ARCADA

ReactFast: Design och utveckling av ett reflexspel för mobil

Sebastian Berner

Lärdomsprov

Informationsteknik

2025

Lärdomsprov

Sebastian Berner

ReactFast: Design och utveckling av ett reflexspel för mobil

Yrkeshögskolan Arcada: Informationsteknik, 2025

Sammandrag:

Detta examensarbete beskriver utvecklingen av *ReactFast*, ett snabbt reflexbaserat spel för mobilen där spelaren drar fingret genom fördefinierade mönster för att uppnå så låg reaktionstid som möjligt. Arbetets syfte var dels att ta reda på vilka design- och teknikfaktorer som är centrala när man skapar dragbaserade mobilspel, dels att undersöka hur spelare upplever användbarhet och engagemang i *ReactFast*. De huvudsakliga forskningsfrågorna löd: 1. Vilka design- och utvecklingsaspekter är viktigast för ett snabbt, dragbaserat mobilspel? 2. Hur upplever spelare spelet i termer av användbarhet och engagemang?

Prototypen byggdes i Unity och kopplades ihop med Firebase för inloggning, datalagring och topplistan; spelet består av sju scener med responsiv dragmekanik och omedelbar ljud-/visuell återkoppling. Studien använde en mixed-methods-ansats: tjugo testare spelade TestFlight-versionen i 10–20 rundor; fjorton besvarade en enkät med fem skalfrågor 1–5 och tre öppna frågor. Medelvärdena visade att spelet upplevdes som roligt (4,27 /5) och intuitivt (4,36 /5), medan kvalitativa svar lyfte den ”smidiga” dragkänslan och tävlingsmomentet som största styrkor. Förbättringsönskemål gällde större målobjekt, en kort tutorial och fler mönster.

Slutsatserna pekar på att noggrant kalibrerad träffzon, omedelbar feedback och social konkurrens är starka drivkrafter för engagemang, men att variation i innehåll och tydlig introduktion är avgörande för långvarig spelglädje. Rekommenderade vidareutvecklingar är en valbar tutorial, fler banor, adaptiva träffzoner samt utökad insamling av spelardata för djupare analys.

Nyckelord:

mobilspel, draggest, reflexspel, speldesign, användarupplevelse, flowteori

Degree Thesis

Sebastian Berner

ReactFast: Design and development of a mobile reaction-based game

Arcada University of Applied Sciences: Information Technology, 2025

Abstract:

This thesis details the development of ReactFast, a fast-paced mobile game where players swipe through predefined patterns to achieve the lowest possible reaction time. The aim was partly to identify key design and technical factors important when creating swipe-based mobile games, and partly to examine how players experience usability and engagement in ReactFast. The research questions were: 1. Which design and development aspects are most important for a fast, swipe-driven mobile game? 2. How do players perceive the game in terms of usability and engagement?

The prototype was developed using Unity and connected with Firebase for login, data storage, and a leaderboard. The game includes seven scenes with responsive swipe mechanics and immediate audiovisual feedback. A mixed-methods study was conducted: twenty TestFlight participants played 10–20 rounds each, and fourteen completed a survey consisting of five scale questions and three open-ended questions. The average scores showed that participants experienced the game as enjoyable (4.27/5) and intuitive (4.36/5). Open-ended responses highlighted the smooth feel of the swipe mechanics and the competitive leaderboard as main strengths. Suggested improvements included larger targets, a brief tutorial, and more varied patterns.

The findings indicate that precisely tuned hitboxes, immediate feedback, and competition are strong drivers of engagement, while varied content and clear introduction are important for sustained enjoyment. Recommended future improvements include an optional tutorial, more levels, adaptive target zones, and expanded player data collection for deeper analysis.

Keywords:

mobile game, swipe gesture, reflex game, game design, user experience, flow theory

Innehåll

1 Inledning 6

2 Bakgrund och relaterade arbeten 7

2.1 Mobilspelsmarknaden i korthet 7

2.2 Reflexbaserade och dragbaserade mobilspel 8

2.3 Teoretiska ramverk: UX, Flow och MDA 8

2.3.1 Användarupplevelse (UX) och användbarhet 9

2.3.2 Flow-teori 9

2.3.3 MDA-ramverket 9

2.4 Koppling till ReactFast 10

2.5 Sammanfattning 11

3 Metod 11

3.1 Forskningsansats 11

3.2 Utveckling av spelet 12

3.2.1 Val av spelmotor och verktyg 12

3.2.2 Projektstruktur och scener 12

3.2.3 Integration av Firebase 13

3.3 Användarundersökning och test 14

3.3.1 Urval av testpersoner 14

3.3.2 Testformat 14

3.4 Insamlingsmetoder 14

3.4.1 Enkät 14

3.4.2 Speldata 15

3.5 Dataanalys 15

3.5.1 Kvantitativ analys 15

3.5.2 Kvalitativ analys 15

3.6 Etiska överväganden 15

3.7 Koppling till forskningsfrågorna 16

4 Implementation 17

4.1 Översikt av projektets struktur 18

4.2 StartUp Scene 19

4.3 Login Scene 21

4.4 SignUp Scene 24

4.5 Menu Scene 26

4.5.1 Profilvyn i Menu-scenen 28

4.6 Store Scene 30

4.7 Leaderboard Scene 33

4.8 Game Scene 35

4.8.1 GameController.cs 35

4.8.2 TouchController.cs 36

4.8.3 PatternManager.cs ... 37
4.9 Ljud och Musik ... 39
4.9.1 AudioManager ... 40
4.9.2 ButtonSoundManager ... 41
4.9.3 GameSoundManager ... 41

5 Resultat ... 43

5.1 Kvantitativa resultat ... 43
5.1.1 Analys av skalfrågorna ... 44
5.2 Kvalitativa resultat ... 45
5.2.1 Vad tyckte du var bäst? ... 45
5.2.2 Vad tyckte du var mindre bra, eller vad saknade du? ... 45
5.3 Sammanfattning av resultaten ... 46

6 Diskussion ... 46

6.1 Reflektion kring spelets design och mottagande ... 46
6.1.1 Positiv respons på dragmekaniken och tempot ... 46
6.1.2 Tävlingsmomentet via topplistan ... 47
6.2 Utmaningar och förbättringsförslag ... 48
6.2.1 Instruktioner och tutorial ... 48
6.2.2 Storlek och variation i objekt ... 48
6.2.3 Ojämnhet och "tur" ... 48
6.3 Koppling till forskningsfrågorna ... 49
6.3.1 Viktiga design- och utvecklingsaspekter ... 49
6.3.2 Upplevelse (användbarhet och engagemang) ... 49
6.4 Metodologiska begränsningar ... 49
6.4.1 Slutsatser och vidare arbete ... 50

7 Avslutande reflektioner och slutsatser ... 50

7.1 Centrala lärdomar och slutsatser ... 51
7.2 Begränsningar i arbetet ... 51
7.3 Förslag på vidare utveckling ... 52
7.4 Slutord ... 53

8 Källförteckning ... 54

9 Bilaga A – Källkod till ReactFast ... 56

1 Inledning

Mobilspelsmarknaden växer fortfarande snabbt och är idag en central del av spelindustrin (Newzoo 2021). Enligt Newzoo (2024) genererade mobilspelintäkterna cirka **92,6 miljarder USD**, vilket motsvarar nästan **49 % av den globala spelmarknaden**. I Finland har ett flertal utvecklare nått stora framgångar inom mobilspel, mycket tack vare ökad tillgång till smarttelefoner och en stark utvecklarkultur. För mig, som vill arbeta med spelutveckling, är det därför särskilt spännande att undersöka både tekniska och designmässiga frågor kring hur mobilspel utvecklas.

Jag har under en längre tid funderat på att utveckla ett eget spel, driven av personlig ambition. Idén till *ReactFast slog* mig när jag funderade på vad jag själv uppskattar i mobilspel: att det ska gå snabbt, vara enkelt att komma igång med och samtidigt erbjuda någon form av tävling. *ReactFast* går ut på att spelaren drar sitt finger genom olika mönster så fort som möjligt för att nå en så låg tid som möjligt. En inbyggd topplista gör att spelarna kan tävla mot varandra, vilket är motiverande. Det finns även andra snabba spel, som *Geometry Dash* eller *Color Switch*, där man främst trycker på skärmen och *Fruit Ninja*, som bygger på svepgester. Däremot saknade jag ett spel som har dragbaserade mönster som sin kärnmekanik.

I det här examensarbetet vill jag undersöka två centrala frågor:

1. **Vilka är de viktigaste design- och utvecklingsaspekterna vid skapandet av ett snabbt, dragbaserat mobilspel?**

2. **Hur upplever spelare *ReactFast* i termer av användbarhet och engagemang?**

Syftet är att undersöka hur man bygger ett mobilspel från grunden, med fokus på både speldesign och tekniska lösningar.

Jag vill dels lära mig mer om hur man skapar ett skalbart och användarvänligt spel, dels se hur spelare reagerar på dragbaserade rörelser i stället för traditionella pek- eller

tryckgester. Förhoppningen är att jag ska få en tydligare bild av hur man utvecklar ett mobilspel som är snabbt, intuitivt och engagerande.

En tydlig begränsning i projektet är att testgruppen för *ReactFast* sannolikt blir ganska liten. Ändå kan den feedback som samlas in ge viktiga insikter om hur spelet upplevs i praktiken. Jag planerar att använda en kort enkätundersökning för att samla in både kvalitativa och kvantitativa svar. Dessutom kommer jag att söka information online om vilka metoder som anses fungera bäst inom mobilspelutveckling.

Sammanfattningsvis kommer denna uppsats att visa hur *ReactFast* har skapats – från idé till en fungerande prototyp – samt hur teknikval, designbeslut och användarfeedback har format utvecklingsprocessen. I nästa kapitel går jag igenom relevant bakgrund och tidigare arbeten som berör snabba spel och dragbaserad mekanik. Därefter beskriver jag spelets utveckling i detalj (metod, implementering), för att sedan presentera och diskutera resultatet utifrån mina forskningsfrågor. Avslutningsvis sammanfattar jag de viktigaste slutsatserna och ger förslag på framtida förbättringar.

Tillgänglighet. ReactFast distribueras för närvarande som en intern TestFlight-build; testare bjuds in individuellt via App Store Connect. Testet är kostnadsfritt och insamlade data används enbart för detta examensarbete. När feedbacken har integrerats är målet att publicera en förbättrad gratisversion på App Store under en MIT-licens.

2 Bakgrund och relaterade arbeten

2.1 Mobilspelsmarknaden i korthet

Under de senaste fem åren har mobilspelsmarknaden vuxit med 26,2 % och utgör idag 49 % av den globala spelindustrin (Newzoo 2021, 2024). Enligt Newzoo (2024) är marknaden enorm och fortsätter att växa, med de mest nedladdade spelen som Supercells titlar som genererar intäkter på miljardnivån (Statista 2024,2025).

Enligt Zippia (2023) beror tillväxten på ökade intäkter, fler nedladdningar och en växande användarbas. Den snabba spridningen av smarttelefoner och förbättrad hårdvara

möjliggör mer avancerade och bättre spelupplevelser, vilket ytterligare bidrar till marknadens expansion.

Finland har etablerat sig som en ledande aktör inom mobilspelsutveckling, med företag som Supercell och Rovio i spetsen. Dessa bolag har skapat internationella succéer och stärkt landets rykte som en innovationshubb för mobilspel.

2.2 Reflexbaserade och dragbaserade mobilspel

En stor underkategori inom mobilspel är reflexbaserade spel, där spelarens snabba reaktioner är avgörande för att klara av utmaningar. Exempelvis använder *Fruit Ninja* svep-/dragbaserade gester för att ”skära” frukter, medan titlar som *Geometry Dash* och *Color Switch* ofta baserar sig på tryckbaserade kontroller.

En studie av Sanders (2017) jämför olika kontrollmetoder på smarttelefoner och visar att gester kan upplevas som mer intuitiva än enkla tryck. Samtidigt visar forskning att om gester inte är tydligt definierade eller där återkopplingen är svag kan det leda till felregistreringar och frustration (Zubair och Muhammad, 2021).

För just reflexbaserade spel är feedback och kontrollmetod centrala (Thai, 2022). Eftersom många mobilspelare förväntar sig snabba sessioner, behöver man göra kontrollen så att den känns direkt och responsiv. I nästa avsnitt (2.3) diskuteras teoretiska ramverk som är särskilt relevanta för att förklara hur användarupplevelsen påverkas av reflexbaserade mekaniker och varför dragbaserade gester kan öka engagemanget.

2.3 Teoretiska ramverk: UX, Flow och MDA

För att analysera och motivera designen av reflexbaserade mobilspel används flera teoretiska perspektiv: UX (User Experience), Flow-teorin och MDA-ramverket. Tillsammans ger dessa en helhetssyn på hur man kan skapa en utmärkt spelupplevelse. och förklara varför vissa spel (som dragbaserade reflexspel) kan kännas extra engagerande.

2.3.1 Användarupplevelse (UX) och användbarhet

UX handlar om hur spelare upplever spelet både funktionellt och känslomässigt. Enligt Thai (2022) är det i mobilspel med korta sessioner särskilt viktigt att ha enkla touchkontroller och omedelbar feedback, något som även är centralt för *ReactFast.* I reflexbaserade spel kan otydlig kontrollmetod leda till ökad frustration, medan en tydlig och responsiv geststyrning kan höja spelglädjen (Sanders, 2017).

2.3.2 Flow-teori

Flow-teorin, såsom den beskrivs av Cowley (2008) i *Toward an Understanding of Flow in Video Games*, behandlar hur spelmekanik och estetik kan utformas för att skapa optimala spelupplevelser där utmaning och skicklighet är i balans. Enligt Cowley (2008) uppstår ett flow-tillstånd när spelaren engageras i en kontinuerlig cykel av utmaningar som är nya men samtidigt relaterbara till tidigare erfarenheter, vilket möjliggör ett tillstånd av djupkoncentration och engagemang. Detta flow-tillstånd ursprungligen definierat av Csikszentmihalyi (1990), är centralt för att förstå varför vissa spel blir särskilt engagerande och beroendeframkallande.

I utvecklingen av *ReactFast* har dessa principer eftersträvats genom att drag ger direkt visuell och ljudfeedback, samtidigt som
svårighetsgraden gradvis ökar. På detta sätt stimuleras spelaren att upprätthålla fokus och kontinuerligt sträva efter att förbättra sin prestation, vilket stödjer den dynamiska interaktion mellan spel och spelare som enligt Cowley är avgörande för att uppnå ett hållbart flow-tillstånd.

2.3.3 MDA-ramverket

MDA-ramverket (Mechanics, Dynamics, Aesthetics) är ett analytiskt och metodiskt verktyg för speldesign och spelanalys (Hunicke, LeBlanc och Zubek, 2004). Ramverket delar upp ett spel i tre tydliga abstraktionsnivåer:

Mechanics avser de grundläggande reglerna och komponenterna som styr spelarens möjliga handlingar och interaktioner. För *ReactFast* innebär detta exempelvis reglerna för hur drag-gester registreras, hur mönster genereras och hur reaktionstiden beräknas.

Dynamics beskriver det beteende och de interaktioner som uppstår när spelaren agerar inom dessa regler. I *ReactFast* kan dynamiken bestå av en stegvis ökande svårighetsgrad samt det tävlingsmoment som uppstår genom spelets toplista.

Aesthetics avser de känslomässiga upplevelser spelet väcker hos spelaren. *För ReactFast* kan dessa estetiska mål vara spänning, intensiv koncentration (flow) och engagemang, vilket förstärks av spelets ljud, visuella feedback och direkt respons från geststyrningen.

MDA-ramverket är särskilt användbart för att analysera och designa reflexbaserade spel som *ReactFast*, eftersom det gör det möjligt att tydligt identifiera hur specifika mekaniska förändringar påverkar den övergripande spelupplevelsen.

2.4 Koppling till ReactFast

Trots att reflexbaserade spel är väl etablerade, använder många titlar enbart tryckbaserade kontroller. Medan *Fruit Ninja* använder svepgester, bygger titlar som *Geometry Dash* och *Color Switch* på enklare tryck. *ReactFast* erbjuder en unik vinkel genom att implementera en dragbaserad mekanik där spelarens reaktionstid mäts när de drar fingret genom fördefinierade mönster.

Studier visar att en tydlig koppling mellan gester och specifika spelhandlingar kan bidra till en mer intuitiv och tydlig spelupplevelse (Zubair och Muhammad, 2021). Engagemanget höjs ytterligare genom ett tävlingsmoment i form av en topplista vilket enligt Butler (2013) kan öka spelarnas motivation och i förlängningen stärka deras flow (Csikszentmihalyi, 1990).

Genom att kombinera en snabb, dragbaserad Mechanic (MDA) och en tävlingsinriktad Dynamic (topplista) strävar *ReactFast* efter att skapa en spelupplevelse som känns in-

tensiv och naturligt flytande. Samtidigt följer designen centrala UX-principer om omedelbar återkoppling – varje gång spelaren drar fingret genom ett mönster ska spelet svara direkt med responsiva ljud- och visuella effekter, vilket ger en smidig och belönande upplevelse för spelaren. Detta ligger i linje med Korhonen, Holm och Heikkinen (2007), som betonar hur ljudfeedback i realtid kan förstärka användarens upplevelse, öka inlevelsen och göra interaktionen både effektiv och underhållande.

2.5 Sammanfattning

Sammanfattningsvis är mobilspelsmarknaden enorm och fortsätter växa (newzoo 2021, 2024), vilket skapar en gynnsam miljö för innovation. Reflexbaserade spel utnyttjar oftast tryck- eller svepgester, men dragbaserade interaktioner erbjuder en mer intuitiv och omedelbar feedback som kan leda till högre engagemang (Sanders 2017). Genom att tillämpa Flow-teorin (Csikszentmihalyi, 1990) och MDA-ramverket (Hunicke, LeBlanc och Zubek, 2004) kan man bättre förstå hur designval – som dragbaserade rörelser och tävlingsmoment – påverkar spelupplevelsen. *ReactFast* fyller därmed en lucka i forskningen genom att kombinera dragbaserad interaktion med ett tävlingsmoment (Toplista), vilket potentiellt kan ge ett unikt och beroendeframkallande flow.

3 Metod

I detta kapitel förklaras hur min undersökning genomförs, både när det gäller utvecklingen av spelet *ReactFast* och hur användarundersökningen är planerad. Syftet är att kunna svara på följande två forskningsfrågor:

1. Vilka är de viktigaste design- och utvecklingsaspekterna vid skapandet av ett snabbt, dragbaserat mobilspel?

2. Hur upplever spelare *ReactFast* i termer av användbarhet och engagemang?

3.1 Forskningsansats

Eftersom den första frågan fokuserar på tekniska och designmässiga aspekter, medan den andra frågan rör användarupplevelse, används en blandad metod (mixed methods). Detta innebär:

En teoretisk del där tidigare forskning och ramverk (Flow, MDA, UX) studerats för att identifiera centrala faktorer i reflexbaserade mobilspel.

En empirisk del bestående av en speltestperiod samt en enkät med både slutna och öppna frågor.

Denna kombination ger möjlighet att dels beskriva hur spelet byggts och vilka designval som gjorts (första forskningsfrågan), dels undersöka hur spelarna faktiskt upplever spelet i praktiken (andra forskningsfrågan).

3.2 Utveckling av spelet

3.2.1 Val av spelmotor och verktyg

Spelet *ReactFast* utvecklas i spelmotorn Unity, framförallt för att:

- Unity har en stor community och många färdiga funktioner för 2D mobilspel
- Unity stödjer enkel export till både Android och iOS.
- Effektiv och flexibel programmering i C#, ett modernt objektorienterat programmeringsspråk.

Dessutom används:

- Firebase för databas och autentisering (e-post/lösenord eller gästkonto).
- Egenproducerade resurser (assets), till exempel, Skript, ljudfiler, sprites för animerade bakgrunder (ex. planeter, meteorer) och UI-knappar.

3.2.2 Projektstruktur och scener

För att hålla ordning på spelets olika delar används begreppet "scen" i Unity. En scen kan beskrivas som en separat del eller skärm i spelet, som innehåller specifika spelobjekt, gränssnitt och funktioner. Spelet *ReactFast* består av flera scener, där varje scen hanterar en tydlig uppgift:

1. **LoginScene:** Användaren kan logga in eller köra gästläge.

2. **SignUpScene**: Möjlighet att skapa nytt konto (användarnamn, ålder, etc.).
3. **MenuScene**: Huvudmeny, där man ser sin profil, kan välja *trail*, gå till Store, topplistan eller starta spelet.
4. **StoreScene**: Köpa nya *trails* för in-game-valuta.
5. **LeaderboardScene**: Topp 20-lista baserat på spelarnas bästa tider.
6. **GameScene**: Själva speldelen där spelaren drar fingret genom 10 mönster. Medeltiden beräknas sedan och jämförs med spelarens tidigare PB (personbästa).

3.2.3 Integration av Firebase

Firebase används i *ReactFast* främst för två saker:

- **Inloggning:** Spelare kan logga in med e-post och lösenord eller välja att spela som gäst utan att behöva registrera sig.
- **Spara användardata:** Spelet sparar viktig information om spelaren, till exempel ålder, bästa resultat, valuta och vilka trails spelaren har köpt och använder.

En **trail** är en effekt som visas bakom spelarens finger när man drar över skärmen. Spelaren kan köpa nya trails i spelets butik med spelets egna pengar (virtuell valuta). När man köper en trail sparas detta i databasen, och sedan kan spelaren använda den trailen i spelet.

Firebase är lätt att använda tillsammans med Unity eftersom det finns färdiga **SDK:er**. Ett SDK är ett paket med verktyg och funktioner som gör det enklare att lägga till olika tjänster i ett spel, till exempel att logga in och att spara data online.

I Firebase finns två viktiga områden där data sparas:

/store: Här sparas trails som spelaren kan köpa. Varje trail har ett namn, ett pris och en sprite. En sprite är en enkel 2D-bild som används för att visa saker i spelet, i det här fallet själva trail-effekten.

/users: Här sparas information om varje spelare, till exempel deras bästa resultat, hur mycket pengar de har i spelet och vilka trails de har köpt.

Allt detta sker automatiskt och enkelt genom Firebase SDK för Unity, som innehåller färdiga funktioner för inloggning och att spara data direkt online.

3.3 Användarundersökning och test

3.3.1 Urval av testpersoner

Spelet testas av totalt cirka 20 personer, främst via ett bekvämlighetsurval bestående av vänner och bekanta i åldrarna 20–30 år. Detta urval begränsar generaliserbarheten, men ger en första indikation på hur spelet upplevs av en yngre målgrupp.

3.3.2 Testformat

- **Distans:** Spelet laddas ner via TestFlight (iOS).
- **Speltid:** Varje person ska spela *ReactFast* för cirka (10–20) rundor, vilket tar cirka 5–10 minuter.
- **Instruktioner:** Inga formella tutorial-moment eller detaljerade instruktioner implementerades i denna version av spelet. Spelarna uppmuntrades endast att försöka få bästa möjliga tid och tävla på topplistan.
- **Feedback:** Efter spelets test uppmanas spelarna att fylla i en enkät (se nedan).

3.4 Insamlingsmetoder

3.4.1 Enkät

En enkät i **Google Forms** skickas ut till testarna. Den består av:

- **Fem skalfrågor** (där 1 = negativ/låg och 5 = mycket positiv/hög) om hur roligt spelet var, hur tydlig dragmekaniken upplevdes, om spelet uppfattades som för svårt eller för lätt, hur motiverade deltagarna var att spela fler rundor samt om de kunde tänka sig att rekommendera spelet.

- **Tre öppna frågor** om vad som var bäst, vad som saknades/var mindre bra och övriga kommentarer.

3.4.2 Speldata

- **Genomsnittstider:** Varje gång en spelare slutförde 10 mönster beräknades en genomsnittlig reaktionstid, som sparas i Firebase.

- **Uppdatering av topplistan**: Den bästa uppmätta tiden för varje användare uppdaterades och kan ses av alla i spelet.

3.5 Dataanalys

3.5.1 Kvantitativ analys

Enkätsvar: För de fem skalfrågorna (betyg 1–5) beräknades medelvärde (M) och standardavvikelse (SD).

3.5.2 Kvalitativ analys

- **Tematisk genomgång**: De öppna enkätsvaren (”Vad var bäst?”, ”Vad saknades?” m.fl.) delades in i övergripande teman, såsom ”positiva aspekter” (dragmekanik, tävlingsmoment, design) och ”förbättringsförslag” (mer variation, större objekt, tutorial).

- **Koppling till teori**: Kommentarerna jämfördes sedan med teoretiska perspektiv (t.ex. Flow, MDA, UX) för att se hur spelarnas upplevelser stämmer överens med centrala principer för reflexbaserade mobilspel.

3.6 Etiska överväganden

- Informerat samtycke: Alla deltagare fick information om att deras speltest var frivilligt och att svaren skulle användas i ett examensarbete.

- Anonymitet: Enkäten genomfördes anonymt. I Firebase lagrades dock e-postadresser för inloggning, men endast för internt bruk.

- Dataskydd: Inga känsliga personuppgifter (t.ex. personnummer) samlades in. Endast spelarnas tider och begränsade profiluppgifter (t.ex. användarnamn, valuta) hanterades i databasen.

3.7 Koppling till forskningsfrågorna

1. **Design- och utvecklingsaspekter**

- Genom att studera litteratur och dokumentera den egna byggprocessen i Unity och Firebase kan viktiga faktorer för ett dragbaserat, snabbt mobilspel identifieras.
- Feedback från spelare hjälper dessutom till att avslöja vilka beslut som fungerat bra eller dåligt i praktiken.

2. **Upplevelse (användbarhet och engagemang)**

- Enkätsvar och öppen feedback ger en bred bild av hur spelare upplever *React-Fast.*
- De öppna frågorna belyser vad spelarna uppskattade mest (t.ex. snabb feedback, tävlingsmoment) och vad de saknade (mer variation, tutorial), vilket tydligt relaterar till användarupplevelse och engagemangsnivå.

Med denna metod är det alltså möjligt att både analysera hur spelet konstrueras och hur det faktiskt upplevs av spelarna.

4 Implementation

I detta kapitel presenteras den praktiska utvecklingen av spelet *ReactFast*, från projektets struktur i Unity till viktiga kodavsnitt för inloggning, spelmekanik och datalagring. Kapitlet syftar till att visa hur designidéer och tekniska lösningar tillämpas i praktik, med fokus på dragbaserade gester, Firebase-integration och användarvänlig UI/UX.

I föregående kapitel (Metod och genomförande) beskrevs hur utvecklingsprocessen planerades och hur användartester skulle genomföras. Nu fördjupas perspektivet till den faktiska koden och uppbyggnaden av spelet. Implementationens innehåll kan sammanfattas i tre delar:

1. **Scenbaserad struktur i Unity**: Genomgång av de viktigaste scenerna – Start-Up, Login, SignUp, Menu, Store, Leaderboard och Game – samt skript som styr spelets flow och logik.

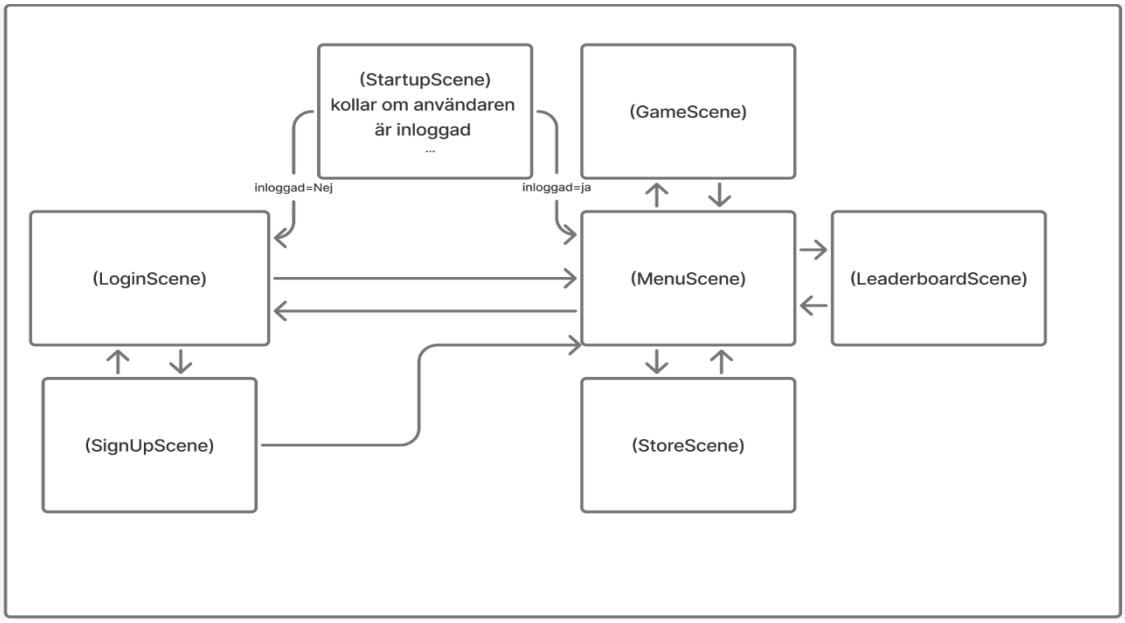
2. **Integrering med Firebase:** Hur inloggning, datalagring och gästkonton hanteras och hur spelare knyts till varsin post i databasen för att spara personbästa, val av "trail" och valuta.

3. **Interaktionsdesign och spelmekanik:** Hur dragbaserade gester och tidtagning implementeras i Game-scenen, vilka Unity-verktyg som används och hur feedback visas för spelaren.

Målet är att ge en tydlig överblick av varför koden ser ut som den gör och hur de tekniska valen stödjer spelets designmål: att erbjuda ett snabbt, intuitivt och engagerande mobilspel. För detaljerad teknisk implementation och fullständig källkod, se Bilaga A. Nedan följer först en översikt av scenstrukturen, därefter presenteras varje scen mer ingående.

4.1 Översikt av projektets struktur

Projektet består av sju huvudsakliga scener i Unity



Varje scen ansvarar för en tydlig del av spelupplevelsen:

StartUp: Initierar Firebase och kontrollerar om en användare redan är inloggad.

Login: Erbjuder inloggning via e-post eller anonymt gästkonto.

SignUp: Låter nya användare skapa ett konto med användarnamn och ålder.

Menu: Huvudmeny där spelaren kan se sin profil, välja trail, gå till Store eller topplistan (Leaderboard), samt starta själva spelet.

Store: Butik där spelaren kan köpa olika "trails" för att ändra effekt på dragningen.

Leaderboard: Visar toppresultat (lägstatiden) och låter spelare jämföra sina resultat.

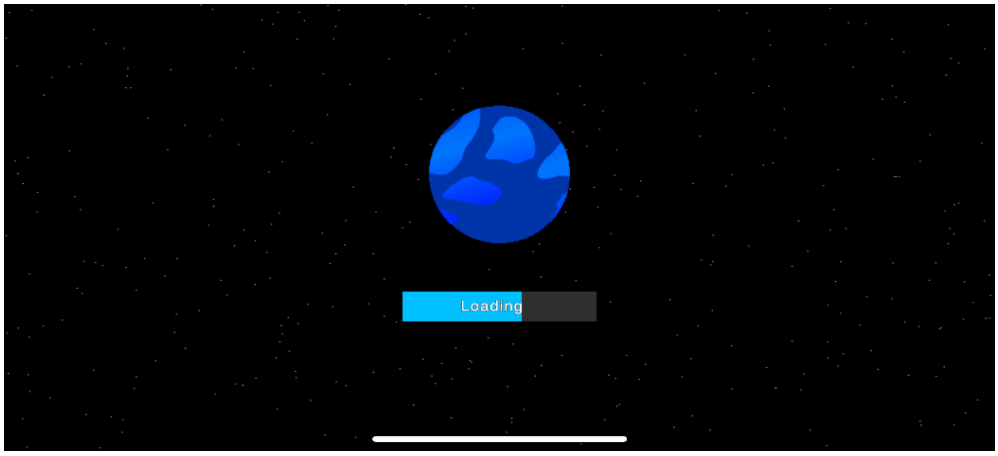
Game: Spelscenen, där användaren drar fingret genom mönster och försöker få så låg tid som möjligt.

Varje scen har ett eller flera skript som förklaras i de kommande avsnitten. Här används C# för logik och Unity för att arrangera UI-element, spelfysik och resurser (sprites, ljud). För att stödja enkel vidareutveckling är koden uppdelad i moduler som ansvarar för varsin funktion, exempelvis LoginManager.cs, GameManager.cs, och Startuploader.cs I nästa avsnitt inleds en fördjupad beskrivning av varje scen och dess nyckelskript, inloggning och kontoskapande först, följt av butik, leaderboard och till sist själva spelscenen.

4.2 StartUp Scene

Syfte

StartUp Scene är den första scenen som laddas när spelet startar. Här kontrolleras om Firebase är klart att användas och om en användare redan är inloggad hoppar spelet direkt till Menu; annars får spelaren logga in via Login-scenen.



***Figur 4.2.1**. StartUp Scene – spelscenen som initierar Firebase och styr användaren vidare.*

Viktigt skript: StartupLoader.cs

StartupLoader.cs ansvarar för:

Initiering av Firebase och autentisering.

Väntan på en laddningsanimation innan nästa scen laddas.

Scenbyte baserat på om en användare är inloggad eller inte.

```
void Start()
{
    // Initiera Firebase och kontrollera beroenden
    Fire-baseApp.CheckAndFixDependenciesAsync().ContinueWithOnMainThread(task
=>  {
        if (task.Result == DependencyStatus.Available)
            auth = FirebaseAuth.DefaultInstance;
        else
            Debug.LogError("Firebase initialization error: " + task.Exception);
    });

    // Starta en korutin som väntar på laddningsanimationen
    StartCoroutine(WaitAndSwitchScene());
}

IEnumerator WaitAndSwitchScene()
{
    while (!loadingBarController.isFinished)
        yield return null;

    yield return new WaitForSeconds(0.5f);

    // Välj nästa scen baserat på inloggning
    SceneManager.LoadScene(auth.CurrentUser != null ? "Menu" : "LoginScene");
}
```

Kodexempel 4.2.1: *Förkortat utdrag ur StartupLoader.cs.För hela koden, se GitHub-länken i Bilaga A.*

Förklaring

Initiering: Metoden *CheckAndFixDependenciesAsync()* säkerställer att Firebase är redo.

Auth-instans: Om Firebase fungerar, skapas auth = FirebaseAuth.DefaultInstance; vilket senare låter oss avgöra om en användare är inloggad.

Korutin: I *WaitAndSwitchScene()* väntar spelet på en laddningsanimation (!loadingBarController.isFinished). Därefter fördröjs scenbytet en kort stund för en mjuk övergång.

Scenbytet: Beroende på om auth.CurrentUser är null (ingen inloggning) eller inte (redan inloggad), går spelet till LoginScene eller Menu.

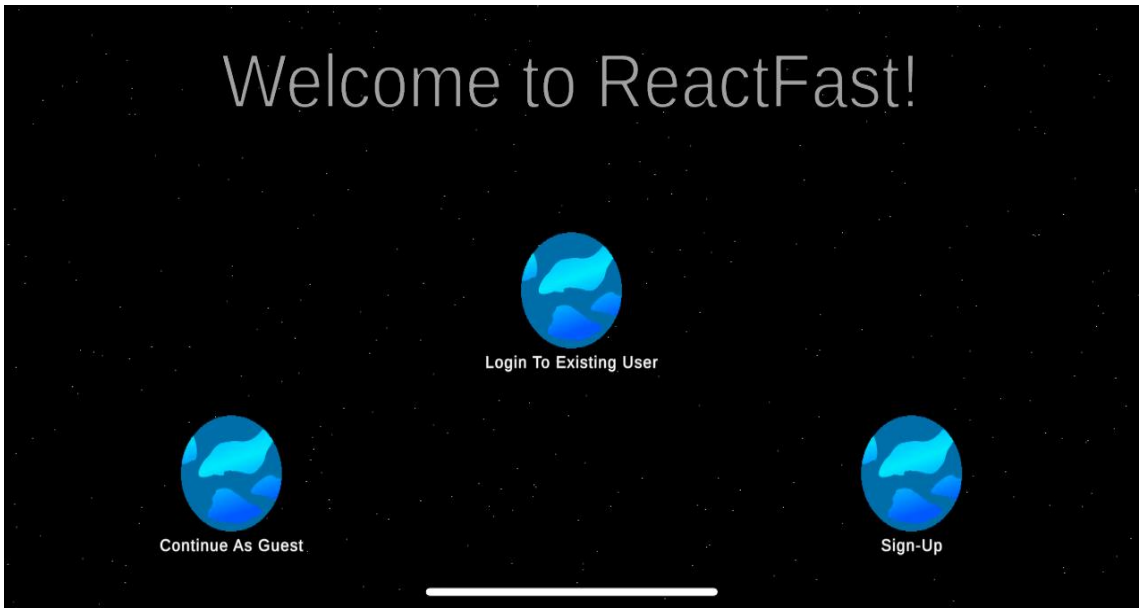
Reflektion

Att lägga Firebase-initialisering i en separat scen håller övriga scener avskalade från uppstartslogik. Under inladdningen kan användaren se en laddningsbar, vilket förbättrar upplevelsen jämfört med en plötslig svart skärm.

4.3 Login Scene

Syfte

I Login-scenen kan användaren logga in med e-post och lösenord, fortsätta som gäst eller navigera till SignUpScene för att skapa ett nytt konto. Den innehåller animerade knappar och ett välkomnande meddelande med skrivmaskinseffekt, vilket ger en mer levande användarupplevelse.



***Figur 4.3.1**. Login Scene – initial vy med valen "Login To Existing User", "Continue As Guest" och "Sign-Up".*

Huvudsakligt skript: *LoginManager.cs*

LoginManager ansvarar för:

- Inloggning via Firebase med e-post och lösenord.
- Gästinloggning om spelaren vill prova spelet utan att registrera sig.
- Navigering till SignUp-scenen.
- Animering av knappar (DOTween) samt ett skrivmaskinsliknande välkomst-meddelande.

```
void OnLoginButtonClicked()
{
    string email = emailInput.text.Trim();
    string password = passwordInput.text;

    // Enkel validering
    if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
    {
        statusText.text = "Please enter both email and pass-word.";
        return;
    }

    auth.SignInWithEmailAndPasswordAsync(email, password)
        .ContinueWithOnMainThread(task =>
        {
            if (task.IsCanceled || task.IsFaulted)
            {
                Debug.LogError("Login error: " + task.Exception);
                statusText.text = "Login error.";
                return;
            }
            // Lyckad inloggning
            SceneManager.LoadScene("Menu");
        });
}
```

***Kodexempel 4.3.1**: Förkortat utdrag ur LoginManager.cs, som visar e-postinloggning via Firebase (Se Bilaga A för fullständig kod).*

Funktioner i korthet

- **E-postinloggning:**
  Med *SignInWithEmailAndPasswordAsync()* verifierar spelet användaren inloggningsuppgifter i Firebase. Vid lyckad inloggning laddas Menu-scenen.
- **Gästinloggning:**
  Via *OnContinueAsGuestClicked()* skapas ett anonymt konto med ett genererat gästnamn och en startvaluta. Eftersom Firebase stödjer anonym inloggning slipper spelaren lämna e-postuppgifter.

Navigation till SignUp:

- Med *OnSignUpButtonClicked()* byter scenen till SignUpScene, så att en ny användare kan registrera sig ordentligt.
- **Knappanimering** (DOTween):
  När man trycker ner en knapp (t.ex. "Login"), skalar skriptet ner knappen lätt, och när man släpper, skalar det tillbaka upp och anropar önskad metod:

```
private void OnButtonPress(Button button)
{
    button.transform.DOScale(0.9f, 0.1f);
}
```

Kodexempel 4.3.2 *Förkortad kod för knappanimationer*

Välkomstmeddelande:

- En korutin (TypeTextEffect) skriver ut texten bokstav för bokstav och låter färgen sakta tona in och ut. Det gör login scenen lite mer levande.

Reflektion

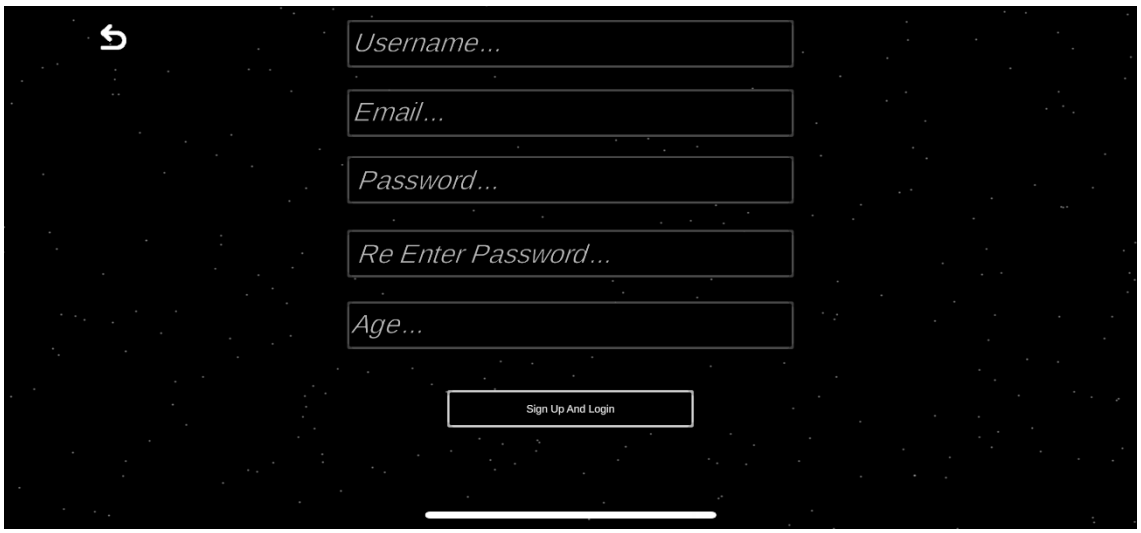
Login-scenen fokuserar på enkelhet och tydlighet för att sänka tröskeln för nya spelare. Enligt Hogue (2023) mer än 67 % av användarna överger sin registre-

ring om de stöter på komplikationer, vilket understryker vikten av att hålla processen så smidig som möjligt. Genom att erbjuda både registrerad och anonym inloggning ökar spelet chansen att fler faktiskt testar spelet direkt, utan att nödvändigtvis skapa konto. Dessutom lyfts upplevelsen med animationer och typningseffekt, vilket gör att spelet känns mer välkommande redan från början

4.4 SignUp Scene

Syfte

I SignUp-scenen kan en ny användare registrera sig genom att ange e-post, lösenord, användarnamn och ålder. Efter registrering sparas användardata i Firebase, och spelaren loggas automatiskt in och skickas till Menu-scenen.



Figur 4.4.1 *SignUp Scene – spelets registreringsskärm.*

Huvudsakligt skript: SignUpManager.cs

SignUpManager ansvarar för:

- Validering av inmatade fält (lösenord, ålder etc.).
- Kontoskapande via FirebaseAuth (CreateUserWithEmailAndPasswordAsync).
- Lagring av extradata (t.ex. username, bestResult, valuta) i Realtime Database.
- Navigering tillbaka till LoginScene om användaren ångrar sig.

```
void OnSubmitButtonClicked()
{
    string email = emailInput.text.Trim();
    string password = passwordInput.text;
    string confirmPassword = confirmPasswordInput.text;

    // Enkel validering
    if (password != confirmPassword)
    {
        statusText.text = "Passwords do not match.";
        return;
    }

    auth.CreateUserWithEmailAndPasswordAsync(email, password)
        .ContinueWithOnMainThread(task =>
        {
            if (task.IsFaulted)
            {
                Debug.LogError("Sign-Up error: " + task.Exception);
                statusText.text = "Sign-Up error.";
                return;
            }

            // Lyckad registrering
     FirebaseUser newUser = task.Result.User;StoreUserDataAndLogin
     (newUser.UserId, usernameInput.text,
      int.Parse(ageInput.text));
    });
}
```

***Kodexempel 4.4.1**: Förkortat utdrag ur SignUpManager.cs. (Se Bilaga A för hela koden.)*

Funktioner i korthet

- Validering: Koden ser till att lösenorden matchar och att åldern är inom rimliga gränser.
- Kontoskapande: *CreateUserWithEmailAndPasswordAsync()* anropar FirebaseAuth för att skapa ett nytt konto.
- Lagring av extradata: Efter kontots skapande kallas *StoreUserDataAndLogin()*, som sparar till exempel username, bestResult och valuta i users/{userId} i databasen.

- Automatisk inloggning: Användaren skickas direkt vidare till Menu-scenen efter en lyckad registrering.
- Tillbaka-knapp: Genom *OnBackButtonClicked()* kan man återvända till Login-Scene om man redan har ett konto.

Reflektion

SignUp-scenen erbjuder en smidig onboarding för nya spelare. Genom att spara extra-data (användarnamn, trail, valuta) får varje spelare sin egen profil, vilket skapar en mer personlig spelupplevelse. Att spelaren automatiskt loggas in efter registrering minskar friktionen ytterligare, principer enligt Hogue (2023) ökar chansen att nya användare stannar kvar i spelet.

4.5 Menu Scene

Syfte

I Menu-scenen samlas de viktigaste valen för spelaren:

- Starta spelet (”SinglePlayer”).
- Öppna StoreScene för att köpa nya trails.
- Öppna LeaderboardScene för att se topplistan.
- Visa och ändra sin profil (genom en separat vy i samma scen).

Figur 4.5.1. *Menu Scene – spelets Meny-skärm*

Huvudsakligtskript: MenuController.cs

MenuController styr övergångar mellan olika scener och ansvarar för knappanimeringar (DOTween).

```
private IEnumerator DelayedSceneTransition(Button button)
{
    yield return new WaitForSeconds(0.2f);
    DisableAllUI();

    switch (button.name)
    {
        case "SinglePlayerButton":
            StartSinglePlayer();
            break;
        case "StoreButton":
            OpenStore();
            break;
        case "LeaderboardButton":
            OpenLeaderboard();
            break;
        case "ProfileButton":
            OpenProfile();
            break;
        default:
            Debug.LogWarning("No action assigned to this but-ton");
            break;
    }
}
```

***Kodexempel 4.5.1**: Metod som öppnar rätt scen baserat på knappnamn ur MenuController.cs. (För fullständig kod, se Bilaga A).*

Funktioner i korthet

- **Scenval via knappar**

  När spelaren släpper en knapp kallas *DelayedSceneTransition()*, vilket ger en animerad fördröjning (0.2s) innan rätt scen laddas (Store, Leaderboard, osv.).
- **Knappanimationer**

  Knappar ”skalas” ned när spelaren trycker ned dem och skalas tillbaka vid släpp. Menyn ”zoomar ut” innan scenbytet för en mjuk övergång.

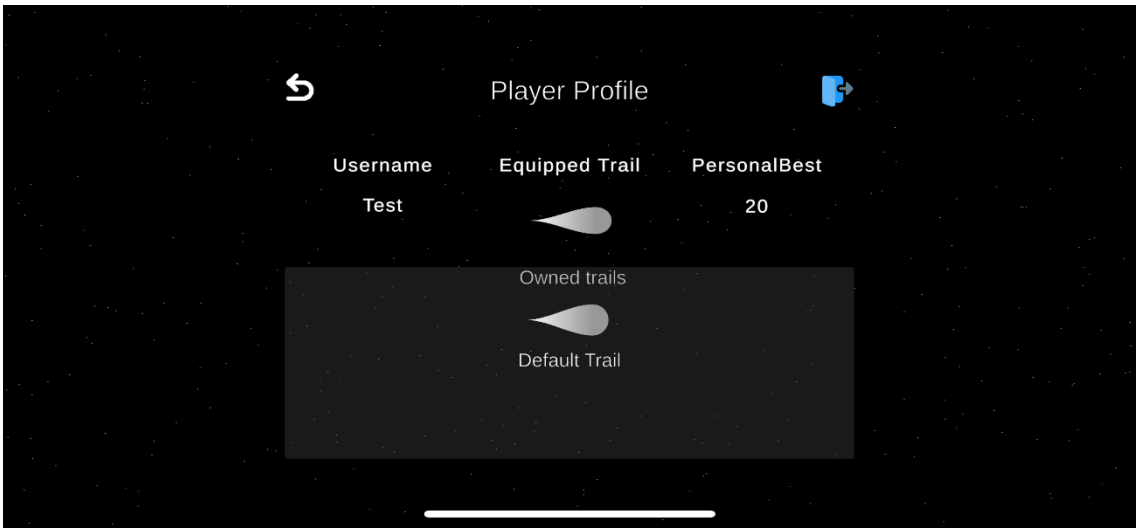
- **Profilvy**
  OpenProfile() döljer huvudmenyn och visar en separat canvas för spelarens profilinställningar (hanteras i ProfileManager).

Reflektion

Menu-scenen blir spelets **”huvudmeny”** som spelaren ofta återkommer till. Genom animerade övergångar och knappar skapas en **mer responsiv** känsla (DOTween), samtidigt som spelaren snabbt hittar de viktigaste alternativen.

4.5.1 Profilvyn i Menu-scenen

När man klickar på ProfileButton visas en under-canvas med spelarens namn, bästa resultat och en lista över ägda trails. Detta hanteras av ProfileManager.



Figur 4.5.2. *Profilvy – vy som visar spelarens användarnamn,” valda trail” och personliga rekord.*

Huvudsakligt skript: ProfileManager.cs

ProfileManager hämtar spelarens data från Firebase och uppdaterar UI-elementen.

```
void LoadUserProfile()
{
    FirebaseUser currentUser = FirebaseAuth.DefaultInstance.CurrentUser;
    if (currentUser == null) return;

    var userRef = FirebaseDatabase.DefaultInstance
        .GetReference("users").Child(currentUser.UserId);

    userRef.GetValueAsync().ContinueWithOnMainThread(task =>
    {
        if (task.IsCompleted && task.Result.Exists)
        {
            // T.ex. username, bestResult, currency, equippedTrail
            string username = task.Result.Child("username").Value.ToString();
            usernameText.text = username;
        }
    });
}
```

Kodexempel 4.5.2: *Nedkortat utdrag ur ProfileManager.cs. för fullständig kod se Bilaga A.*

Funktioner i korthet

Data från Firebase

LoadUserProfile() hämtar en snapshot av användardata, som användarnamn, bestResult och vilka trails man äger.

Visa och byta trail

Metoder som *DisplayOwnedTrails()* och EquipTrail(trailId) visar spelarens trails och låter hen byta aktiv trail.

Default-trail

Om man inte äger andra trails används en DefaultTrailSprite.

Stäng-knapp

OnCloseButtonClicked() stänger profilvyn och går tillbaks till huvudmenyn.

Reflektion

Att hålla profilvyn i samma scen som menyn (i stället för en separat scen) gör laddningstiden kortare och övergången lättare. Eftersom all data sparas i Firebase kan spelaren byta apparat eller starta om spelet utan att förlora sina framsteg så länge hen loggar in med sin e-postadress och sitt lösenord

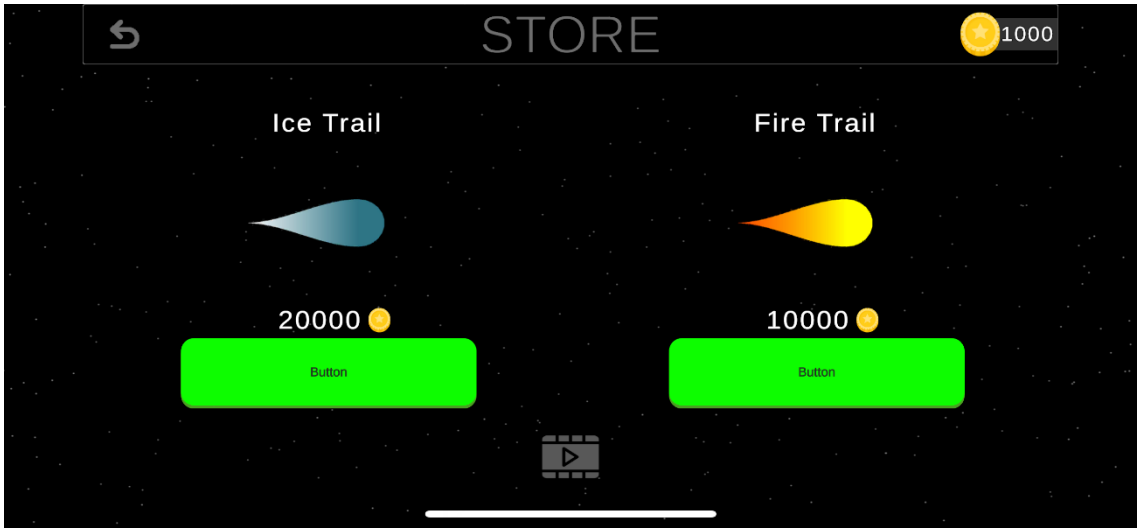
Sammanfattning för Menu Scene

Menu-scenen, inklusive ProfileManager, fungerar som en ”huvudhub” för spelaren. Här kan man starta spelet, se topplistan, köpa trails och ändra sin profil utan att behöva gå via flera separata scener. Genom att kombinera DOTween-animeringar, explosionseffekter och Firebase-integration får menyn en livlig känsla och gör det enkelt att hålla koll på spelarens personliga framsteg.

4.6 Store Scene

Syfte

I Store-scenen kan spelaren köpa nya “trails” (visuella effekter) med en virtuell valuta. Scenen hämtar spelarens nuvarande valuta och ägda trails från Firebase och låter spelaren göra köp, varvid databasen uppdateras.



Figur 5.6.1. *Store Scene – butiksvyn där spelaren ser tillgängliga trails med pris och sin nuvarande valuta.*

Huvudsakligt skript: StoreManager.cs

StoreManager ansvarar för:

- Ladda ägda trails och valuta från Firebase.
- Generera en UI-lista över alla trails i store/trails.
- Köphantering och bekräftelsepanel (“Are you sure?”).
- UI-animationer (in-/utzoomning när butiken öppnas/stängs)

```
void LoadData()
{
    LoadOwnedTrails()
        .ContinueWithOnMainThread(task => LoadTrails())
        .ContinueWithOnMainThread(task => LoadCurrency())
        .ContinueWithOnMainThread(task => ShowStoreUI());
}
```

Kodexempel 4.6.1: *Metodkedja för att ladda spelarens data (ägda trails, valuta) och sedan visa Store-scenen. för fullständig kod, se Bilaga A*

Funktioner i korthet

LoadOwnedTrails()

- Hämtar trail-namn som spelaren redan äger och inaktiverar köpknappar för dessa trails.

LoadTrails()

- Hämtar allmänna trail-poster från store/trails i Firebase, skapar en UI-prefab för varje trail med namn, pris, sprite.

LoadCurrency()

- Läser users/{userId}/currency och uppdaterar en valuta-text i UI. Vid köp dras priset från spelarens saldo.

Köpbekräftelse

- RequestPurchase(int price, string trailName, GameObject trailItem) visar en panel med "Are you sure?" innan köpet slutförs i *OnConfirmPurchase()*.
- Om köpet godkänns, uppdateras databasen (valuta - pris, trailen läggs till ownedTrails).

```
public void RequestPurchase(int price, string trailName, GameObject trailItem)
{
    confirmationMessage.text = $"Buy \"{trailName}\" for {price} coins?";
    ShowConfirmationPanel();
}

void OnConfirmPurchase()
{
    PurchaseTrail(pendingTrailPrice, pendingTrailName, pendingTrailItem);
}
```

Kodexempel 4.6.2: *(förkortat utdrag ur `StoreManager.cs`) Köpprocessen, inklusive en enkel bekräftelse innan valutan dras. för fullständig kod, se Bilaga A*

UI-animationer

Med DOTween zoomas StoreCanvas in från skala 0 till 1 vid öppning och zoomas ut igen vid stängning, vilket får scenebyte att se bättre ut.

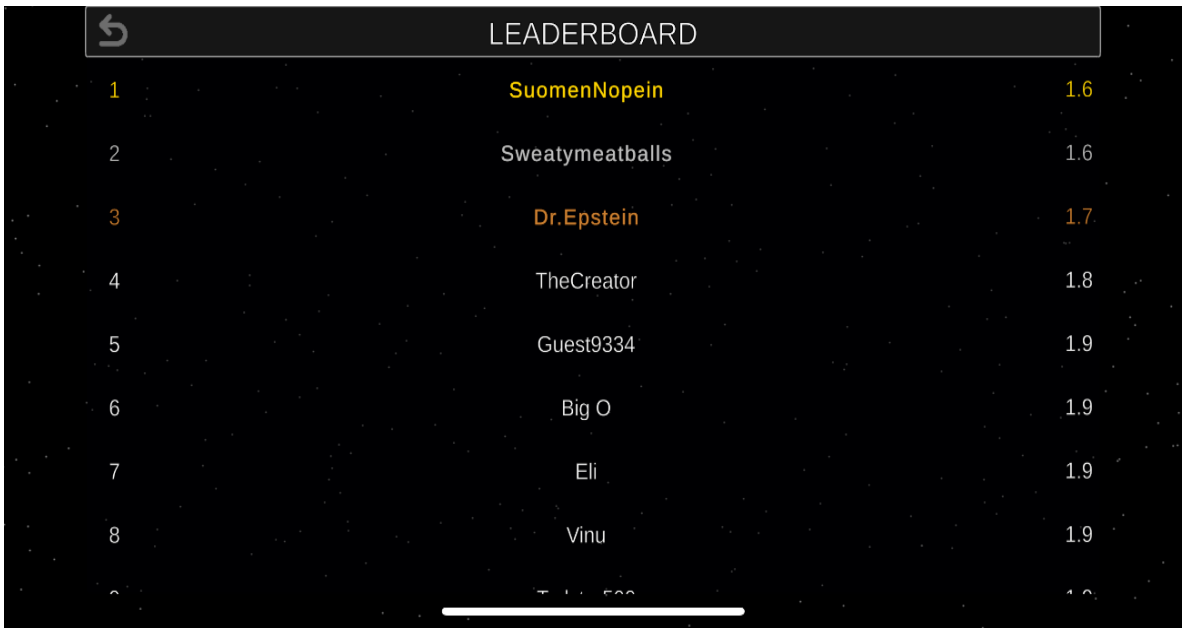
Reflektion

Store-scenen låter spelare köpa olika trails och motiverar spelare att tjäna mer valuta. Samtidigt säkrar Firebase att köpdata sparas, så att spelaren inte förlorar sina inköp om spelaren byter apparat. Att ha en tydlig "Are you sure?"-panel vid köp minskar risken för köp som sker i misstag. (För hela StoreManager.cs, se Bilaga A.)

4.7 Leaderboard Scene

Syfte

I Leaderboard-scenen visas en topplista över de bästa tiderna (bestResult) i *ReactFast*. Spelarnas data hämtas från Firebase, sorteras i stigande ordning och presenteras i en UI-lista, där de 50 bästa resultaten syns. Placeringarna 1–3 har specialfärger (guld, silver, brons) för att höja tävlingskänslan.



Figur 4.7.1. *Leaderboard Scene – spelets topplista*

Huvudsakligt skript: LeaderboardManager.cs

LeaderboardManager ansvarar för:

- Hämtning av användardata från /users i Firebase.
- Sortering baserat på bestResult.
- Visning av topp 50-spelare i en förbered UI-prefab.
- Zoom-animation (DOTween) när scenen öppnas och stängs.

```
void FetchLeaderboardData()
{
    // Hämtar data sorterat på 'bestResult'
    dbReference.OrderByChild("bestResult").GetValueAsync()
        .ContinueWithOnMainThread(task =>
        {
            if (task.IsCompleted)
            {
                List<UserData> leaderboardData = new List<UserData>();

                // Samla username + bestResult för alla användare
                foreach (DataSnapshot userSnapshot in task.Result.Children)
                {
                    if (userSnapshot.HasChild("username") && userSnapshot.HasChild("bestResult"))
                    {
                        string username = userSnap-shot.Child("username").Value.ToString();
                        float bestResult = float.Parse(userSnapshot.Child("bestResult").Value.ToString());
                        leaderboardData.Add(new UserData(username, bestResult));
                    }
                }
                // Visar top 50
              DisplayLeaderboard(leaderboardData);
            }
            else
            {
                Debug.LogError("Failed to retrieve data from Firebase.");
            }
        });
```

Kodexempel 4.7.1 *Hämta och sortera bestResult från Firebase (se Bilaga A för fullständiga koden LeaderboardManager.cs).*

Funktioner i korthet

Hämtning av data

- *FetchLeaderboardData()* använder OrderByChild("bestResult") för att få en sorterad lista med lägsta tid först.

Sortering och visning

-*DisplayLeaderboard()* ordnar spelarna i stigande ordning och instansierar en prefab för varje post, men begränsar sig till 50 poster för överskådlighet.

Design av topplaceringar

-Rank 1–3 får speciella färger (guld, silver, brons) och fetstil för att uppmuntra tävlingsinstinkt.

Animationer

-Scenen zoomar in när den visas och zoomar ut när spelaren lämnar den, liknande effekterna i Store-scenen.

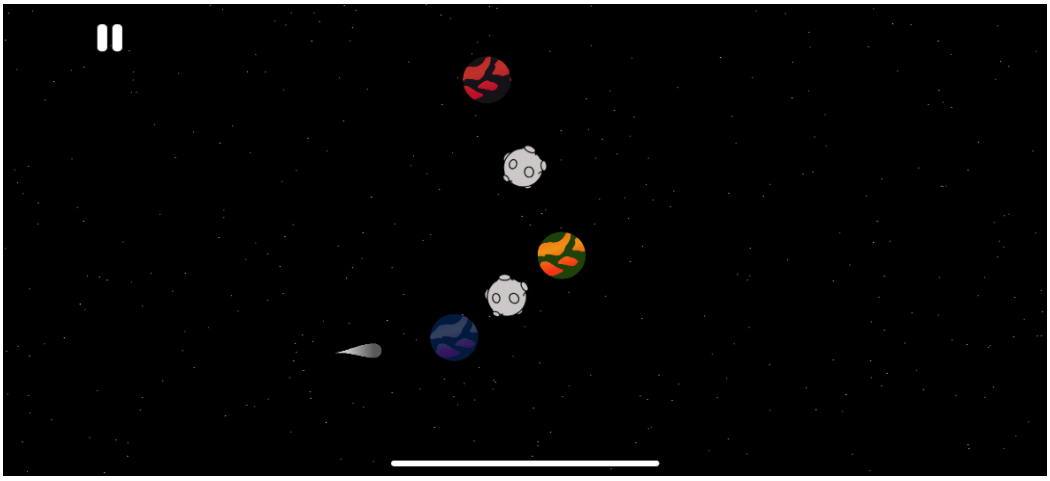
Reflektion

En topplista (leaderboard) är ofta en stark drivkraft i spel (Amo m.fl.2018), då spelare motiveras att förbättra sin tid och klättra på listan. Genom att markera topp 3 extra tydligt förstärks tävlingsmomentet ytterligare. Begränsningen till 50 poster hjälper också att hålla gränssnittet lättöverskådligt på en mobilskärm (UX-aspekt).

4.8 Game Scene

Syfte

Game-scenen är spelets huvuddel. Här drar spelaren fingret genom olika mönster för att få så kort tid som möjligt. Här hanteras bland annat spelloopen (start, nivåer, slut), svårighetsökning och inmatning via touch.



Figur 4.8.1. *Game Scene – spelvy med dragmönster och trail-effekt.*

4.8.1 GameController.cs

Ansvarsområden

- Start och slut av spelet (metoder som *StartLevel(), EndGame()*).

- Tidtagning (beräkning av snittid) och uppdatering av spelarens bästa resultat i Firebase.
- Svårighetsökning (Easy → Moderate → Hard → VeryHard).
- Valuta (exempelvis +50 mynt när spelet är klart).

```
async void EndGame()
{
    // Exempel: beräkna medeltid och jämför med personbästa
    float totalTime = 0;
    foreach (float t in levelTimes) totalTime += t;
    float averageTime = totalTime / levelTimes.Count;

    string userId = Fire-baseAuth.DefaultInstance.CurrentUser.UserId;
    var snapshot = await FirebaseDatabase.DefaultInstance
        .GetReference("users").Child(userId).Child("bestResult")
        .GetValueAsync();

    // Uppdatera personbästa om medeltiden är bättre
    if (snapshot.Exists && float.TryParse(snapshot.Value.ToString(), out float storedBest))
    {
        if (averageTime < storedBest)
          awaitFirebaseData-base.DefaultInstance.GetReference("users")
                .Child(userId).Child("bestResult").SetValueAsync(averageTime);
    }

    // Visa slutpanel, etc.
    endGamePanel.SetActive(true);
}
tePatternRandomly(chosenPattern);
}
```

Kodexempel 4.8.1: *Nedkortad metod EndGame(). Se Bilaga A för hela GameController.cs.*

4.8.2 TouchController.cs

Ansvarsområden

- Hantering av inmatning (mus eller touch).
- Streck (trail) efter fingret, om spelaren drar på skärmen.
- Identifiering av vilka "dots" som berörs, i vilken ordning.
- Ljudeffekter (slice-ljud, explosion) för bättre spelkänsla.

```
void CheckForDotTouch(Vector2 position)
{
    RaycastHit2D hit = Physics2D.Raycast(position, Vector2.zero);
    if (!hit.collider) return;

    // Om collider har 'BlackDot' => kolla ordning
    if (hit.collider.CompareTag("BlackDot"))
    {
        TMP_Text numberText = hit.collider.GetComponentInChildren<TMP_Text>();
        if (numberText != null && int.TryParse(numberText.text, out int dotIndex))
        {
            if (dotIndex == currentDotIndex)
            {
                // Rätt punkt => stäng av dot, ev. explosion, etc.
                hit.collider.gameObject.SetActive(false);
                blackDotPool.ReturnObject(hit.collider.gameObject);
                currentDotIndex++;

                // Om inga fler "BlackDot" => meddela GameCon-troller
                if (GameObject.FindGameObjectsWithTag("BlackDot").Length == 0)
                {
                    gameController.CompleteLevel();
                    ResetDotIndex();
                }
            }
        }
    }
}
```

Kodexempel 4.8.2: *Ser till att dotarna träffas i rätt ordning. Se Bilaga A för hela TouchController.cs.*

4.8.3 PatternManager.cs

Ansvarsområden

- **Generera mönster** (patterns) för fyra svårighetsgrader (Easy, Moderate, Hard, VeryHard).
- **Spawn av punkter** (stora och små “dots”).
- **Eventuell slumpmässig rotation** och om mönstret ska vara ”slutet” (closePattern).

```
public PatternData GetRandomPatternData(Difficulty difficulty)
{
    List<PatternData> patternList;

    switch (difficulty)
    {
        case Difficulty.Moderate:
            patternList = moderatePatterns;
            break;
        case Difficulty.Hard:
            patternList = hardPatterns;
            break;
        case Difficulty.VeryHard:
            patternList = veryHardPatterns;
            break;
        default:
            patternList = easyPatterns;
            break;
    }

    int randIndex = Random.Range(0, patternList.Count);
    PatternData chosenPattern = patternList[randIndex];

    return RotatePatternRandomly(chosenPattern);
}
```

Kodexempel 4.8.3: *Väljer slumpmässigt ett mönster. Se Bilaga A för hela Pattern-Manager.cs.*

Så fungerar Game-scenen i praktiken

- **Spelstart**: GameController initierar objektpooler (dots, explosioner) och sätter svårighetsnivån till Easy.
- **Mönstergenerering**: Vid varje ny nivå anropar GameController → Pattern-Manager för ett slumpmässigt mönster.
- **Drag och träff**: TouchController följer spelarens finger och registrerar om rätt "BlackDot" träffas i ordning.
- **Nivåavslut**: När alla punkter är träffade, höjs nivån (och eventuellt. svårighet).
- **Spelslut**: Efter 10 nivåer räknar EndGame() ut snittiden och uppdaterar Fire-base.

Reflektion

Den uppdelning (GameController, TouchController, PatternManager) gör spelet lätt att felsöka och vidareutveckla. Man kan lägga till nya mönster i PatternManager utan att röra inputkoden i TouchController. Genom att logga personbästa i Firebase kan spelare jämföra sina resultat via topplistan och ständigt försöka förbättra sin tid

Sammanfattning

- **GameController**: Huvudflöde (start, slut, snittid, svårighetsökning).
- **TouchController**: Hanterar draget efter fingret, ordningsföljd för dots och explosioner.
- **PatternManager**: Genererar mönstren i olika svårighetsgrader.

Tillsammans skapar dessa skript en intuitiv och repetitionsbar spelupplevelse, där spelaren tävlar mot klockan för att få så låg snittid som möjligt. Detta kombineras med spelmekaniska element (anpassning av trails, progressiv svårighet, topplista) för ökad motivation och engagemang.

4.9 Ljud och Musik

I det här avsnittet beskrivs hur ljud och musik implementerats i *ReactFast* för att förstärka spelkänslan. Genom olika ljudeffekter – som bakgrundsmusik, knappklick och särskilda ljudeffekter i spelscenen – skapas en mer engagerande och responsiv upplevelse.

4.9.1 AudioManager

AudioManager är ett centralt script som sköter de flesta ljudfunktionerna i spelet och ser till att endast ett (Singleton) ljudobjekt finns. Det gör att samma ljudresurser kan användas genom hela spelet, även när man byter scener.

```
public class AudioManager : MonoBehaviour
{
    private static AudioManager _instance;
    public static AudioManager Instance { /* Singleton-getter */ }

    public AudioSource audioSource;    // Spelar upp korta ljudeffekter
    private AudioSource laserAudioSource; // För loopande ljud

    void Awake()
    {

        if (_instance != null && _instance != this) { De-stroy(gameObject); return; }
        _instance = this;
        DontDestroyOnLoad(gameObject);
    }

    public void PlaySound(AudioClip clip)
    {
        audioSource.PlayOneShot(clip);
    }

    public void StartLoopingSound(AudioClip clip) { /* ...starta en loop... */ }
    public void StopLoopingSound() { /* ...stoppa loopen... */ }
}
```

Kodexempel 4.9.1 *(förkortat utdrag ur `AudioManager.cs`) – `PlaySound()`, `StartLoopingSound()`, `StopLoopingSound()` för fullständig kod se bilaga A.*

Singleton: Gör att AudioManager.Instance kan anropas överallt utan extra referenser.

Looping: Genom två olika AudioSource kan ett ljud loopas medan andra engångsljud spelas i bakgrunden.

DontDestroyOnLoad: Objektet följer med mellan scener, så att musik eller bakgrundsljud inte avbryts vid scenbyte.

4.9.2 ButtonSoundManager

Många scener har knappar som användaren trycker på. ButtonSoundManager letar upp alla knappar i scenen och kopplar samma klick-ljud, så att utvecklaren slipper ange ett ljudklipp för varje enskild knapp.

```
public class ButtonSoundManager : MonoBehaviour
{
    public AudioClip customClickSound;

    void Start()
    {
        Button[] buttons = FindObjectsOfType<Button>(true);
        foreach (Button button in buttons)
        {
            button.onClick.AddListener(() => {
                AudioManag-er.Instance.PlaySound(customClickSound);
            });
        }
    }
}
```

Kodexempel 4.9.2 *(förkortat utdrag ur `ButtonSoundManager.cs`) – kopplar `customClickSound` till alla knappar i `Start()` för fullständiga kod se bilaga A.*

Automatisk koppling: Söker upp alla Button i scenen och lägger till en onClick-händelse som spelar upp customClickSound.

Anpassningsbart: Om du vill använda olika klick-ljud i olika scener kan du ange olika AudioClip i Inspector.

4.9.3 GameSoundManager

I GameScene används ett skript kallat GameSoundManager för att hantera de ljud som faktiskt används i själva spelet, till exempel ett ”slice”-ljud när spelaren gör en snabb

rörelse, en ”explosion”-ljud när en punkt förstörs, samt en ”gameComplete”-ljud när alla nivåer är avklarade.

```
public class GameSoundManager : MonoBehaviour
{
    public static GameSoundManager Instance;

    public AudioClip sliceSound;
    public AudioClip explosionSound;      // Spelas när en “dot” rörs
    public AudioClip gameCompleteSound;   // Spelas när spelet är klart
    public AudioClip nextLevelDotSpawnSound;

    void Awake()
    {
        if (Instance == null) {
            Instance = this;
        } else {
            Destroy(gameObject);
        }
    }

    public void PlaySliceSound(Vector3 position)
    {
        if (sliceSound != null) {
            AudioSource.PlayClipAtPoint(sliceSound, position);
        }
    }

    public void PlayExplosionSound(Vector3 position)
    {
        if (explosionSound != null) {
            AudioSource.PlayClipAtPoint(explosionSound, position);
        }
    }

    public void PlayGameCompleteSound(Vector3 position)
    {
        if (gameCompleteSound != null) {
            AudioSource.PlayClipAtPoint(gameCompleteSound, position);
        }
    }

    // Exempel: ljud när nästa-nivå-dot spawns
    public void PlayNextLevelDotSpawnSound(Vector3 position)
    {
        if (nextLevelDotSpawnSound != null) {
            AudioSource.PlayClipAtPoint(nextLevelDotSpawnSound, posi-tion);
        }
    }
}
```

Kodexempel 4.9.3*(förkortat utdrag ur `GameSoundManager.cs`) –`PlaySliceSound()`, `PlayExplosionSound()`, `PlayGameCompleteSound()`.för fullständiga kod se bilaga A.*

Funktioner i korthet

Slice-ljud: Spelas när spelaren lyfter fingret eller ändrar riktning snabbt.

Explosion-ljud: Körs när en ”dot” förstörs (ex. i TouchController eller GameController).

Game complete: Anropas när spelaren klarat spelets alla 10 nivåer.

Övriga ljudeffekter: Exempelvis “nextLevelDotSpawnSound” för att förstärka känslan när en ny nivå startar.

Reflektion

Även om skriptet stödjer fler ljud (*exempelvis dragSound*), används här enbart de effekter som faktiskt behövs för att betona spelets centrala händelser (när en punkt träffas eller när spelet tar slut). Genom att koppla ljuden direkt till spelets händelser i TouchController eller GameController får spelaren omedelbar feedback, vilket både ökar engagemanget och gör spelets rytm tydligare (Korhonen, Holm och Heikkinen 2007).

5 Resultat

I detta kapitel presenteras resultaten från enkäten som skickades ut till 20 personer som testade betaversionen av spelet *ReactFast*. Urvalet bestod främst av personer i åldern 20–30 år, med varierande erfarenhet av mobilspel 14 av ~20 testpersoner besvarade enkäten. Den var anonym, vilket innebär att svaren inte kunde kopplas direkt till enskilda spelares resultat i databasen. Nedan redovisas först de kvantitativa resultaten från fem skalfrågor, därefter de kvalitativa svaren från öppna frågor om vad som var bäst och vad som saknades eller behövde förbättras.

5.1 Kvantitativa resultat

Deltagarna fick bedöma fem påståenden på en skala från 1–5 (1 = negativ/låg, 5 = mycket positiv/hög). Tabellen nedan visar medelvärde (M) och standardavvikelse (SD) för varje fråga.

Fråga	M	SD
Hur roligt upplevde du spelet	4,27	0,79
Hur lätt var det att förstå dragmekaniken?	4,36	0,67
Upplevdes spelet som för enkelt eller för svårt	3,00	0,45
Hur motiverad var du att spela flera rundor	4,27	0,79
Skulle du kunna tänka dig att fortsätta spela eller rekommendera spelet	4,18	0,75

5.1.1 Analys av skalfrågorna

Hur roligt upplevde du spelet? (M=4,27, SD=0,79)

De flesta upplevde spelet som roligt eller mycket roligt, vilket tyder på en överlag positiv spelupplevelse.

Samtidigt bör man tänka på att de som testade spelet främst var vänner och bekanta, vilket kan ha påverkat resultatet. Det är möjligt att en grupp av helt okända användare hade bedömt spelet annorlunda. Därför vore det värdefullt att i framtida tester inkludera deltagare utan personlig koppling till utvecklaren för att få en mer objektiv bild av spelets upplevelse.

Hur lätt var det att förstå dragmekaniken? (M=4,36, SD=0,67)

Spelet ansågs överlag tydligt och intuitivt vad gäller själva dragmomentet.

Var det för enkelt eller för svårt? (M=3,00, SD=0,45)

Spelet upplevdes i genomsnitt som lagom svårt, utan större frustration eller underutmaning.

Hur motiverad var du att spela flera rundor? (M=4,27, SD=0,79)

Deltagarna kände sig generellt motiverade att fortsätta spela för att förbättra sin reaktionstid.

Skulle du kunna tänka dig att fortsätta spela eller rekommendera spelet? (M=4,18, SD=0,75)

De flesta kan tänka sig att spela vidare och rekommendera spelet till andra, vilket signalerar en god spridningspotential.

5.2 Kvalitativa resultat

Utöver skalfrågorna fick deltagarna möjlighet att kommentera vad som var bäst med spelet samt vad de ansåg kunde förbättras eller saknades.

5.2.1 Vad tyckte du var bäst?

Dragmekaniken och responsen

Flera deltagare nämnde att dragmekaniken kändes ”smooth” och att spelet reagerade snabbt på spelarens rörelser, vilket upplevdes positivt.

Tävling och topplista

Flera betonade hur tävlingsmomentet ökade motivationen, t.ex. ”Att man kunde tävla mot andra och se deras bästa resultat.” Vissa beskrev en drivkraft att uppnå bättre “high score” i varje runda.

Design, ljud och tempo

Ett par spelare lyfte fram att designen var snygg eller tydlig, och att tempot gav en känsla av energi. Även spelets ljud (inklusive musiken) fick positiv feedback.

5.2.2 Vad tyckte du var mindre bra, eller vad saknade du?

Instruktioner och tutorial

Flera efterfrågade en tydligare introduktion för nya spelare. Vissa skrev att de förstod mekaniken snabbt, men andra ansåg att en inbyggd tutorial hade underlättat.

Storlek och precision på objekt

Vissa tyckte att objekt (dots) var något för små, särskilt för större fingrar, vilket kunde orsaka missar även när man trodde sig ha träffat exakt.

Variation och ytterligare innehåll

Några önskade fler mönster, fler svårighetsgrader eller nya “modes”. En respondent beskrev spelet som ”aningens simpelt” och såg gärna större omväxling i uppdragen.

Ojämnhet eller tur

Ett fåtal menade att svårighetsgraden kunde variera mycket mellan rundor, vilket ibland upplevdes som om “tur” spelade in i hur bra resultatet blev.

5.3 Sammanfattning av resultaten

Sammanfattningsvis visar enkätsvaren att *ReactFast* har mottagits positivt: spelet anses roligt, tydligt och motiverande att spela om flera gånger. Förbättringsförslag handlar framför allt om större objekt, en kort tutorial och mer variation i mönster eller spellägen. Tävlingsmomentet via topplistan upplevdes av många som en stark drivkraft att spela flera rundor och jämföra resultat.

I nästa kapitel (Diskussion) kopplas dessa resultat till teoretiska ramverk (Flow, MDA, UX) och spelmekanikers betydelse för engagemang.

6 Diskussion

I detta kapitel diskuteras de resultat som framkommit i föregående avsnitt (Resultat), och sätts i relation till spelets designmål samt de teoretiska ramverk som presenterades tidigare: Flow-teorin (Csikszentmihalyi, 1990), MDA-modellen (Hunicke 2004) och grundläggande principer för användarupplevelse (UX).

6.1 Reflektion kring spelets design och mottagande

6.1.1 Positiv respons på dragmekaniken och tempot

Ett av huvudmålen med *ReactFast* var att skapa en snabb, intuitiv dragbaserad spelupplevelse som skulle kännas utmanande men ändå lättillgänglig. Av enkätresultaten framgick att:

- Deltagarna i genomsnitt gav höga betyg (4,36) för “Hur lätt var det att förstå dragmekaniken?”, och
- Flest positiva kommentarer rörde hur “smooth” spelet kändes när man drog fingret genom mönstren.

Utifrån MDA-ramverket (Mechanics, Dynamics, Aesthetics) kan dragmekaniken betraktas som en kärn-”Mechanic”. Detta har i sin tur skapat Dynamics av snabb respons, vilket många beskrev som engagerande och motiverande. Estetiken (Aesthetics) uppfattas alltså som en snabb, flytande spelkänsla, precis som man hoppats.

Samtidigt verkar detta bidra till att spelarna lättare kunde nå ett flow-tillstånd (Csikszentmihalyi, 1990) i och med att kontrollen (mechanic) kändes både enkel och responsiv. Flow-faktorer som omedelbar feedback och tydliga mål (förbättra sin tid, hålla sig i dragbanan) kan förklara varför många också upplevde spelet som roligt (M=4,27) och motiverande (M=4,27).

6.1.2 Tävlingsmomentet via topplistan

Ett annat designmål var att öka engagemanget och konkurrensandan hos spelarna genom en topplista. Enligt resultaten var detta mycket uppskattat; flera personer nämnde att det motiverade dem att jaga bättre tider:

“Snabba tempot var bra, att tävla mot andra på leaderboard gav motivation att grinda för bättre resultat.”

I termer av MDA kan topplistan ses som ett “meta-layer” på dynamics-sidan, som skapar en känsla av tävling och driver spelaren att återupprepa samma mechanic för att förbättra sitt resultat. Samtidigt syns det i enkätsvaren (4,18 i “skulle du fortsätta spela/rekommendera?”) att många vill spela vidare – sannolikt för att klättra i rankingen. Detta överensstämmer även med forskning om reflexbaserade spel (Amo 2018 m.fl.) där tävlingsmomentet ofta är en stark drivkraft.

6.2 Utmaningar och förbättringsförslag

6.2.1 Instruktioner och tutorial

Flera testare efterfrågade tydligare instruktioner eller en kort tutorial i början. Även om spelet i sig ansågs intuitivt, kunde en enkel steg-för-steg-guide vid start ha underlättat för nya spelare. Detta rör UX-principer om att sänka tröskeln för nybörjare (Cao och Liu 2022; Diwald 2022). Att lägga till en valbar tutorial eller tips i början skulle ytterligare kunna höja användarvänligheten, särskilt för dem som inte är vana mobilspelare.

6.2.2 Storlek och variation i objekt

En annan återkommande synpunkt var att objekten (dots) ibland kändes för små, vilket ledde till missar trots att spelaren tyckte sig träffa rätt. Detta skulle kunna lösas antingen genom att göra objekten större eller genom att ge dem en större träffzon (hitbox), så att spelet registrerar drag även vid mindre avvikelser.”.

Dessutom önskade vissa deltagare mer variation i mönstren eller fler svårighetsgrader. Enligt Flow-teorin kan en brantare svårighetskurva eller fler mönster motverka att spelet blir för enformigt när spelarens färdighet ökar. Detta är viktigt eftersom flow uppnås när utmaningen ligger i linje med eller strax över spelarens förmåga (Csikszentmihalyi, 1990).

6.2.3 Ojämnhet och “tur”

Ett fåtal menade att spelet var ojämnt från runda till runda, vilket kunde ge en känsla av “tur” snarare än skicklighet. Detta kan bero på hur mönstren slumpas. Genom att exempelvis justera slumpgeneratorn eller införa en svårighetsparameter vid generering av banor (för att hålla banor likvärdiga) kan man öka upplevelsen av rättvisa. I MDA-termer är detta en finjustering av spelets Dynamics, så att slumpmomentet inte upplevs för starkt.

6.3 Koppling till forskningsfrågorna

6.3.1 Viktiga design- och utvecklingsaspekter

De resultat som framkommit pekar på att en snabb och responsiv dragbaserad mekanik, kombinerad med ett tävlingselement via topplistan, är en effektiv grund för ett reflexbaserat mobilspel. Detta sammanfaller väl med tidigare studier som betonat vikten av tydlig feedback och enkelt greppbara gester (Diwald 2022).

Samtidigt visar feedbacken att UI-aspekter som storleken på objekt och behovet av instruktioner är avgörande att tänka på redan från början i utvecklingsprocessen. Från ett tekniskt perspektiv kan en flexibel hitbox och en bra slumpningsalgoritm göra stor skillnad för spelkänslan.

6.3.2 Upplevelse (användbarhet och engagemang)

En majoritet av testarna upplevde spelet som roligt (M=4,27) och tydligt (M=4,36), vilket indikerar att dragmekaniken förenklar användarupplevelsen, samt att tävlingsmomentet höjer engagemanget. Att många frivilligt spelade flera rundor och att vissa uttryckte att de gärna fortsätter spela (M=4,18) indikerar att spelet når en relativt hög nivå av användbarhet och engagemang.

Med vissa justeringar, som tydligare tutorial och större objekt, kan spelet *ReactFast* förhoppningsvis ännu bättre upprätthålla ett flow-tillstånd över tid. Genom att gradvis öka utmaningen eller introducera fler mönster kan spelet hålla i gång spelarnas engagemang längre (Csikszentmihalyi, 1990).

6.4 Metodologiska begränsningar

Endast fjorton svar samlades in, vilket gör att resultaten inte går att generalisera till en bredare population. Testgruppen bestod dessutom främst av bekanta i åldern 20–30 år, vilket kan innebära viss skevhet i urvalet. Trots detta ger svaren en första indikation om att *ReactFast* har en lovande grunddesign och engagerande spelmekanik.

6.4.1 Slutsatser och vidare arbete

Sammanfattningsvis stödjer resultaten att *ReactFast* har en dragbaserad mekanik och ett tävlingsmoment som effektivt motiverar spelare att upprepa rundor för att förbättra sina tider. Samtidigt belyser testarna behovet av mer variation, bättre instruktioner samt potentiellt större eller mer förlåtande objekt.
I framtiden kan utvecklingen fokusera på:

- Implementering av en kort tutorial, främst för nya spelare.
- Fler mönster och eventuellt nya spellägen för att öka variation.
- Justering av objektstorlek eller hitbox för att minska ofrivilliga missar.
- Finjustering av slumpen i banor för att minska upplevelsen av tur eller ojämnhet.

Härnäst följer en sammanfattning av arbetets slutsatser i kapitel 7, där även kopplingen till forskningsfrågorna belyses i mer definitiv form.

7 Avslutande reflektioner och slutsatser

I detta examensarbete har jag undersökt hur ett snabbt, dragbaserat mobilspel – *ReactFast* – kan utformas och upplevas av spelare, med särskild tonvikt på tekniska lösningar, designaspekter och användarupplevelse. Arbetet har utgått från två centrala frågeställningar:

1. Vilka är de viktigaste design- och utvecklingsaspekterna vid skapandet av ett snabbt, dragbaserat mobilspel?
2. Hur upplever spelare *ReactFast* i termer av användbarhet och engagemang?

Genom att kombinera teoretiska ramverk (Flow-teorin, MDA-modellen, UX-principer) med praktisk spelutveckling (Unity, Firebase) och empiriska användartester har jag fått insikter i hur dragbaserad interaktion kan skapa en responsiv spelupplevelse som motiverar spelare att återkomma. Nedan sammanfattas de viktigaste slutsatserna samt förslag på vidare utveckling.

7.1 Centrala lärdomar och slutsatser

- **Dragbaserad mekanik och snabb feedback**

  En av de mest uppskattade aspekterna i *ReactFast* var hur ”smooth” och direkt spelupplevelsen kändes när spelaren drog fingret genom mönstren. Kombinationen av tydlig visuell feedback och snabb respons understryker vikten av att ha en väl kalibrerad ”hitbox” och lagom stora objekt. Detta stöds av tidigare forskning om touchbaserade gester, där omedelbar återkoppling anses avgörande för att spelare ska känna flyt (flow).

- **Tävlingsmomentet ökar engagemang**

  Topplistan (leaderboard) upplevdes av många som en stark drivkraft. Att kunna jämföra sina resultat med andra spelare bidrog till att fler spelade om flera gånger för att förbättra sin tid. Detta ligger i linje med MDA-ramverkets syn på hur en övergripande **Dynamic** (tävling) kan förstärka **Mechanics** (snabb dragning) och därmed påverka **Aesthetics** (engagerande, motiverande).

- **Vikten av tydlig onboarding**

  Trots att många fann själva spelet intuitivt, uppstod önskemål om en enkel tutorial eller korta instruktioner i början. Detta visar att en genomtänkt onboardingprocess är viktig, särskilt för en bredare målgrupp med varierande erfarenhet av reflexbaserade spel.

- **Balans mellan enkelhet och variation**

  Medan enkelheten i dragmekaniken uppskattades, nämnde några spelare att de saknade fler spellägen eller mönster. En mer dynamisk svårighetskurva och ytterligare designade banvarianter kan bidra till att behålla spelglädjen över längre tid. Här blir flow-teorins princip om att matcha utmaning med spelarens växande förmåga särskilt relevant.

7.2 Begränsningar i arbetet

Då antalet testdeltagare var begränsat och främst bestod av bekanta i åldern 20–30 år, är resultatens generaliserbarhet begränsad. En bredare målgrupp – både ur ålders- och erfa-

renhetsperspektiv – hade kunnat ge en mer nyanserad bild av spelets användbarhet och engagemang. Dessutom fokuserade arbetet på enbart en prototyp av *ReactFast* och en enklare analys av speldata. Mer omfattande studier, där spelare får testa spelet under längre perioder eller ingår i kontrollerade experimentmiljöer, kan ge ytterligare insikter om hur dragbaserade reflexspel bäst kan designas och marknadsföras.

7.3 Förslag på vidare utveckling

- **Implementera en kort tutorial**

  För att sänka tröskeln för nybörjare och minska risken för missförstånd kring hur man drar fingret genom mönstren, kan ett inledande guiderum (eller valbar hjälpfunktion) introduceras. Detta skulle förbättra spelets initiala användbarhet och skapa en ännu positivare upplevelse.
- **Flera mönster och spellägen**

  Genom att utöka antalet mönster eller införa nya lägen (t.ex. tidsbegränsad ”Arcade mode” eller ”Endless mode” med ökande svårighetsgrad) kan man öka variationen. Ett gradvis stigande motstånd gör dessutom att skickliga spelare hela tiden känner sig utmanade.
- **Justera objektstorlek och träffzoner**

  Flertalet testdeltagare upplevde att objekten var i minsta laget. Med större träffzoner (eller en mer förlåtande ”hitbox”) kan man minimera frustration orsakad av att spelet inte registrerar rörelser som spelaren själv upplever som korrekt utförda.
- **Finjustera slumpmässighet**

  För att undvika känslan av ”tur” när olika banor genereras, kan man införa en mer kontrollerad slumpgenerator som säkerställer jämn svårighet.

  Utökade spelardata och analys

  Vidare arbete kan fokusera på att logga fler aspekter av spelbeteendet (exempelvis exakt var spelaren missar, hur många försök de gör per dag etc.) och sedan koppla dessa data till enkätsvar. På så sätt kan man mer ingående undersöka hur spelmekanik och svårighet samspelar med engagemang och lärkurva.

7.4 Slutord

Arbetet med *ReactFast* visar att en enkel men genomtänkt dragbaserad mekanik, kombinerad med snabb feedback och ett tävlingsmoment, kan skapa en motiverande och rolig spelupplevelse på mobil. De teoretiska ramarna – Flow, MDA och grundläggande UX-principer – har fungerat väl för att förklara varför spelare upplever spelet som engagerande och vad som kan förbättras.

Samtidigt påminner testresultaten oss om att spelutveckling är en iterativ process. Även när grunden är solid återstår ofta justeringar av balans, variation och användarstöd för att spelet fullt ut ska motsvara spelarnas förväntningar och fortsätta kännas nytt och spännande. Med fler testpersoner, ytterligare funktioner och fortsatta iterationer har *ReactFast* potential att finna en stabil plats bland reflex- och tävlingsinriktade mobilspel.

Genom detta examensarbete har jag inte bara fördjupat mig i dragbaserade spelmekaniker och teknisk implementation, utan också fått en starkare förståelse för hur spelupplevelse och engagemang kan formas av detaljer i interaktionsdesignen. Förhoppningen är att dessa lärdomar ska inspirera vidare utveckling av *ReactFast* och andra mobilspel som kombinerar snabbhet, intuitivitet och tävlingsanda på ett effektivt sätt.

8 Källförteckning

Amo, Laura C, **Ruochen Liao, H.Raghav Rao, Gretchen Walker (2018).** *Effects of Leaderboards in Games on Consumer Engagement.* Proceedings of the 2018 ACM SIGMIS Conference on Computers and People Research.
Tillgänglig på:**Effects of Leaderboards in Games on Consumer Engagement | Proceedings of the 2018 ACM SIGMIS Conference on Computers and People Research** Hämtad: (2.22.2025)

Butler, C. (2013). *The Effect of Leaderboard Ranking on Players' Perception of Gaming Fun*. In: Ozok, A.A., Zaphiris, P. (eds) Online Communities and Social Computing. OCSC 2013. Lecture Notes in Computer Science, vol 8029. Springer, Berlin, Heidelberg. Tillgänglig på: The Effect of Leaderboard Ranking on Players' Perception of Gaming Fun | SpringerLink Hämtad (3.1.2025)

Cao,s & Liu,f (2022). *Learning to play: understanding in-game tutorials with a pilot study on implicit tutorials* Tillgänglig på: Learning to play: understanding in-game tutorials with a pilot study on implicit tutorials: Heliyon (cell.com) Hämtad: (3.10.2025)

Cowley, B.U., Keith, D., Michaela, C., Black, R.J. och Hickey, T. (2008). *Toward an understanding of flow in video games*. Tillgänglig på: (PDF) Toward an understanding of flow in video games Hämtad (3.4.2025)

Csikszentmihalyi, M. (1990). *Flow: The psychology of optimal experience*

Diwald, A. (2022). *Development and Analysis of Mobile Game Mechanics*
Tillgänglig på:reposiTUm:Development and analysis of mobile game mechanics (tuwien.at) (Hämtad 3.20.2025)

Hogue, J. (2023) *6 Tips to Reduce Sign-Up Friction for Your Platform*. [Online]. Tillgänglig på: 6 Ways to Reduce Sign-up Form Friction on Your Platform | Insights | Oomph, Inc Hämtad: (3.15.2025)

Hunicke, R., LeBlanc, M. & Zubek, R. (2004) *MDA: A Formal Approach to Game Design and Game Research*. Tillgänglig på: (PDF) MDA: A Formal Approach to Game Design and Game Research Hämtad (2.20.2025)

Korhonen, H., Holm, J., Heikkinen, M. (2007). *Utilizing Sound Effects in Mobile User Interface Design*. In: Baranauskas, C., Palanque, P., Abascal, J., Barbosa, S.D.J. (eds) Human-Computer Interaction – INTERACT 2007. Tillgänglig på: Utilizing Sound Effects in Mobile User Interface Design | SpringerLink Hämtad: (3.26.2025)

Newzoo (2021) *Global Mobile Market Report*. Tillgänglig på: Free Newzoo Report: Global Mobile Market Report 2021 Hämtad (2.10.2025)

Newzoo (2024) *Global Games Market Report*.Tillgänglig på: **Newzoo's free Global Games Market Report 2024 | New report live!** (hämtad 2.10.2025)

Sanders,K (2017) *To Use Or Not To Use: Touch Gesture Controls For Mobile Interfaces*. Tillgänglig på: To Use Or Not To Use: Touch Gesture Controls For Mobile Interfaces — Smashing Magazine Hämtad (2.20.2025)

Statista (2024) *Supercell most downloaded games 2024*. Tillgänglig på: Supercell most downloaded games 2024 | Statista Hämtad (1.31.2025)

Statista (2025). *Supercell's annual revenue from 2012 to 2024 (in million U.S. dollars).* Tillgänglig på: https://www.statista.com/statistics/298766/supercell-annual-revenue/ Hämtad (2.28.2025)

Thai, L.T (2022) *How to create intuitive game mechanics* Tillgänglig på:NTNU Open: How to create intuitive game mechanics (Hämtad 3.1.2025)

Zippia (*2023) *20+ Mobile Game Statistics [2023]: Revenue, Monetization, Downloads, Users'* Tillgänglig på: 20+ Mobile Game Statistics [2023]: Revenue, Monetization, Downloads, Users - Zippia Hämtad (3.10.2025)

Zubair, M.S & Muhammad, S. (2021) *Interactions and Actions in One Touch Gesture Mobile Games*. Tillgänglig på: (PDF) Interactions and Actions in One Touch Gesture Mobile Games Hämtad (2.20.2025)

9 Bilaga A – Källkod till ReactFast

Här nedan följer länkar till fullständig källkod för viktiga skript från implementationen av spelet ReactFast. All källkod är tillgänglig i projektets GitHub-repository:
ReactFast GitHub-repository:
https://github.com/badeberner/ExamensArbeteScripts

Kapitel 5 – Implementation

Här listas specifika skript som hänvisas till i kapitel 5 med direktlänkar:

Startup Scene (5.2):

StartupLoader.cs

Login Scene (5.3):

LoginManager.cs

SignUp Scene (5.4):

SignUpManager.cs

Menu Scene (5.5):

MenuController.cs

ProfileManager.cs

Store Scene (5.6):

StoreManager.cs

Leaderboard Scene (5.7):

LeaderboardManager.cs

Game Scene (5.8):

GameController.cs

TouchController.cs

PatternManager.cs

Ljud och Musik (5.9):

AudioManager.cs

ButtonSoundManager.cs

GameSoundManager.cs