa objektets position för vid specifika tidpunkter. En rörelsebana skapas sedan av Blender som sköter interpoleringen mellan dessa *keyframes* [11]. För de simplare animationer som detta projekt krävde så sattes en *keyframe* till de flesta ben i skelettet samtidigt utan att ta hänsyn till små individuella förändringar.



Figur 4.11: Dykarmodell med ett skelett adderat som möjliggör för animering av karaktären. Bilden är ett utsnitt ur arbetsytan i Blender.

## Export

När 3D-modellerna var färdigmodellerade, texturerade och animerade exporterades de till formatet FBX som möjliggör att paketera ihop all information om modellen på ett kompakt sätt vilket sedan kan läsas in med hjälp av biblioteket Assimp i spelapplikationen [11].

## 4.6 Spelapplikation

Spelapplikationen som har implementerats har utformats med kravet att spelet ska klara av 50 simultana spelare, enligt avsnitt 4.1.1, men källkodens interna begränsning är 102 spelare vilket motsvarar en fullsatt dom. Därefter tillåts inga fler spelare ansluta till spelet.

Föroringen skapas dynamiskt beroende på hur många spelare som är anslutna till spelet. Varje gång ett givet tidsintervall passerar gör spellogiken en bedömning över hur många föroringar som ska dyka upp på spelplanen.

Maximala antalet simultana föroringar har begränsats till 300 i källkoden. Detta begrundas ej på empirisk data utan en överenskommelse att fler än 300 föroringar skulle bli för rörligt på domytan. Detta innebär att ett värsta fall prestandamässigt vore om 100 spelare tillåter 300 föroringar att existera samtidigt vilket ger totalt 400 objekt som uppdateras varje bildruta.

### 4.6.1 Grafikprogrammering

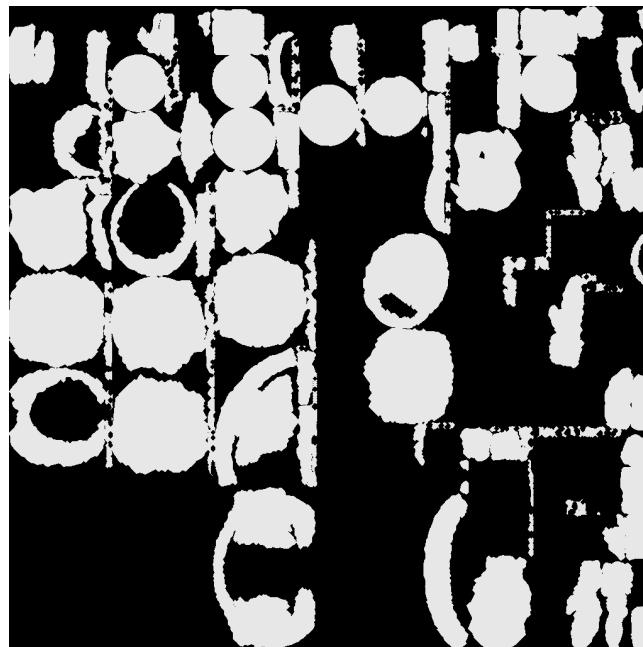
Spelapplikationens grafiska programmering hanteras av *shader*-program vilka enligt beskrivning i avsnitt 4.2 är skrivna i GLSL. Ljussättningen i spelet hanteras av en enkel Phong-reflektionsmodell

[12] med en enkel, riktad ljuskälla. Med olika *shader*-program för olika typer av spelobjekt kan dessa tilldelas olika reflektiva egenskaper – exempelvis får en läskburk mer spekulära effekter än en plastskräp, och dykarmodellens metalliska delar får mer glans än dess kroppsdräkt.

## 4.6.2 Spelarkaktären

Varje spelare har en egen karaktär i form av en dykare vars 3D-modell redogjordes för under avsnitt 4.5.2. För att upp till 50 spelarkaktärer ska kunna särskiljas tilldelas varje karaktär en kombination av primärfärg och sekundärfärg – dessa färger syns på dykarmodellens kroppsdräkt respektive handskar, hjälm och bälte. Dessa kroppsdelar särskiljs av spelapplikationens *shader*-program med en maskeringstextur som visas i figur 4.12. Denna textur skapades genom att tilldela dykarmodellens kroppsdelar svart respektive vit diffus färg och sedan baka ut modellens diffusa textur i Blender. Dykarmoden är därmed UV-mappad till denna textur på sådant vis att dess kroppsdräkt motsvaras av svarta pixelvärdet medan handskar, hjälm och bälte motsvaras av vita pixelvärdet. Färgen för en given pixel erhålls således av (4.1), där  $C_{\text{final}}$  är den slutliga pixelfärgen,  $T_{\text{mask}}$  är det motsvarande pixelvärdet från masktexturen,  $C_{\text{primary}}$  är primärfärgen och  $C_{\text{secondary}}$  är sekundärfärgen. Samtliga färgvärdet representeras i RGB-format som tredimensionella vektorer. Maskeringstexturen används också för att tilldela de olika kroppsdelarna separata reflektiva egenskaper i samma *shader*-program.

$$C_{\text{final}} = T_{\text{mask}} \cdot C_{\text{secondary}} + (1 - T_{\text{mask}}) \cdot C_{\text{primary}} \quad (4.1)$$

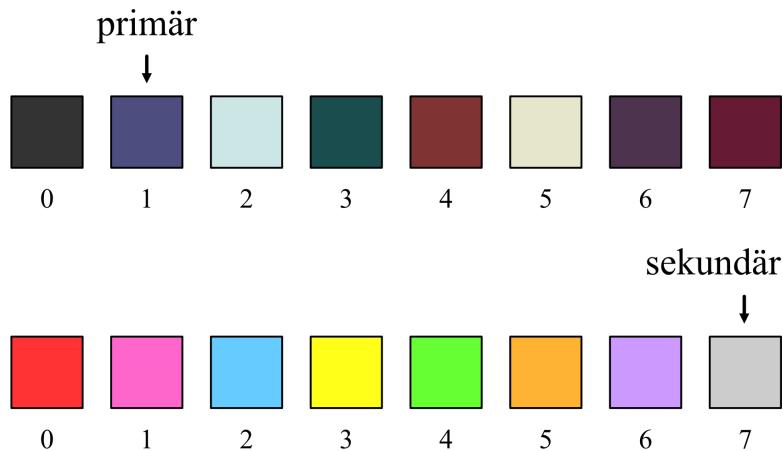


Figur 4.12: Den maskeringstextur som utnyttjas för att särskilja dykarmodellens kroppsdelar, skapad med hjälp av Blender.

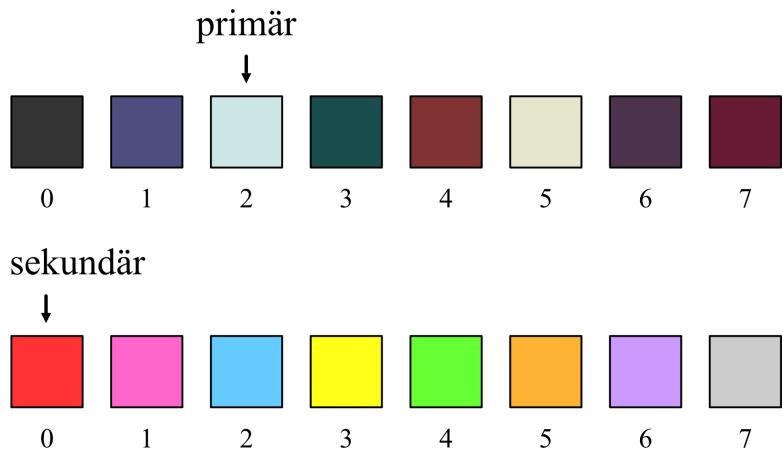
### Tilldelning av spelarfärger

Totalt har systemet som tilldelar varje spelare sina färger 8 primär- och 8 sekundärfärger vilket ger  $8^2 = 64$  unika kombinationer. Färgerna lagras i två sekvensiella minnesblock och färgernas ordning i dessa block slumpas vid spelapplikationens start. Systemet som tilldelar färgerna bibehåller två heltal vilka anger positionen av nästa primär- respektive sekundärfärg. När en ny färg hämtats inkrementeras positionen för sekundärfärgen. Detta fortsätter tills den sista sekundärfärgen har använts varpå den

positionen nollställs och primärfärgen räknas upp. Detta liknas med andra ord vid att räkna upp i ett talsystem med antalet sekundärfärger som bas. Denna princip visas i figurerna 4.13 och 4.14.



Figur 4.13: Illustration över hur nästa färgkombination ges av positionerna 1 och 7 för primär- respektive sekundärfärg.



Figur 4.14: Från figur 4.13 inkrementeras sekundärpositionen förbi slutet av dess lista och nollställs. Istället inkrementeras primärpositionen vilket ger nästa färgkombination av positionerna 2 och 0 för primär- respektive sekundärfärg.

### 4.6.3 Spelsimulering och dataorientering

Inom tidsintervallet som beskrivet i avsnitt 4.1.2 sker spelsimuleringen som för spelets skede vidare. Även om domens datorer är kraftfulla görs utvecklingen på utvecklingsgruppens personliga datorer med varierande prestanda. För att försäkra likvärdig spelupplevelse måste därför spelsimulationen vara oberoende av en dators prestanda. Därför kopplas till exempel positionsuppdateringen av en spelare till hur mycket den bör röra sig per sekund, inte hur mycket den bör röra sig per bildruta.

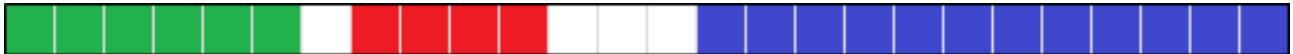
Simuleringen sker i tre steg. Först uppdateras positionen av varje objekt i scenen med hjälp av styrningsdata och hastighet. När den nya positionen är beräknad kontrolleras om någon spelare har försökt ta sig ut ur spelområdet varpå dess karaktär flyttas till den närmsta tillåtna positionen i spelområdet. När all positionsuppdatering är klar kontrolleras kollisioner mellan spelare och föröreningar som beskrivs nedan. Efter detta återstår bara att rendera nästa bildruta.

## Dataorienterad spelsimulering

Även om en objektorienterad systemarkitektur lämpar sig väldigt bra för spels objektbaserade natur finns ett problem med polymorfism. Objektorienterad programmering resulterar i väldigt logiska strukturer i källkoden men dålig datalokalitet i minnet. För att illustrera detta problem presenteras ett exempel med spelet:

I situationen med maximalt antal spelare beskrivet i avsnitt 4.6 kan totalt 100 spelare och 300 föreningar existera samtidigt. Ponera att dessa lagras polymorfiskt i en behållare med totalt 400 element. Innan nästa bindruta renderas måste kollisionsdetektering utföras mellan varje unikt par av spelare och föreningar vilket ger  $\binom{400}{2} = 79800$  kombinationer och totala läsningar i listan. Detta är inte ett problem för en modern dator men konsekvensen av polymorfism är att alla dess objekt ligger utspritt i datorns minne vilket medför *cachemissar* [13].

För att undvika detta problem har dataorienterad design använts vid spelsimuleringen för att försäkra att systemet klarar prestandakraven beskrivet i avsnitt 4.1.2. *Cachens* begränsade storlek har inte alltid plats för att hålla redo på 400 unika element i minnet. Istället för att använda polymorfism lagras istället varje instans av de relevanta subklasserna konkret i en `std::vector`. Denna behållare garanterar att dess minne allokeras angränsande i minnet [14]. Informellt betyder det att alla objekt i `std::vector` ”ligger på rad” i minnet. Detta minskar cachemissar eftersom endast pekaren till första elementet i behållaren behöver sparas i cachen vilket kraftigt reducerar antalet minnesaddresser att hålla redo på. Skillnaden i minnesallokering illustreras i figur 4.15 och 4.16. Konsekvensen av denna dataorienterade lösning är att programmet har flera behållare för spelets unika objekt vilket kan upplevas rörigt i källkoden men prestandaökningen uppväger detta.



Figur 4.15: Modell av hur unika objekt som lagras konkret i tre olika `std::vector` kan se ut i datorns minne. De färgade områdena representerar objekten, ofärgade områdena fritt minne.



Figur 4.16: Modell av hur minnesallokering med polymorfisk struktur kan se ut i datorns minne. De färgade områdena representerar objekten, ofärgade områdena fritt minne.

### 4.6.4 Föreningar i objektpooler

I spelet skapas och förstörs föreningar dynamiskt skalat till antalet spelare. I en situation med stort antal spelare innebär detta många minnesallokeringar vilket är en relativt dyr operation som kan leda till försämrad prestanda. För att lösa detta är klassen `CollectiblePool`, illustrerad i figur 4.3 och beskriven i avsnitt 4.2.3, implementerad med design-mönstret *object pool* [15]. Med detta mönster skapas en `std::vector` med ett stort antal `Collectible`-objekt innan spelet börjar. Istället för att kontinuerligt allokerar nytt minne för föreningar hämtas istället en referens till en förening i förskapade behållaren. Föreningens position uppdateras och markeras som aktiverat. När föreningar blir bortrensad markeras den som inaktiverad och kan inte längre renderas eller integreras med.

Objektpoolen löser minnesallokeringproblemet men i vissa situationer tappas prestanda ändå. När en ny förening ska hämtas måste behållaren sökas igenom för ett inaktiverat element som kan återanvändas vilket har tidskomplexiteten  $O(n)$ , där  $n$  är antalet element i behållaren. Detta problem löstes genom att kombinera objektpoolen med design-mönstret *free list* [15]. Med detta mönster behövs lite

extra bokföring då varje objekt i poolen även länkas ihop med pekare som en enkellänkad lista. Själva poolen i sig har alltid en pekare till första inaktiverade elementet redo för återanvändning. När detta objekt aktiveras flyttas pekaren för det första inaktiverade objektet till nästa nod i listan. När ett objekt inaktiveras trådas den tillbaka på första plats i listan. Detta gör att operationen för att aktivera en ny förening i spelet får tidskomplexiteten  $O(1)$  vilket försäkrar god prestanda i alla situationer som kan uppstå.

### 4.6.5 Nodsynkronisering

I domen kontrolleras projektionen av ett antal olika datorer. Dessa datorer kör alla en egen instans av spelet för att rendera och projicera på sin beskärda del av domen. Detta ger två möjligheter till hur spelsimuleringen görs i spelet. Antingen får alla datorer styrningsinformation från spelarna i domen och gör spelsimuleringen oberoende av de andra datorerna i domen. Detta kan dock medföra desynkronisering i spelet mellan datorerna som följd av till exempel avrundningsfel i de många flytalsoperationer som ingår i spelsimuleringen.

Den andra möjligheten och den implementerade lösningen i spelet är att enbart en huvudator genomför spelsimuleringen för att sedan synkronisera det nya speltillståndet till resten av domens datorer. Detta försäkrar total synkronisation i spelet mellan datorerna men ökar systemkomplexiteten med mer trådlös kommunikation.

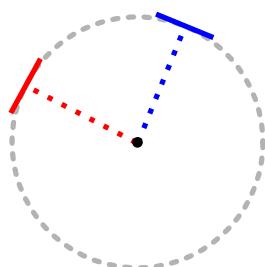
### 4.6.6 Modellhantering i spelet

I avsnitt 4.3.5 beskrivs hur datastrukturen som biblioteket Assimp lagrar 3D-modeller, texturer och animationer i ser ut. Denna trädstruktur fungerar inte att skicka till OpenGL, därmed skapades klasserna `Mesh` och `Model`, som illustreras i figur 4.3. `Mesh` extraherar alla vertiser, indexdata och texturer ur trädstrukturen och skapar array- och bufferobjekt för GPU. `Model` hanterar en behållare med instanser av `Mesh` då en 3D-modell kan bestå av flera *meshes*. `Model` implementerar även funktionalitet för att rendera geometri och texturer.

Som nämnt ovan i avsnitt 4.2.3 lagras alla 3D-modeller i klassen `ModelManager`. Denna klass implementerar enbart funktionalitet för att lämna ut en referens till önskad 3D-modell som används när nya spelare eller föreningar skapas.

### 4.6.7 Position i domen

Domens form innebär specielllösningar för spelobjektens position. Eftersom objekten rör sig på ytan av en sfär lämpar sig inte kartesiska koordinater särskilt väl för att beskriva positionen. Istället translateras objektet till sfärens yta och positionen representeras sedan av en rotation. På så vis är en av objektets sidor alltid vänd mot origo och positionen är alltid på sfären, se figur 4.17.



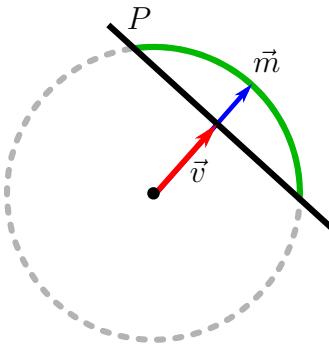
Figur 4.17: Visualisering av objekts position i domen. Figuren är skapad i Inkscape.

För att representera dessa rotationer används kvaternioner. En kvaternion är ett slags fyrdimensionellt komplex tal, som i enhetatform kan användas för att beskriva en rotation i tre dimensioner, på samma sätt som ett vanligt komplex tal  $z : |z| = 1$  kan beskriva en rotation i planet. Att använda kvaternioner istället för Euler-vinklar ger både en kompakt kod men också fördelar när det kommer till interpolation mellan två vinklar som gör att spelstyrningen upplevs jämnare [12]. Två rotationer representerade som kvaternioner kan enkelt kombineras till en enligt (4.2)

$$q_{\text{comp}} = q_2 q_1 \quad (4.2)$$

där  $q_{\text{comp}}$  är rotationen  $q_1$  följd av  $q_2$  [16]. Det här sambandet utnyttjas för spelobjektens rörelse; spelarens förflyttning multipliceras med den gamla positionen för att få den nya. Kvaternionerna representeras med GLM-klassen `quat` som har grundläggande operationer samt en del andra användbara funktioner definierade.

För att säkerställa att spelare inte simmar ur bild måste ett tillåtet område på sfären definieras. Eftersom spelarens avstånd till origo är konstant och positionen endast består av rotationer kan beräkningarna göras utan att blanda in domens radie. Alla beräkningar utförs istället utifrån enhetssfären. Området definieras med en enhetsvektor,  $\vec{m}$ , som pekar från origo till mitten av det tillåtna området samt ett mot  $\vec{m}$  ortogonalt plan,  $P$ , på ett lämpligt avstånd från origo. Det tillåtna området definieras sedan som den del av sfären som ligger på den sidan av  $P$  som har ytan i samma riktning som  $\vec{m}$  pekar, se figur 4.18. Planet representeras i beräkningarna av en vektor  $\vec{v}$  från origo till den närmsta punkten på planet;  $\vec{v}$  är alltså parallell med  $\vec{m}$ .



Figur 4.18: Hur det tillåtna området på sfären definieras. Det tillåtna området är markerat i grön färg. Figuren är skapad i Inkscape.

Först utförs ett test för att avgöra om den nuvarande positionen  $q$  är tillåten. Det första steget är att utifrån kvaternionen ta fram en enhetsvektor  $\vec{p}$  som representerar objektets position genom att applicera rotationen  $q$  på en enhetsvektor som pekar i den riktningen som translationen utförts i. Positionen jämfört med toppen på  $\vec{v}$  blir då  $\vec{w} = \vec{p} - \vec{v}$ . Då kan  $\vec{w}$  projiceras på  $\vec{m}$  enligt (4.3) ur [17]

$$\vec{w}_{\parallel \vec{m}} = \frac{\vec{w} \cdot \vec{m}}{\|\vec{m}\|^2} \vec{m} = \{\|\vec{m}\| = 1\} = (\vec{w} \cdot \vec{m}) \vec{m} \quad (4.3)$$

och då gäller att  $w_{\parallel \vec{m}}$  pekar i samma riktning som  $\vec{m}$  för positiva värden på  $\vec{w} \cdot \vec{m}$ , och annars i motsatt riktning. Alltså är positionen tillåten om och endast om  $\vec{w} \cdot \vec{m}$  är positiv. Om positionen är tillåten görs inget mer. Om objektet har rört sig ut från området tvingas det tillbaka till närmsta tillåtna position. Avståndet mellan toppen på  $\vec{v}$  och det tillåtna områdets kant beräknas i (4.4) enligt Pythagoras sats

$$d = \sqrt{1 - \|\vec{v}\|^2} \quad (4.4)$$

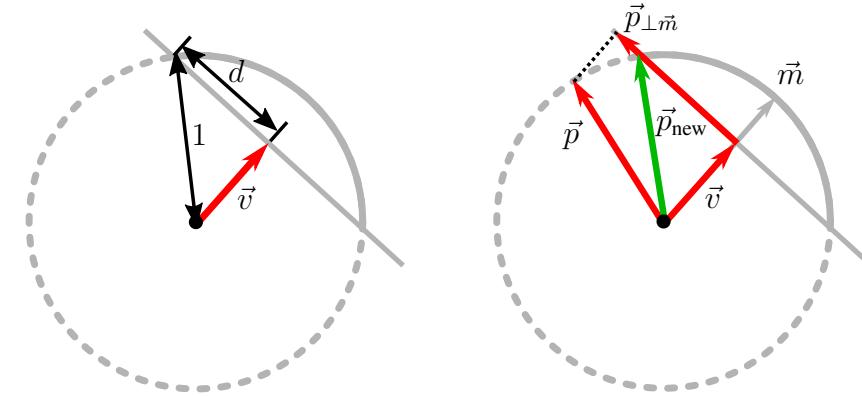
då radien på sfären är 1, se figur 4.19. Den korrigrade positionen kommer hamna på planet  $P$ , eftersom områdets kant definieras av skärningen med  $P$ . Genom att projicera  $p$  på  $P$ , det vill säga ta

den delen som är ortogonal mot  $\vec{m}$ , enligt (4.5) ur [17]

$$\vec{p}_{\perp \vec{m}} = \vec{p} - \frac{\vec{p} \cdot \vec{m}}{\|\vec{m}\|^2} \vec{m} \quad (4.5)$$

fås riktningen från toppen av  $\vec{v}$  till den nya positionen. Genom att studera figur 4.19 blir det tydligt att  $\vec{p}_{\perp \vec{m}}$  endast behöver skalas om till längden  $d$  för att med  $\vec{v}$  ge den nya positionsvektorn enligt (4.6)

$$\vec{p}_{\text{new}} = d \frac{\vec{p}_{\perp \vec{m}}}{\|\vec{p}_{\perp \vec{m}}\|} + \vec{v}. \quad (4.6)$$



Figur 4.19: Hjälpvektorer vid beräkninget av korrigerad position. Figuren är skapad i Inkscape.

Slutligen konverteras  $\vec{p}_{\text{new}}$  till en kvaternion med hjälp av GLM-funktionen `quatLookAt`, som skapar en kvaternion av en framåt-vektor och en upp-vektor. I vårt fall används den icke-korrigerade positionens upp-vektor vilket ger ett tillräckligt bra resultat eforsom positionen aldrig är långt utanför det tillåtna området.

#### 4.6.8 Kollisionsdetektering och poängutdelning

Eftersom varje entitets position i spelet definieras med kvaternioner ökar detta komplexiteten för hur kollisioner detekteras. Istället för att beräkna det euklidiska avståndet mellan två entiteter måste vinkeln mellan dessa entiteters kvaternioner beräknas istället.

För två givna kvaternioner  $q_1$  och  $q_2$  måste det finns en kvaternion  $q_t$  som beskriver hur kvaternionen  $q_1$  kan transformeras till  $q_2$ . Detta formuleras enligt (4.7).

$$q_t q_1 = q_2 \Leftrightarrow q_t = q_1^{-1} q_2 \quad (4.7)$$

När  $q_t$  är beräknad kontrolleras om vinkeln som denna kvaternion representerar är liten nog för att räknas som en kollision mellan  $q_1$  och  $q_2$ .

Denna beräkning appliceras i en *brute force*-metod för kollisionsdetektering. Detta innebär att varje unik kombination av spelare och förorening kontrolleras för kollisioner vilket resulterar i en tidskomplexitet på  $O(nm)$ , där  $n$  är antalet spelare och  $m$  antalet föroreningar. Denna tidskomplexitet är acceptabelt med motivering av det relativt låga maximala antalet unika par som kan existera samtidigt enligt gränserna beskrivna i avsnitt 4.6 och de kraftfulla datorerna i domen.

När en spelare kolliderar med en förorening tas föroreningen bort från spelplanen och spelaren tilldelas tio poäng vilket uppdateras på spelarens telefon.

### 4.6.9 Ekvirektangulär mappning

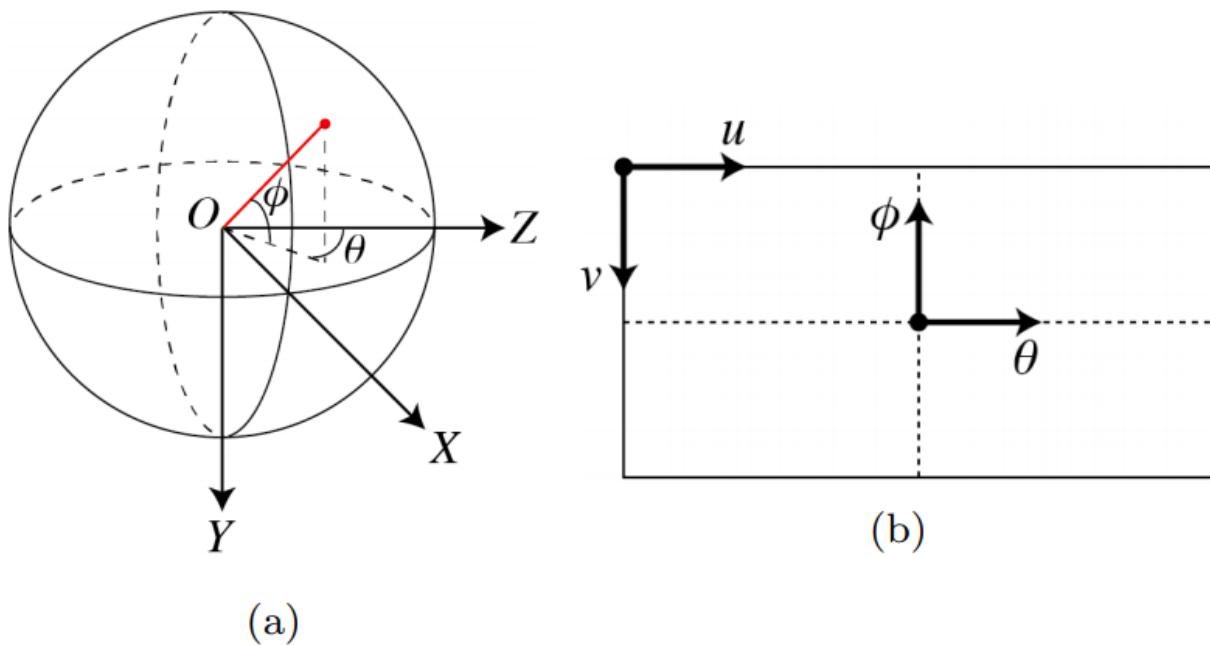
Att skapa innehåll för en domskärm ställer krav på att mappa texturer och koordinater på ett sådant sätt att de inte upplevs förvrängda när de syns på den cirkelformade skärmen. Eftersom verktyget SGCT sköter hanteringen av att fördela ut vad domeprojektorerna ska projicera behövs en mappning som gör att en sfärisk vy kan representeras på ett korrekt sätt på en 2D-monitor.

Detta kan göras med hjälp av olika metoder som alla har olika fördelar och nackdelar. I detta projekt har ekvirektangulär mappning använts, vilket även är en väl använd metod för realtidsapplikationer. En ekvirektangulär mappning är en cylindrisk ekvidistant projektion som använder sig av de sfäriska koordinaterna och mappar longitud direkt på den horisontella koordinaten och latitud till den vertikala [12]. Transformationen för den ekvirektangulära mappningen mellan longitud och latitud till motsvarande ekvirektangulära texturkoordinater sker genom ekvation (4.8),

$$u = \frac{w}{2\pi}(\theta + \pi) \quad (-\pi \leq \theta < \pi) \quad (4.8a)$$

$$v = \frac{h}{\pi}(-\phi + \pi) \quad \left(-\frac{\pi}{2} \leq \phi < \frac{\pi}{2}\right) \quad (4.8b)$$

där  $w$  motsvarar bildens bredd i pixlar och  $h$  bildens höjd i pixlar [18]. Denna transformation visualiseras i figur 4.20. Detta samband har givetvis också en invers för att kunna transformera även åt andra hållet, givet ekvirektangulära texturkoordinater.

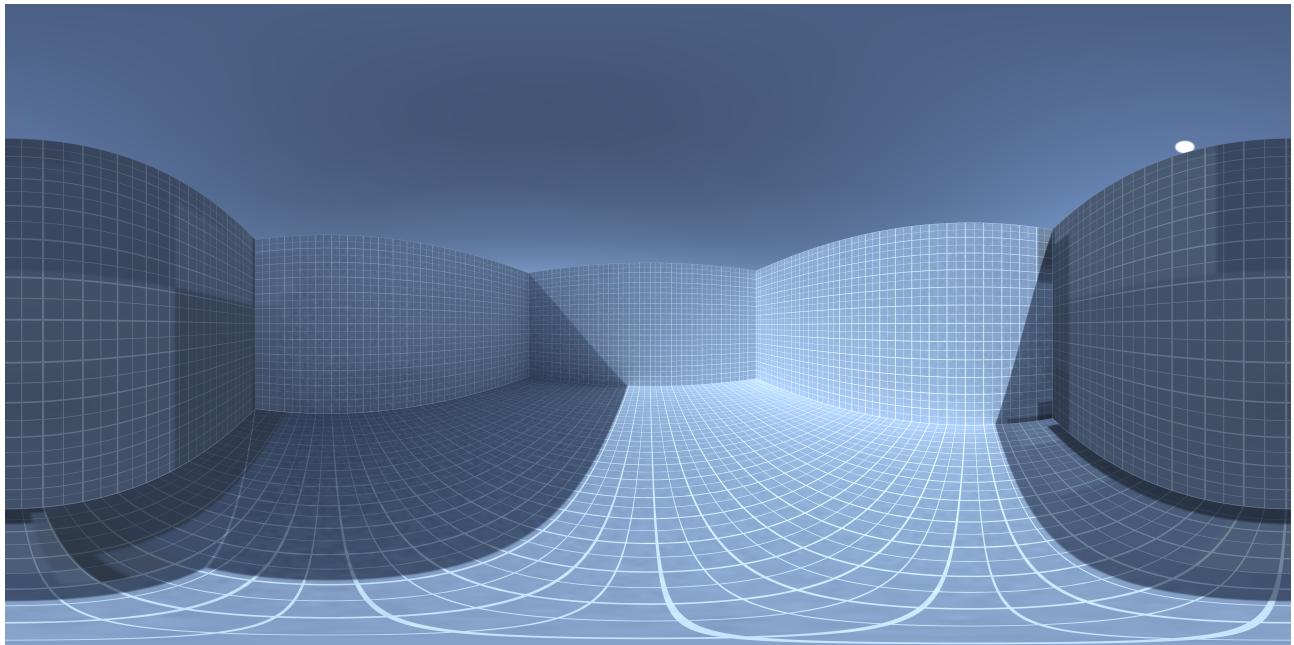


Figur 4.20: Övergripande illustration över ekvirektangulär mappning mellan longitud och latitud  $\theta, \phi$  och den horisontella och vertikala texturkoordinaterna,  $u, v$ . I (a) visas de sfäriska koordinaterna och i (b) de ekvirektangulära texturkoordinaterna. Bild hämtad från [18].

Eftersom formen på domskärmen är sfärisk behöver mappningen för domeskärmen ske genom inversen av ekvation (4.8). Det vill säga, en textur behövs skapas som sedan kan projiceras sfäriskt på domskärmen.

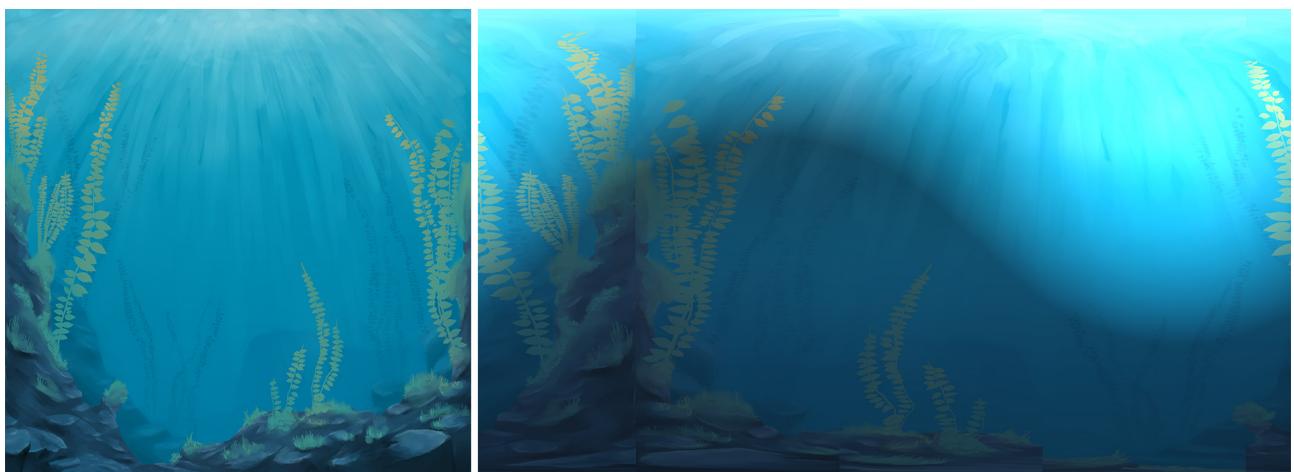
I spelapplikationen representeras bakgrundsobjektet av klassen `BackgroundObject` som adderar en halvsfär som alltid renderas bakom alla andra objekt i scenen. Denna halvsfär är positionerad så att den är sedd från insidan från kamerans riktning och därmed skapas den ekvirektangulära mappningen på korrekt sätt på domskärmen och ger spelare en känsla av att vara omslutna av domskärmen.

Den ekvirektangulära mappningen för spelet består av en 2D-textur som skapas i spelmotorn Unity där en *shader* skapats som utför den ekvirektangulära mappningen enligt (4.8). Genom att placera ut en scen med objekt och en kamera kan ett script placeras på kameran som möjliggör det för användaren att fånga en bild av scenen efter den ekvirektangulära mappningen. För att kontrollera att mappningen utförs korrekt adderades ett testobjekt bestående av ett rektangulärt rum till scenen och fångades med hjälp av kamerans script. Resultatet av testscenen kan ses i figur 4.21.



Figur 4.21: En testscen skapad i Unity med hjälp av en ekvirektangulärmappning och där kameran fångar en 360° sfärisk panorama av scenen.

För den resulterande spelapplikationen placeras kameran inuti en sfär som har texturmappats med den färdiga bakgrundsbilden och *backface-culling*, som enbart renderar vertiser som är synliga i scenen, stängs av för att få texturen att mappas på insidan av sfären.



Figur 4.22: Till vänster syns ett av bakgrundskoncepten skapade i projektet och till höger är resultatet efter att ha texturerat denna bild på insidan av en sfär i Unity. Denna sfär har sedan projicerats med hjälp av ekvirektangulär mappning och sparats som en 360° sfärisk panorama.

## 4.7 Nätverkskommunikation

Kommunikationen mellan klient och spel är en grundläggande del av projektet och är uppbyggd av plattformen Node.js och C++-biblioteket Libwebsockets. För kommunikationen mellan webbservern och huvuddatorn i domen används JavaScript-biblioteket Websockets. Kommunikation med servern på spelsidan sker med biblioteket Libwebsockets, se figur 4.2 under avsnitt 4.2.2. Under projektet gång upptäcktes det att det krävdes en säker anslutning för att kunna använda den mobila enhetens accelerometer, se avsnitt 4.5.1. Möjligheten att hantera hemsidan genom det krypterade protokollet *HTTPS* med egna certifikat undersöktes. Det visade sig vara svårt att stödja *HTTPS* eftersom det skulle kräva en omstrukturering av hela serverns kodbas och kräva mycket tid. Det valdes då att använda *HTTP* som protokoll och ett annat system för att hantera styrningen, vilket diskuterades i avsnitt 4.5.1.

För att ansluta till servern kopplar användaren upp sig via hemsidan. Användarens namn skickas till servern och blir tilldelad ett unikt identifikationsnummer. Detta nummer skickas sedan, tillsammans med användarens namn, till spelet där en karaktär skapas. När användaren styr sin karaktär skickas styrningsinformation från hemsidan via servern till spelet. Användaren styr med två knappar på hemsidan. Genom att trycka på en av knapparna skickas information till spelapplikationen om vilken riktning karaktären ska roteras mot.

Spelet utnyttjar en två-vägskommunikation, vilket betyder att information skickas och mottas både på hemsidan och i spelapplikationen. Detta möjliggör att användaren får återkoppling från spelet på hemsidan. Användarens karaktär blir tilldelad en unik färgkombination i spelet som via servern skickas till hemsidan. Färgerna visas sedan på användarens mobila enhet, både före spelets start och under spelets gång, för att underlätta lokalisering av karaktär på domskärmen.

Under spelets gång uppdateras användarens poäng vid uppsamling av föroreningar. I spelapplikationen parar poängen ihop med spelarnas identifikationsnummer, vilket innebär att poänginformationen kan skickas från spelapplikationen till hemsidan för korrekt användare vid varje uppdaterad poäng. Detta åstadkoms genom att servern jämför det medföljda identifikationsnumret med användarens unika identifikationsnummer. Om dessa stämmer överens med varandra skickas poängen därefter vidare till användarens personliga hemsida. Användaren kan på så sätt se sina poäng uppdateras på sin mobila enhet enligt beskrivningen av gränssnittet i avsnitt 4.5.1.

För att kunna avgöra den mest optimala lösningen vid händelse av frånkopplad användare testades två olika implementationer. För båda lösningarna gällde att spelarens karaktär avaktiverades, snarare än togs bort helt, så fort användarens anslutning bröts. En avaktiverad karaktär slutade renderas på domskärmen. I den första lösningen möjliggjordes återanslutning för användare med hjälp av *cookies*. När användaren anslöt till hemsidan skapades då en personlig *cookie* som sammankopplade webbläsaren med spelarens unika ID. Om användaren sedan laddade om hemsidan på sin mobila enhet, eller på något annat vis tappade anslutningen till spelet och försökte återansluta via hemsidan, skickades användarens unika identifikationsnummer till spelapplikationen. Med hjälp av detta nummer avaktiverades dykarkaraktären. Användaren hamnade sedan på en ny sida där denne hade möjligheten att återansluta till spelet. Vid återanslutning aktiverades användarens tidigare tilldelade spelare och användaren kunde fortsätta spela med samma namn. I den andra lösningen, om användaren blev frånkopplad från spelet, skickades information om detta till spelapplikationen. Användarens dykarkaraktär blev därmed avaktiverad och försvann från domskärmen. Användaren hade sedan möjligheten att återansluta till spelet med en ny spelare via samma process som första anslutningen. Den senare lösningen, utan användning av sparad data, valdes att implementeras i projektet eftersom den beslutas vara mer tillförlitlig än lösningen med *cookies*, som krävde att spelapplikationen omstrukturerades för att söka efter potentiell sparad data vid bortfall av spelare.

Ytterligare en implementerad funktion i spelet är starttid och sluttid. Spelet startas i spelapplikationen med ett knapptryck från ansvarig personal som sköter kopplingen mellan applikationen och domskärmarna och avslutas efter en viss given tid. En tidsmätare på hemsidan tar emot information från spelapplikationen och visar användaren hur mycket tid som är kvar av spelet.

Ett system finns för att kunna avgöra vad för typ av data som skickas mellan servrarna. Informationen som skickas blir tilldelad ett unikt ID beroende på datans ändamål. Strukturering av kommunikation från webbserver till spelapplikation syns i tabell 4.1. Kommunikationen från spelapplikationen till webbservern syns i tabell 4.2.

Tabell 4.1: Strukturering av kommunikation från webbserver till spelapplikation

| Aktivitet             | Meddelande |      |      |
|-----------------------|------------|------|------|
| Skapa spelare         | N          | Namn | ID   |
| Hämta färgkombination | I          | ID   |      |
| Skicka styrningsdata  | C          | ID   | Data |
| Avaktivera spelare    | D          | ID   |      |

Tabell 4.2: Strukturering av kommunikation från spelapplikation till webbserver

| Aktivitet             | Meddelande |            |    |
|-----------------------|------------|------------|----|
| Skicka primärfärg     | A          | RGB-färg   | ID |
| Skicka sekundärfärg   | B          | RGB-färg   | ID |
| Skicka poäng          | P          | Poäng      | ID |
| Skicka tid            | T          | Tid        |    |
| Skicka start-/sluttid | U          | Start/slut |    |

# Kapitel 5

## Resultat

Undersökningsarbetet i att bestämma hur ett immersivt och skalbart system för en domskärm utformas optimalt resulterade i ett fleranvändarspel för upp till 50 användare som har använts för att kunna dra slutsatser om de frågeställningar som definierades i avsnitt 1.2. I detta spel tillåts användarna ansluta till spelapplikationen via en hemsida och blir där tilldelad en unik dykarkarakter på domskärmen. Alla spelare har sedan en begränsad mängd tid att få ihop poäng genom att samla in föroreningar som dyker upp på skärmen. Styrning av den personliga karaktären sker via två knappar på användarens mobila enhet. När spelet startar dyker föroreningar upp kontinuerligt som spelaren har som mål att samla in.

### 5.1 Grafik och modeller

Detta avsnitt redogör för den grafiska design och 3D-modellering som utvecklingsarbetet resulterade i. Att bibehålla en enhetlig design över hela systemet var viktigt i utvecklingsgruppens arbete för att på bästa sätt undersöka och besvara de frågeställningar som definierades i avsnitt 1.2 samt för att erhålla en produkt som var förenlig med utvecklingsgruppens initiala vision.

#### 5.1.1 3D-modeller i Blender

I figur 5.1 visas de olika plastmodellerna som skapades för att bli insamlade av spelarna. Spelarmodellen som skapades med hjälp av de modellerings- och skulpteringstekniker som har beskrivits i avsnitt 4.5.2 kan ses i figur 5.2.

För att testa hur de definierade materialegenskaperna såg ut visuellt gjordes även ett antal testrenderingar med ljussättning som skulle likna det i en undervattensmiljö. Ett renderingsresultat med ett adderat bakgrundsplan texturerat med ett av bakgrundsconcepten kan ses i figur 5.3.



Figur 5.1: Alla de plastmodeller som skapats och texturerats i Blender för att samla i spelet.



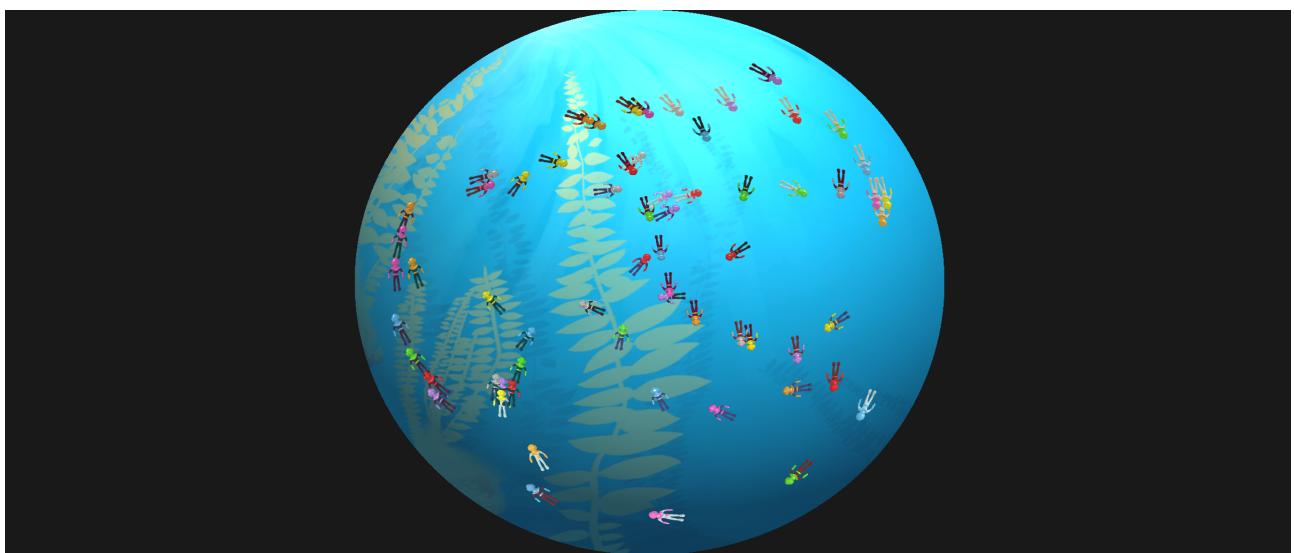
Figur 5.2: Den färdiga dykarmodellen som har skapats i Blender visas med grundläggande ljussättning och material till vänster, som wireframe-modell i mitten och med texturering till höger.



Figur 5.3: Dykarmodell med texturer och adderad ljussättning samt ett bakgrundsplan texturerat med ett av de framtagna bakgrundskonceptet som sedan har renderats i Blender med renderaren *Cycles*.

### 5.1.2 Spelarmodellens färger

Spelarens karaktär, dykaren, som presenterades i föregående avsnitt tilldelades en unik kombination av färger vid anslutning. Dessa användes för att färglägga modellens kropp respektive hjälm, handskar och bälte. Systemet som utvecklades för att hantera färgvalet utgick från två separata listor à 8 färger och itererade genom varje kombination av dessa för att tilldela varje spelare en unik färgkombination. Totalt  $8^2 = 64$  unika färgkombinationer kunde således tilldelas under en spelsession. I figur 5.4 visas en lokal spelsession med 64 spelarkaraktärer där samtliga färgkombinationer har utnyttjats.



Figur 5.4: 64 dykare i spelet samtidigt, alla med en egen färgkombination.

Denna teknik resulterade till att över 50 unika användare kan ansluta till spelet och ha en unik karaktär. Upplägget med primärfärg och sekundärfärg kan även skalias upp till att tillåta ännu fler spelare då antalet färger lätt kan justeras.

## 5.2 Användartester

Under perioden 05/05/20–08/05/20 utfördes en användarundersökning av spelets grafiska gränssnitt. Undersökningen bestod av ett antal flervalsfrågor samt fritextfrågor avseende användbarheten och den upplevda intuitionen av två prototyper av spelets styrreglage. Två prototyperna presenterades för användaren var och en för sig samt sida vid sida. Dessa två prototyper kan ses i figur B.1 och figur B.2 i bilaga B. Data samlades in med hjälp av ett *Google Form*.

Antalet svar som samlades in var 63 stycken och gav möjlighet att fastställa vilken styrningsdesign som upplevs lättast att förstå och skapa en så bra spelupplevelse som möjligt. Noterbart är att alternativet med en *slider* som styrningsmekanik upplevdes av 85% ha flest val av styrriktning men alternativet med knappar upplevdes mest intuitiv av 73% och 71% hade valt knapparna om de själva hade fått välja mellan knappar och *slider*.

I de fria svaren uppgav många av de svarande att anledningen till att de hade föredragit knappar som gränssnitt var för att de ansåg att de snabbt skulle förstå sig på knapparna genom att testa sig fram, till skillnad från en *slider*. Genom de fria svaren som lyftes fram i användarundersökningen, se avsnitt B.2.1 i bilaga B, och de kvantitativa resultat som kan ses i avsnitt B.2 i samma bilaga. Att använda kontroller som användare är bekanta med sedan tidigare och är enkla att använda gör därmed att spelaren kan behålla sin blick fokuserad uppåt på en domskärm. Fullständigt resultat av användarundersökningen kan ses i bilaga B.

## 5.3 Nätverkskommunikation

Som nämnt i avsnitt 4.7 använder spelet en två-vägskommunikation. Ett system skapades för att avgöra vilken data som skickas, samt vilken användare den tillhör. Informationen som skickas blir tilldelad användarens unika identifikationsnummer, vilket möjliggör att informationen endast används till att hantera den berörda användaren. Detta gäller för båda kommunikationsvägarna vid exempelvis hantering av styrdata, poänguppdatering eller bortfall av spelare.

Utvecklingsgruppens undersökning med nätverkskommunikation för spelapplikationen resulterade i en lösning som tillåter återanslutning av en spelare då detta säkerhetsställer att ingen blir bortlockad på grund av eventuella nätverksstörningar. Det implementerades två lösningar under utvecklingen för att hantera bortfall av spelare under spelets gång. Den ena lösningen möjliggjorde återanslutning med hjälp av *cookies*. Efter frånkoppling från spelet fick användaren då möjligheten att återansluta till spelet med samma tilldelade dykarkarakter som tidigare. Detta krävde sparad data, såsom spelarens användarnamn och unika identifikationsnummer. Denna lösning var bra i teorin men i spelapplikationen fungerade den inte tillförlitligt då spelapplikationen behövde omstruktureras för att söka efter potentiella sparade *cookies* vid ett eventuellt bortfall.

I den andra lösningen möjliggjordes återanslutning utan sparad data och användaren fick möjlighet att återansluta till spelet med en ny dykarkarakter. Det beslutades att den andra lösningen fungerade bättre av de två vid ett kort fleranvändarspel där det viktigaste med spelet är att skapa en engagerande upplevelse för spelarna och utvecklingsgruppen ville främja att alla kan delta. Den resulterade lösningen blev därför att vid frånkoppling av användare i spelet skickas det unika identifikationsnumret till spelapplikationen och den berörda dykarkarakturen blir avaktiverad. Karaktären försvinner därmed från domskärmen. Användaren får sedan möjligheten att ansluta till spelet igen med en ny karaktär.

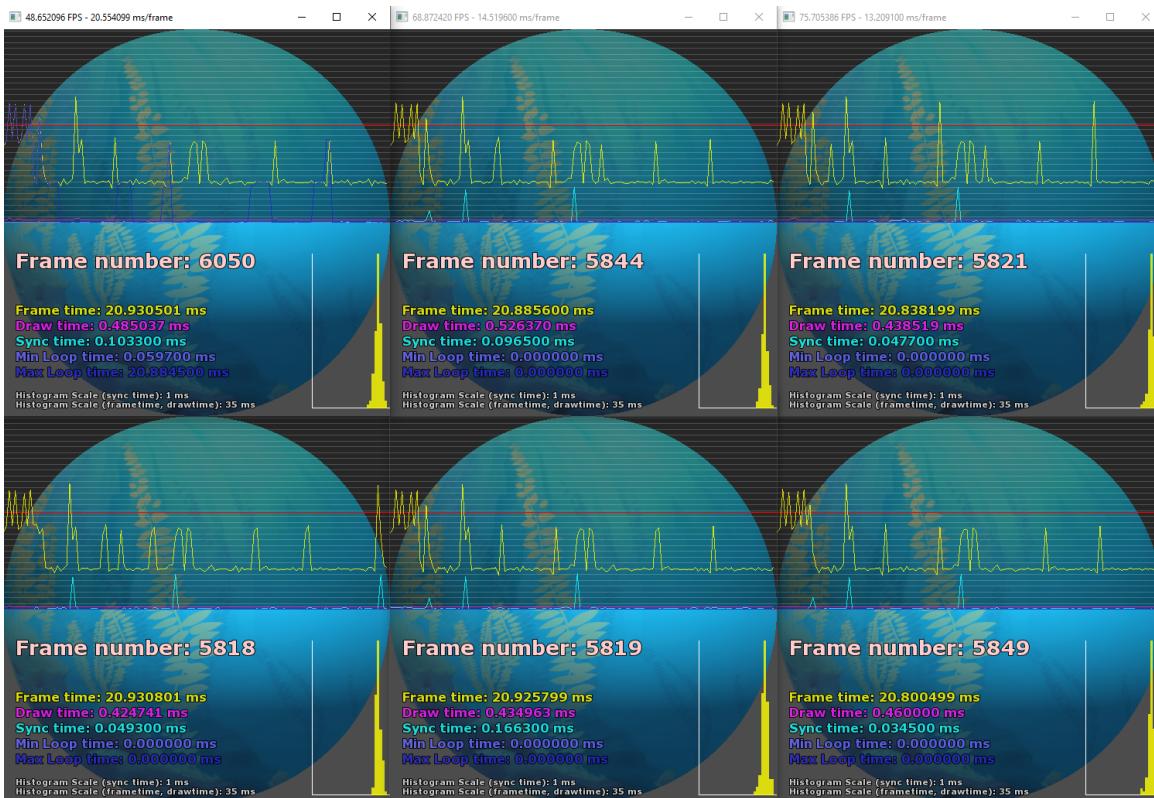
För att skicka all nödvändig data implementerades en WebSocket-lösning med JavaScript-biblioteket Websockets och C++-biblioteket Libwebsockets. Genom Websockets kan data skickas på ett sätt som är både skalbart och icke-blockerande. Att lösningen är icke-blockerande innebär att spelet inte behöver stanna upp för att skicka eller ta emot data utan att detta sker vid sidan av resterande spellogik.

Spelet skickar ut data till alla spelare samtidigt och är optimerat för att skicka så lite onödig information som möjligt. Information som inte behöver skickas varje bildruta, så som tiden, skickas en gång varje sekund för att spara in på den data som behöver skickas.

## 5.4 Spellogik och prestanda

Den spellogik som den slutliga spelapplikationen bygger på är ett system där antalet *collectibles* definieras av antalet spelare som är anslutna. Detta resulterar i en underhållande spelupplevelse för ett fåtal spelare likväld som för 50 spelare där det till exempel inte finns hundratals föroreningsobjekt när det enbart är 10 spelare anslutna. Genom att använda en objektpool för att initialisera alla *collectibles* skapas även ett system som inte behöver allokerar nytt minne för föroreningar kontinuerligt utan istället kan hämta en referens till en förskapade behållaren. Detta säkerhetsställer ett välfungerande system som dessutom är skalbart för flertalet spelare och då objektpoolen kan justeras i antal möjliggör det även för vidareutveckla projektet för ännu högre spelarkapacitet.

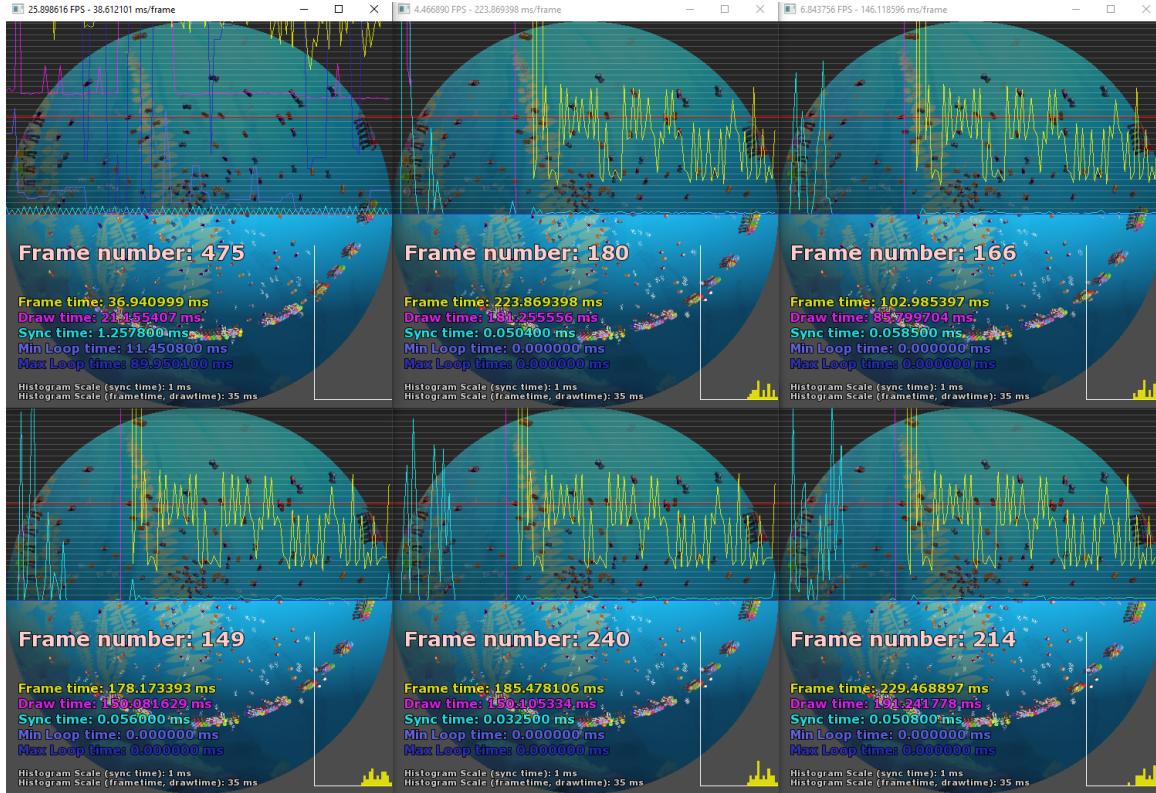
För att mäta prestandan av nodsynkroniseringen används ett profileringsverktyg som är inbyggt i SGCT för att visualisera hur lång tid olika processer i applikationen tar. För nodsynkroniseringen är parametern *sync time* (synkningstid) relevant då denna redovisar dels hur lång tid spelsimuleringen tar på huvudnoden och hur lång tid det nya tillstånden i spelet tar att appliceras på övriga klientnoder. Detta redovisas utan objekt i figur 5.5 och värsta fallet med 402 objekt i figur 5.6 där huvudnoden är den övre vänstra rutan och resterande är klientnoder.



Figur 5.5: Nodsynkronisering med sex noder utan spelobjekt.

I figur 5.5, utan några aktiva spelobjekt, ligger synkningstid inom intervallet [0.03, 0.16] ms för klientnoderna och  $\approx 0.10$  ms för huvudnoden. Trots att det inte finns några objekt att synkronisera till klientnoderna finns det ändå en viss *overhead* inom synkfunktionaliteten som står för tidsåtgången. Huvudnodens synkningstid spenderas även på *overhead* då det inte finns några objekt att uppdatera.

I figur 5.6, med 402 aktiva spelobjekt, ligger synkningstid inom intervallet [0.032, 0.058] ms för klinoden och  $\approx 1.26$  ms för huvudnoden. Därmed syns att datatransmissionen inte nödvändigtvis tar mycket längre tid än tiden som läggs på *overhead* i funktionaliteten. Den ökade tidsåtgången för huvudnoden beror främst på att kollisionsdetektering måste utföras mellan varje unikt par av spelare och förrening, vilket beskrivs i avsnitt 4.6.8.



Figur 5.6: Nodsynkronisering med sex noder och 402 spelobjekt.

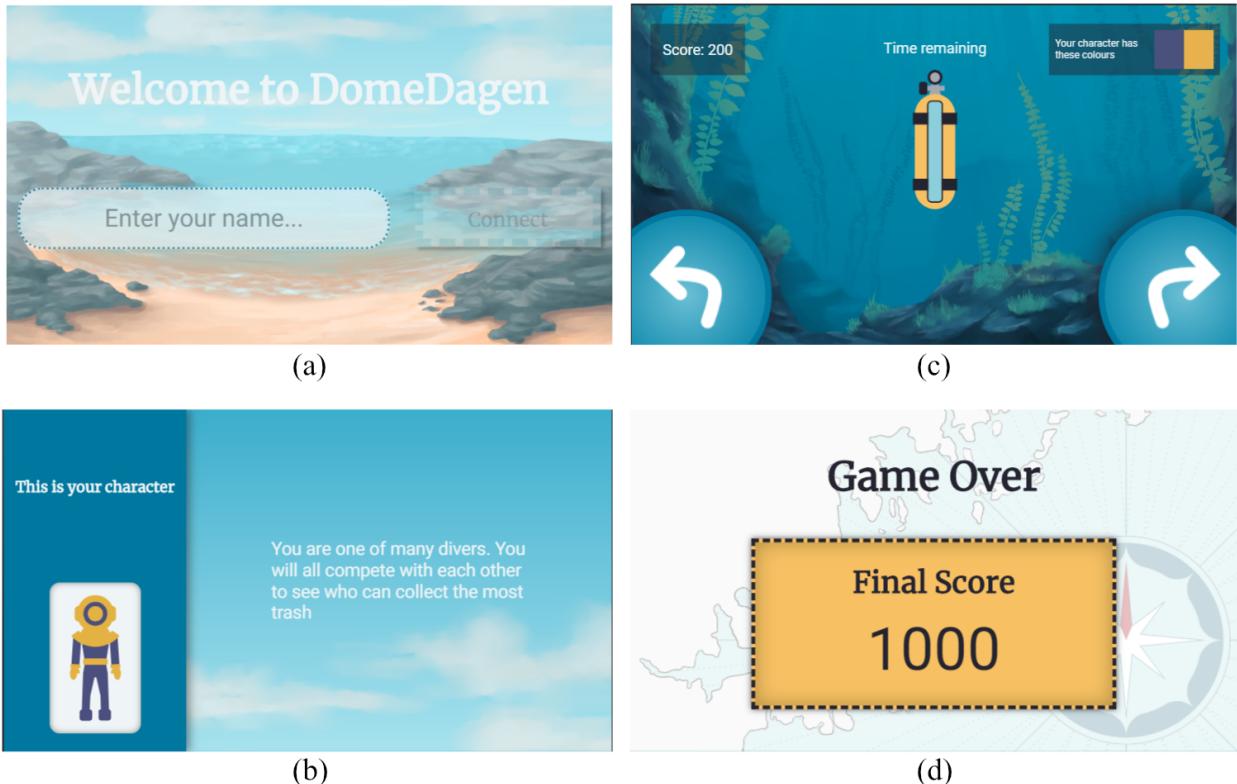
Då denna nodsynkronisering är uppmätt på en av utvecklarnas personliga dator, med ett grafikkort mycket svagare än de som används i domen läggs mycket tid på *draw time*, vilket mäter hur lång tid det tog att rendera den aktuella bildrutan. I domen har varje grafikkort endast en instans av spelet att rendera medan dessa bilder krävde att grafikkortet renderade alla sex instanser samtidigt. Testerna i domen har visat att *draw time* är det minsta problemet för att klara systembegränsningen beskriven i avsnitt 4.1.2 på grund av den tillgängliga grafiska processorkraften.

## 5.5 Spelapplikationen

Detta avsnitt visar den slutliga spelapplikationen som har utvecklats under arbetets gång för att kunna besvara frågeställningarna i avsnitt 1.2 på bästa sätt. Här visas ett antal bilder för att ge läsaren förståelse för den produkt som har varit basen till att undersöka projektets syfte.

### 5.5.1 Användargränssnitt

I figur 5.7 visas det färdigställda användargränssnittet för hemsidans fyra huvudsakliga delar. Spelets slutliga användargränssnitt låter användaren ansluta till spelet genom att ange ett spelarnamn varefter spelets information, principer och mål samt spelarens karaktär presenteras – detta sker innan spelet i sig har startats. När spelet väl har startat visas spelets kontroller, med två knappar enligt resultat från användarundersökningen, spelarens poäng samt en indikator för återstående speltid i form av en syrgastub. När spelet är över visas slutligen spelarens totala poäng.



Figur 5.7: Hemsidans olika skärmar i dess form vid rapportens skrivande. (a) visar anslutningsskärmen när användaren ej har angett sitt namn. (b) visar informationsskärmen med spelarens karaktär och information om spelet. (c) visas när spelet körs och har bland annat spelarens poäng, färger och kvarstående tid. (d) visas när spelet är över och presenterar spelarens slutpoäng.

### 5.5.2 Fullständig spelupplevelse

I figur 5.8 och figur 5.9 visas resultatet från en session av spelapplikationen på domskärmen i Visualiseringsscenter C. Det kan noteras att ett antal användare är anslutna och har blivit tilldelad en dykarkarakter i olika färger.



Figur 5.8: Bild på spelapplikationen i dombiografen på Visualiseringsscenter C.



Figur 5.9: Bild på spelapplikationen i dombiografen på Visualiseringsscenter C.

# Kapitel 6

## Analys och diskussion

De val som gjorts i denna rapport gällande projektmetodik och rutiner för implementation av ett fleranvändarspel i dommiljö har tagit fäste på de positiva aspekter som lyfts fram gällande agila utvecklingmetoder för just spelutveckling. De resultat som har tagits fram under utvecklingsprocessen har analyserats ur ett perspektiv av produktutveckling men också för hur väl metoderna och processerna har främjat undersökandet av de frågeställningar som togs upp i avsnitt 1.2.

### 6.1 Metod

Projektets arbete inleddes med planering av ett överliggande upplägg för hela utvecklingsperioden. Detta schema har till stor del följts men två sprintar blev förlängda, den första gången på grund av påskledighet och långhelg och den andra gången på grund av att domen var otillgänglig vilket innebar att ett viktigt domtest blev framskjutet.

Tidigt under arbetet bröt den globala virusspridningen av COVID-19 ut vilket tvingade framtida arbetet att ske på distans, se avsnitt 1.3. Detta har lett till att en stor del av det planerade arbetet har behövt anpassas till de nya förutsättningarna, vilket har ställt stora krav på projektmetodiken. Valet att använda Scrum fick därmed större fördelar än vad som först kunnat planeras för i projektstarten då den flexibilitet och anpassningsbarhet som lyfts fram i Scrum har fått testas till sin gräns. Med de korta sprintar som planerats för i början av projektet lyckades utvecklingsarbetet anpassas till distanssituationen på ett funktionellt sätt där även ett antal av sprintarna blev förlängda för att utvecklingsgruppen skulle hinna med allt det arbete som var satt för den sprinten.

En av de större bristerna med projektmetoden som sattes innan projektet var att mycket av undersöningen för frågeställningarna var baserade på en kontinuerlig testning av spelet på plats i dombiografen. Testning på plats i domen blev efter distansläget tvingat till att ske med enbart projektgruppen på plats. Under utvecklingens gång hade det varit en stor fördel att genomföra ett större användartest i domen för att säkerhetsställa att kapaciteten för flertalet simultana användare håller kraven enligt avsnitt 4.1.1 samt att spelkontrollerna upplevs lättbegripliga enligt krav i samma avsnitt. Som tidigare nämnt kunde detta inte genomföras på grund av de externa förhållanden med COVID-19 som låg bortom utvecklingsgruppens möjlighet att påverka.

Utvecklingsgruppen tog i projektplaneringen hänsyn till de svårigheter som kunde tänkas inträffas vid utveckling för en domskärm, så som krav på virtuell testning utanför den faktiska dombiografen. Den inläsningsperiod på verktyget SGCT och säkerhetsställning att samtliga gruppmedlemmar kan köra en virtuell modell av domskärmen fungerade helt enligt plan och gav gruppen en solid grund att stå på i arbetet. Ett problem som uppkom under projektets gång var svårigheten att uppskatta den enorma skala som finns i domteatern. Det är svårt att översätta den välvda bioskärmen till de platta

datorskärmar som används för testning i projektet då saker som ser små ut på datorskärmen kan vara enorma på bioduken. För att kunna färdigställa saker i rätt skala måste testning ske i dombiografen, vilket blev extra problematiskt under distansläget. Projektmetodiken för en situation där kontinuerlig domtestning kan ske bör därför ta ytterligare hänsyn till svårigheten att översätta en 2D-skärm till en sfärisk domskärm.

Den metod som presenterades av projektgruppen innan projektet bedöms slutligen ha varit en bra metodik om allt hade gått att genomföra utan problem med testning. All kod har blivit testad och granskats av flera personer innan den till slut läggs till den slutgiltiga produkten och kommunikationen kring detta har fungerat bra. Effektiviteten i testningsprocessen kan ha blivit lägre för gruppen då en utvecklare kunde behöva granska kod som var i ett annat programmeringsspråk än dennes fokuseringsområde men projektets huvudsakliga effektivitet har ökats med detta då alla utvecklare snabbt kunde byta fokuseringsområde och hjälpa till där behov uppstod om de egna arbetsuppgifterna var färdigställda.

Utvecklingsmetoden har förlitat sig mycket på det interna verktyget SGCT och detta har visat sig vara väldigt givande i utvecklingen då SGCT utvecklas på Linköpings Universitet vilket har gjort det enkelt att få tag på dem som aktivt utvecklar ramverket. Det är även öppen källkod vilket innebär att källkoden kan ändras vid behov.

### 6.1.1 Källkritik

För att säkerhetsställa att detta utvecklingsprojekt har korrekt information och den vetenskapliga grund som krävs har källor enbart hämtats ifrån officiell dokumentation, litteratur och från upphovsmakare. Detta gör att den metodik som har användts har varit förespråkad av de ledande rösterna från varje verktyg.

Vid mjukvaruutvecklingen har vissa avsteg från formell praxis gjorts och ytterligare källor används för att se till att slutprodukten beter sig i enlighet med dess syfte. Detta är ett nödvändigt steg gällande lösning av programkodsrelaterade problem men för att säkerhetsställa ett källkritiskt informationshämtande har dock aldrig konkret teori hämtats från andra källor än officiell dokumentation och litteratur.

Då utvecklingen främst har skett på plattformar med öppen källkod har det varit viktigt att se till att kodbasen är uppdaterad och att de tidigare utvecklarna inte övergivit koden. Det finns en risk med att om ett verktyg eller bibliotek är utan uppdatering under en längre tid kan detta bli en säkerhetsrisk då det kan sakna de moderna skydd som kommer med aktiv utveckling. Genom den genomarbetade plan för projektmetodiken avsåg utvecklingsgruppen förebygga och motarbeta de brister som kan komma med agil utvecklingsmetodik för programmering där ett snabbt och praktiskt arbetssätt går före planering och dokumentation. Med denna förkunskap kunde utvecklingsgruppen bibehålla ett kritiskt synsätt till öppen källkod och externa verktyg och säkerhetsställa att den implementerade koden och dokumentationen både refererade till korrekt information och höll en hög standard.

I högsta möjliga grad har utvecklingsgruppen försökt vara kritiska till eventuella säkerhetsrisker. Det existerar en viss risk för en skada och sabotage gällande en spelapplikation vid exempelvis kodinjektion. I detta projekt har inga konkreta åtgärder tagits gällande säkerhetsaspekter då den interna spelapplikationen bedömdes hanteras på ett sådant sätt att även vid en skadlig användarattack skulle enbart spelet kunna kraschas och ej ta ned hela det system som bygger upp dombiografen då detta skyddas av egna säkerhetsmekanismer. För ett mer omfattande projekt som skulle behöva användas i ett vidare sammanhang borde mer vikt läggas på potentiella säkerhetsrisker för att se till att spelet kan uppnå sitt syfte gentemot såväl kund som användare.

## 6.2 Resultat

De resultat som lyfts fram i detta projekt har överlag varit positiva och lett till att frågeställningarna som sattes upp vid projektets start har kunnat besvaras. I och med situationen med distansläge, se avsnitt 1.3, finns det dock en ökad felfaktor gällande resultaten än vid ett projekt som skett utan denna yttre påverkan. Den största problematiken omfattar de uteblivna systemtesten som skulle ha utförts i dombiografen. Detta betyder att även om projektgruppen har kunna utföra en viss del av designrelaterade tester mot användare via internet och utvärderat spelapplikationen både lokalt och i dombiografen så har inga maximala stresstester kunna genomföras.

Ett av projektets mål var att spelat ska kunna spelas innan föreställningar i domen. Eftersom användartest som tidigare nämnt inte kunnat utföras som planerat i domen är det osäkert hur intuitiva sådant som spelkontrollerna upplevs av användare. Den användarundersökning som gjordes gav användbar feedback, men ett användartest i domen med många användare hade varit till ännu större nytta eftersom mappningen mellan spelkontrollerna och spelkaraktären på domens sfäriska yta är svår att förmedla i en enkätundersökning. Genom att låta styrningen utgå ifrån spelarens riktning istället för någon slags absolut riktning kan dock många problem undvikas; de vinkelegenskaper som är annorlunda på en sfär jämfört med ett plan blir mindre uppenbara och användares kunskaper från ”vanliga” spel blir mer applicerbara.

Hemsidan spelskärm innehåller mycket information med tanke på att den endast visas under spelet då användarna rimligen borde rikta sitt fokus mot domen. Information som nuvarande poäng är dock något som potentiellt är praktiskt för användaren att ha tillgång till, och inget som lätt går att visa på domen med tanke på antalet spelare. Eftersom användarna inte *måste* veta sin poäng under spelets gång ansågs det inte som ett större problem att det visades på den mobila enheten. Tidsmätaren är ändå viktigare och den visar dessutom samma information för alla spelare. Den skulle därför kunna visas på domen istället för i den mobila enheten. Spelarna skulle även kunna uppmärksammas på att tiden börjar ta slut genom någon typ av feedback.

En av de stora utmaningarna var att göra det möjligt för spelarna att identifiera sin spelare på domen trots att antalet spelare är stort. Den huvudsakliga lösningen som implementerades var att ge varje spelare en unik färgkombination som visas på hemsidan. Domens storlek och antalet möjliga spelare kan dock innebära att det ändå kan ta en stund att hitta karaktären i början av spelet. Om mer tid hade funnits kunde även spelarens position ha skickats till klienten för att visa ungefärlig position i domen på hemsidan, vilket även kan gynna i fall såsom färgblindhet hos en spelare där det är än svårare att urskilja sin karaktär snabbt. En utökning med mönster utöver färgkombinationerna kunde även ha varit ett sätt att ge ytterligare individualitet till spelarna.

Spelet har en begränsad mängd funktionalitet. Att utöka spelmekaniken med diverse *powerups* och dylikt hade kunnat bidra till ett spel som är roligare i längden. Det ska dock beaktas att fler funktioner skulle göra inlärningskurvan brantare. Funktioners nytta måste alltså vägas mot den komplexitet de drar med sig. I ett spel för korta sessioner i publika miljöer är det utan tvekan intuitiviteten som är viktigast och därför bör produktens ”brist” på extra funktioner på många sätt ses som en tillgång då detta medför maximalt underhållningsvärde för en så bred målgrupp som möjligt.

De resultat som diskuteras ovan, om saker som kunde gjorts annorlunda eller utvecklats med spelapplikationen, har alla influens på resultatet av den undersökning av hur ett optimalt, immersivt och skalbart spelsystem utvecklas för en domskärm. Den främsta påverkan på resultatet bedöms vara bristen på kontinuerliga stresstester och detta innebär att det potentiellt finns ytterligare optimeringar, omstruktureringar och designrelaterade förändringar som kunde ha lyft projektet och gett ännu bättre resultat. Detta innebär självklart en möjlighet för vidare arbete med detta projekt som utgångspunkt. Dock bedöms projektgruppens undersökningar ha mynnat ut i ett starkt och gott resultat givet den yttre påverkan som uppkom i och med distansläget.

## 6.3 Etisk och samhällelig reflektion

Likt många andra projekt möter detta arbete en etisk aspekt gällande könsdiskriminering. För att ingen individ ska känna sig exkluderad eller utsatt är utformningen av spelets karakterer gjord på ett sådant sätt att ingen specifik könstillhörighet definieras och inte heller exkluderar någon individ. Även speldesignen är utformad med detta i åtanke. Dykarkarakturen är utformad för att vara könsneutral och blir tilldelad slumpmässiga färger.

Eftersom domteatern på Visualiseringsscenter C finns tillgänglig för allmänheten är spelet anpassat för en bred målgrupp av användare. Yngre besökare ska enkelt kunna använda spelets användargränssnitt och kunna ta del av spelet utan att känna sig exkluderade. En bred målgrupp i potentiella spelares ålder ställer även krav på utformningen av spelmekanik med hänsyn till våld och olämpligt innehåll. Ett flertal rapporter tar upp hur visst spelinhåll i spel som barn och ungdomar möter kan ge upphov till mer aggressiva beteenden. Som benämns i [19] finns det viktiga aspekter att lära sig av från tidigare utgivna spel för att kunna förhindra en influens av aggressivt beteende i den mån som det går. Detta spel är därför utformat med minimal interaktion mellan spelare. Spelet går inte ut på att skada varandra, och fienden i fråga är ett icke-levande ting – föröreningarna.

En av den moderna tidens största ödesfrågor är klimatförändringen. Spelet förhåller sig till detta med en speldé grundad i problemet med förörening av världens hav. Detta kan möjliggöra en diskussion hos besökarna på dombiografen och bidra till en ökad medvetenhet kring klimathotet. Genom att anpassa spelet till centrets breda målgrupp kan det för de yngre besökarna vara en informativ och givande upplevelse. En upplevelse som kan inspirera dem till att kämpa mot klimathotet.

# Kapitel 7

## Slutsatser

Denna rapport redogör för det arbetet som utvecklingsgruppen har genomfört i arbetet med att utveckla ett immersivt och skalbart spelsystem för en domskärm. Huvudsyftet med projektet har varit att undersöka hur ett immersivt och skalbart system optimalt utformas för en domskärm och besvara de frågeställningar som togs upp under avsnitt 1.2. Utvecklingsgruppen kan fastställa att i undersökningen för hur ett immersivt och skalbart system utformas är ett fleranvändarspel med nätverkskommunikation genom mobila enheter optimalt. Att använda en spelapplikation möjliggör för undersökningar av underhållning och användarvänlighet samt styrning via mobila enheter möjliggör för undersökningar av nätverksinfrastrukturoptimeringar samt ett varierande antal spelare.

Spelapplikationen som utvecklats under detta projekt kunde framgångsrikt spelas på dombiografens skärm i Visualiseringsscenter C. De tre frågeställningarna är besvarade nedan.

**På vilket sätt bör kontrollerna för ett spel i en dombiograf utformas för att vara lättbegripliga och för att användaren ska kunna behålla sin blick fokuserad uppåt på dombiografens skärm?**

Den första frågeställningen har besvarats till största del av den användarundersökning som har utförts enligt planering i avsnitt 3.6.2 och vars detaljer och resultat återfinns i bilaga B. Genom att använda två knappar kan användaren konstant ha sina fingrar på sin mobila enhet. Det ger också en enkelhet då användaren kan fokusera på domskärmen och hålla blicken lyft. Under utvecklingsprocessen utvärderades ett flertal gränssnitt för styrning, vilket redogjordes för i avsnitt 4.5.1. Detta i samband med resultaten från användarundersökningen ger intrycket av att det slutliga användargränssnittet avseende styrningen uppfyller målet som den första frågeställningen lägger fram.

**Hur utvecklas en nätverksinfrastruktur för att hantera såväl kommunikationsvägar gentemot ett varierande antal simultana användare – beroende på antal besökare i domen – som eventuella, temporära bortfall av användare – exempelvis på grund av en avbruten nätverksanslutning?**

Den andra frågeställningen relaterar till hur en nätverksinfrastruktur bör hantera ett varierande antal simultana användare. Detta har visat sig vara en svårighet och avvägning gällande om ett bortfall av användare bör kunna återansluta sig till sidan och vad som i sådana fall händer med den nu ej anslutna spelarens karaktär i spelet. Genom utvecklingsarbetet i detta projekt fastställs det att den bästa lösningen för denna typ av kortare spel är att nätverkskommunikationen utformas på ett sådant sätt så att datakommunikationen är både skalbart och icke-blockerande. Detta kan säkerhetsställas genom att se till att spelet inte behöver stanna upp för att skicka eller ta emot data utan att detta sker vid sidan av resterande spellogik. Spelare bör även kunna återansluta sig vid ett eventuellt bortfall av internet för att spelupplevelsen ska kunna maximeras men den gamla spelarkarakteren bör tas bort vid ett bortfall, även om spelaren kan tillåtas återansluta till samma namn och färg på karaktären. Detta minimerar risken för att det ska ske någon form av *overflow* av objekt i scenen och gör att spellogiken fungerar enligt utformning för att vara underhållande för minst 5 och upp till 50 spelare.

## Vilka tekniker och strategier kan utnyttjas för att säkerställa ett dynamiskt spelsystem – där allt emellan 5 och 50 användare kan interagera simultant – som är väl anpassat för en dombiograf med avseende på tydlighet och underhållningsvärde för användaren?

Den tredje frågeställningen besvaras till viss del av det som nämntes kort i föregående stycke genom spellogiken och antal objekt i scenen. Genom att sätta hur många simultana *collectibles* som får existera i scenen samtidigt säkerhetsställs det att det blir ett utmanande spel, oavsett antalet spelare. Ytterligare strategier för att säkerhetsställa ett välfungerande och dynamiskt spelsystem som är väl anpassat för en bred skala av antal spelare är tydlighet gällande spelkaraktärerna. Genom att slumpmässigt dela ut primära och sekundära färger på dykarens klädsel, enligt det som togs upp i avsnitt 4.6.2, minimeras risken för att det ska uppstå förvirring gällande vilken karaktär som hör till vilken spelare. Med de avgränsningar som togs upp i avsnitt 1.3 har spellogiken designats och gjorts väl avvägd för att kunna hantera upp till 50 simultana användare. Även den övriga designprocessen ser till att denna frågeställning besvaras genom att hålla ett enhetligt användargränssnitt som speglar spelets tema. Det ger både tydlighet och extra underhållningsvärde genom att knyta samman slutprodukten.

Ett fleranvändarspel som styrs av handhållna enheter och är avsedd för en dombiograf möter flertalet svårigheter gällande design och implementation, men genom en tydlig spellogik och effektiv nätverkskommunikation har utvecklingsgruppen lyckats ta fram en slutprodukt som väl undersöker de frågeställningar som satts upp för projektet, under de förutsättningar som tas upp i avsnitt 1.3.

## 7.1 Vidareutveckling

Med detta projekt som grund finns det ett antal möjligheter för vidareutveckling. Med mer tid och möjligheter till användartester i domen hade det varit intressant att utforska idén om styrning med mobila enhetens lutning, vilket inte kunde utforskas på grund av de tekniska problem som beskrivs i avsnitt 4.7. Vidare kan det undersökas hur systemet kan utvecklas för att ytterligare underlätta för användare att identifiera sin spelare i domen, till exempel via en indikator i den mobila enheten. Hur ska en sådan i så fall utformas för att användaren lätt ska kunna entydigt tolka den?

Utvecklingsgruppen kom fram i undersökningarna fram till att det är optimalt att tillåta återanslutning av en spelare vid ett eventuellt bortfall av internet men ingen optimal lösning för hur spelarnamn och id för spelaren sparas kom fram till under detta projekt. En vidareutveckling som gynnar den övergripande upplevelsen av spelet vore att undersöka hur en återanslutsningsprocess som bibehåller samma karaktär vid en ny anslutning görs på bästa sätt.

Ljud är en annat område som kan undersökas. Det mest intressanta vore att undersöka återkoppling i form av ljud då en förörening plockas upp. Kan det användas med så många spelare som 50, och i så fall hur skall det utformas? Lösningar som kan vara intressanta är surroundljud och ljud i användarens mobila enhet på hemsidan. I övrigt finns det många mindre saker som kan förfinas för att förbättra användarvänligheten och spelupplevelsen, till exempel kunde en QR-kod visas i domen som leder användarna direkt till hemsidan.

# Litteraturförteckning

- [1] Visualiseringscenter C, URL: <http://visualiseringscenter.se/visualiseringscenter-c> (hämtad 2020-05-08).
- [2] E. Lantz, "A survey of large-scale immersive displays", vol. 252, aug. 2007, s. 1. DOI: [10.1145/1278240.1278241](https://doi.org/10.1145/1278240.1278241).
- [3] J. Downs, F. Vetere, S. Howard, S. Loughnan och W. Smith, "Audience Experience in Social Videogaming: Effects of Turn Expectation and Game Physicality", i *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14, Toronto, Ontario, Canada: Association for Computing Machinery, 2014, s. 3473–3482, ISBN: 9781450324731. DOI: [10.1145/2556288.2556965](https://doi.org/10.1145/2556288.2556965). URL: <https://doi.org/10.1145/2556288.2556965>.
- [4] W. Goddard, A. Muscat, J. Manning och J. Holopainen, "Interactive Dome Experiences: Designing Astrosurf", i *Proceedings of the 20th International Academic Mindtrek Conference*, ser. AcademicMindtrek '16, Tampere, Finland: Association for Computing Machinery, 2016, s. 393–402, ISBN: 9781450343671. DOI: [10.1145/2994310.2994339](https://doi.org/10.1145/2994310.2994339). URL: <https://doi.org/10.1145/2994310.2994339>.
- [5] A. Bock, A. Emil, J. Costa, G. Payne, M. Acinapura, V. Trakinski, C. Emmart, C. Silva, C. Hansen och A. Ynnerman, i *OpenSpace: A System for Astrographics*, IEEE Transactions on Visualization och Computer Graphics, 2019, s. 1–10, ISBN: 9781450343671. DOI: [10.1145/2994310.2994339](https://doi.org/10.1145/2994310.2994339). URL: <https://ieeexplore.ieee.org/document/8805462>.
- [6] N. Andersson, D. Camarda, J. Eliasson, E. Häger, J. Hägerstand, F. Johnson och O. Westberg, "Game of Domes", Linköpings Universitet, 2014, ISBN: 9781450343671. DOI: [10.1145/2994310.2994339](https://doi.org/10.1145/2994310.2994339). URL: [http://weber.itn.liu.se/~danjo37/courses/TNM094/reports/TNM094-2014-05\\_Game\\_of\\_Domes.pdf](http://weber.itn.liu.se/~danjo37/courses/TNM094/reports/TNM094-2014-05_Game_of_Domes.pdf).
- [7] Y. Selling, S. Källberg, A. Palm, J. Fröberg, A. Bergman och P. Gunnarsdotter, "DomeNation", Linköpings Universitet, 2019. URL: [http://weber.itn.liu.se/~danjo37/courses/TNM094/reports/TNM094\\_2019VT\\_YN-Projekt%5C%20H%5C%20-%5C%20DomeNation.pdf](http://weber.itn.liu.se/~danjo37/courses/TNM094/reports/TNM094_2019VT_YN-Projekt%5C%20H%5C%20-%5C%20DomeNation.pdf).
- [8] K. Schwaber och J. Sutherland, "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game", engelska, 2017. URL: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf?forcedefault=true> (hämtad 2020-05-13).
- [9] B. Stroustrup och H. Sutter, *C++ Core Guidelines*, 2019. URL: <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines> (hämtad 2020-05-01).
- [10] Vsrajeshvs. (mars 2002). Singleton Pattern & its implementation with C++, URL: <https://www.codeproject.com/Articles/1921/Singleton-Pattern-its-implementation-with-C> (hämtad 2020-05-11).
- [11] Blender Online Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL: <http://www.blender.org>.

- [12] R. S. Wright, G. Sellers och N. Haemel, *OpenGL superBible : comprehensive tutorial and reference*. Addison-Wesley, 2014, ISBN: 9780321902948. URL: <https://login.e.bibl.liu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=cat00115a&AN=lkp.689193&lang=sv&site=eds-live&scope=site>.
- [13] N. Llopis. (dec. 2009). Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP), URL: <http://gamesfromwithin.com/data-oriented-design> (hämtad 2020-05-11).
- [14] Cppreference. (2018). C++ named requirements: ContiguousContainer, URL: [https://en.cppreference.com/w/cpp/named\\_req/ContiguousContainer](https://en.cppreference.com/w/cpp/named_req/ContiguousContainer) (hämtad 2020-04-29).
- [15] R. Nystrom, *Game programming patterns*. Genever Benning, 2014, kap. 19, s. 305–321.
- [16] Y.-B. Jia. (2013). Quaternions and Rotations, URL: <http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf> (hämtad 2020-05-11).
- [17] G. Baravdish, *Linjär algebra TNA002*. Norrköping: Institutionen för Teknik och Naturvetenskap vid Linköpings Universitet, 2016.
- [18] M. Nakazawa och H. Koike, “Synthesizing Fixed Point of Views from a Spinning Omnidirectional Ball Camera”, i *Proceedings of the 8th Augmented Human International Conference*, ser. AH ’17, Silicon Valley, California, USA: Association for Computing Machinery, 2017, ISBN: 9781450348355. DOI: [10.1145/3041164.3041201](https://doi.org/10.1145/3041164.3041201). URL: <https://doi.org/10.1145/3041164.3041201>.
- [19] H. J. Brown, *Videogames and Education [Elektronisk resurs]*. 2008.

## Bilaga A

### Utvecklarnas individuella ansvarsområden och bidrag till slutprodukten

Utvecklingsteamet arbetade såväl individuellt som i grupp under projektets gång. I avsnitt 3.2.1 lades utvecklarnas principiella ansvarsfördelning, arbetsgrupper och särskilda organisatoriska roller fram. Nedan presenteras varje utvecklarens individuella bidrag samt eventuella organisatoriska ansvar för projektarbetet.

**Frans Johansson**, Sekreterare under projektgruppens möten. Arbetade med gränssnittsdesign och utveckling av hemsidan. Hanterade implementation av färgvälvjarsystem för spelarkarakturen och tillhörande *shader* för dykarmodellen i spelet.

**Josefine Klintberg**, Arbetade med 3D-modellering och texturering, riggning och animering av spelobjekt. Implementerade den ekvirektangulära mappningen för bakgrundsobjektet och integrerade i koden för applikationen.

**Iris Kotsinas**, *Scrum Master* som ansvarade för möten och att projektmetodiken följdes. Ansvarade för serverimplementationen och arbetade med att integrera och upprätta kommunikationen mellan spelapplikation, server och hemsida samt se till så att alla bibliotek kopplades samman korrekt.

**Erik Larsson**, Arbetade med hemsidan och serverhanteringen för anslutning av spelare till spelet.

**David Robín Karlsson**, Produktägare och kundkontakt som upprätthöll kommunikation med kunden i projektet. Arbetade på spelapplikationen, nedsynkronisering, implementerade designmönster och såg till att systemarkitekturen samt kodstandard följdes.

**Algut Sandahl**, Arbetade med att implementera styrning och spellogik för spelapplikationen samt rendering.

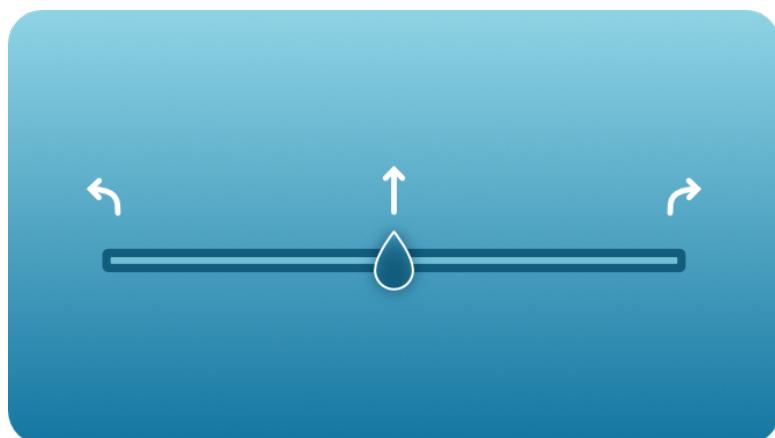
## Bilaga B

### Användarundersökning av styrningsgränssnitt

Under perioden 05/05/20–08/05/20 utfördes en användarundersökning av spelets grafiska gränssnitt. Två prototyperna vilka återges i figur B.1 och B.2 presenterades för användaren var och för sig samt sida vid sida.



Figur B.1: Prototyp av grafiskt gränssnitt för styrning med två knappar.



Figur B.2: Prototyp av grafiskt gränssnitt för styrning med en *slider*.

## B.1 Formulärets frågor

Frågeformuläret bestod av fem delar vilka presenterades för användaren en och en i ordningen 1–5. Nedan följer en redogörelse för formulärets delar, frågor och svarsalternativ.

### Del 1 Frågor avseende användarens målgruppstillhörighet.

Fråga 1: Din ålder

- \* 10–18 år
- \* 19–29 år
- \* 30–50 år
- \* över 50 år

Fråga 2: Har du besökt en dombiograf tidigare?

- \* Ja
- \* Nej

Fråga 3: Hur stor vana har du av mobilspel?

- \* Spelar aldrig
- \* Spelar sällan
- \* Spelar ibland
- \* Spelar ofta
- \* Spelar i princip dagligen

### Del 2 Frågor avseende styrning med knappar.

Figur B.1 presenterades för användaren med följande text: ”Följande bild avser illustrera hur styrreglagen ser ut på din mobiltelefon. Föreställ dig att du håller din mobil framför dig och styr din karaktär som syns på domskärmen framför dig med hjälp av reglagen. Din karaktär har en fast hastighet som den rör sig framåt med.”

Samtliga frågor i denna del hade svarsalternativen: ”Svårt”, ”Medel” och ”Enkelt”.

Fråga 1: Hur upplever du att du förstår hur styrningen går till genom att titta på bilden?

Fråga 2: Hur upplever du att det skulle vara att svänga med denna styrning?

Fråga 3: Hur upplever du att det skulle vara att styra rakt fram med denna styrning?

### Del 3 Frågor avseende styrning med *sliders*.

Figur B.2 presenterades för användaren med följande text: ”Följande bild avser illustrera hur styrreglagen ser ut på din mobiltelefon. Föreställ dig att du håller din mobil framför dig och styr din karaktär som syns på domskärmen framför dig med hjälp av reglagen. Din karaktär har en fast hastighet som den rör sig framåt med.”

Samtliga frågor i denna del hade svarsalternativen: ”Svårt”, ”Medel” och ”Enkelt”.

Fråga 1: Hur upplever du att du förstår hur styrningen går till genom att titta på bilden?

Fråga 2: Hur upplever du att det skulle vara att svänga med denna styrning?

Fråga 3: Hur upplever du att det skulle vara att styra rakt fram med denna styrning?

**Del 4** Jämförelse av de två prototyperna.

Figur B.1 och B.2 presenterades sida vid sida för användaren. Frågorna i denna del uppmanade användaren till att välja mellan alternativen ”Knappar” och ”Slider”.

Fråga 1: Vilken av styrningarna upplever du är mest intuitiv?

Fråga 2: Vilken av styrningarna upplever du har flest val av styrriktning?

Fråga 3: Vilken av styrningarna upplever du är enklast att styra rakt fram med?

Fråga 4: Vilken av styrningarna hade du valt i ett spel?

**Del 5** Avslutning.

Användaren fick möjlighet att lämna sina egna tankar angående de två prototyperna i den avslutande frågan ”Har du någon feedback eller kommentar på de styrningar som har presenterats i denna enkät?”

## B.2 Sammanställning av svarsdata

Totalt samlades 63 svar in. Dessa sammanställs i tabellerna B.1–B.5 samt i avsnitt B.2.1.

Tabell B.1: Svarsdata från användarundersökningen avseende målgruppstillhörighet av de svarande.

| Ålder      |          | Tidigare besök i dombiograf |          | Vana av mobilspel         |          |
|------------|----------|-----------------------------|----------|---------------------------|----------|
| Alternativ | Frekvens | Alternativ                  | Frekvens | Alternativ                | Frekvens |
| 19–29 år   | 55       | Ja                          | 35       | Spelar aldrig             | 6        |
| 30–50 år   | 8        | Nej                         | 28       | Spelar sällan             | 22       |
|            |          |                             |          | Spelar ibland             | 21       |
|            |          |                             |          | Spelar ofta               | 4        |
|            |          |                             |          | Spelar i princip dagligen | 10       |

Tabell B.2: Svarsdata från fråga 1 från del 2 samt del 3 ur formuläret avseende förståelse av de två styrningsgränssnitten.

| Hur upplever du att du förstår hur styrningen går till genom att titta på bilden? |         |        |
|---|---------|--------|
| Alternativ  | Knappar | Slider |
| Enkelt  | 41      | 39     |
| Medelt  | 20      | 17     |
| Svårt   | 2       | 7      |

Tabell B.3: Svarsdata från fråga 2 från del 2 samt del 3 ur formuläret avseende att få sin karaktär att svänga med de två styrningsgränssnitten.

| Hur upplever du att det skulle vara att svänga med denna styrning? |         |        |
|--|---------|--------|
| Alternativ   | Knappar | Slider |
| Enkelt   | 28      | 22     |
| Medelt   | 32      | 22     |
| Svårt  | 3       | 19     |

Tabell B.4: Svarsdata från fråga 3 från del 2 samt del 3 ur formuläret avseende att få sin karaktär att simma rakt fram med de två styrningsgränssnitten.

| Hur upplever du att det skulle vara att styra rakt fram med denna styrning? |         |        |
|---|---------|--------|
| Alternativ  | Knappar | Slider |
| Enkelt  | 32      | 35     |
| Medelt  | 14      | 19     |
| Svårt   | 17      | 9      |

Tabell B.5: Svarsdata från samtliga frågor från del 4 ur formuläret avseende en direkt jämförelse mellan de två styrningsgränssnitten.

| Fråga  | Knappar | Slider |
|--|---------|--------|
| Vilken av styrningarna upplever du är mest intuitiv?                   | 46      | 17     |
| Vilken av styrningarna upplever du har flest val av styrriktning?      | 10      | 53     |
| Vilken av styrningarna upplever du är enklast att styra rakt fram med? | 31      | 32     |
| Vilken av styrningarna hade du valt i ett spel?                        | 45      | 18     |

## B.2.1 Sammanställning av fria svar och allmän återkoppling

I del 5 av frågeformuläret fick den svarande möjlighet att i fritext lämna egna tankar, idéer och allmän återkoppling gällande styrningsgränssnittet. Dessa sammanfattas översiktligt nedan.

*Slider*-styrningen upplevedes icke-intuitiv av många svarande. Spelupplevelsen upplevdes försämrad av det gränssnittet dels på grund av att den upplevdes kräva att användaren vänder blicken mot *smartphonen* snarare än domskärmen under spelets gång samt att greppet runt *smartphonen* skulle blivit mindre bekvämt än för knapparna. Många svarande upplevde dock att detta gränssnitt visuellt förmade sin funktion bättre än knapparna, men att hur själva styrningen gick till återigen förblev aningen oklar. En idé som flera lyfte fram var att *slidern* borde återgå till sitt mittläge den släpps upp.

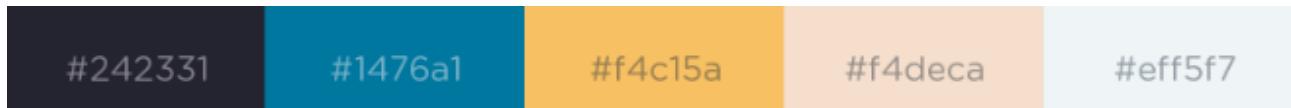
Knapparna uppskattades i det avseendet att de gav upphov till en naturlig position för tummarna under spelets gång. Många av de svarande uppgav att de hade föredragit detta gränssnitt trots att de upplevde att *slider*-styrningen bättre förmade sin funktion då de ansåg att de snabbt skulle förstå sig på knapparna genom att testa sig fram. En del svarande upplevde dock att pilarnas rotation på knapparna gav visst upphov till förvirring och att de hade föredragit om pilarna ej var roterade ”utåt”.

# Bilaga C

## Grafiska element i användargränssnittet

För att en enhetlig visuell känsla skulle förmedlas i slutprodukten formulerades ett dokument för projektets grafiska riktlinjer tidigt. Färgpaletterna ur denna redogörs för i figurerna **C.1–C.3**.

Hemsidans initiala prototyper visas till vänster i figur **C.4** och uppmaningen gentemot användaren att hålla sin mobila enhet i landskapsläge under spelets gång visas till höger i samma figur.



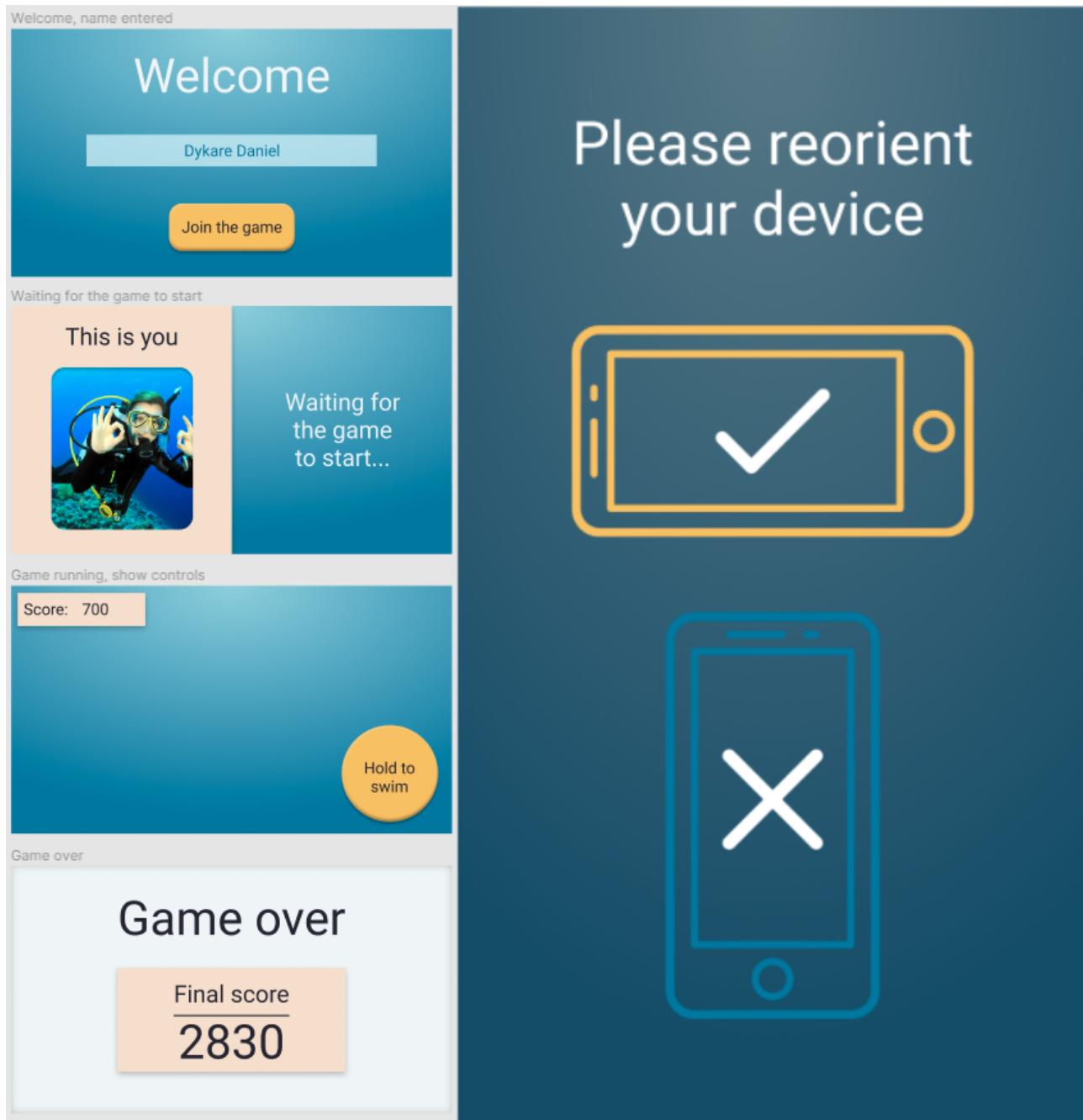
Figur C.1: Färgpalett för utvecklingsprojektets användargränssnitt.



Figur C.2: Färgpalett för bakgrundsfärger gällande framtagandet av konceptbilder.



Figur C.3: Färgpalett för förgrundsdetaljer gällande framtagandet av konceptbilder.



Figur C.4: Till vänster visas en första prototyp av hemsidans olika skärmar. Till höger visas uppmaningen till användaren att hålla sin *smartphone* i landskapsläge då styrningsgränssnittet ej är utformat för porträttläge.