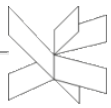


Via University College

Royal GO - ProcessReport

Nikolaj Bræmer Christen, 354322

Patrick Overgaard Blauert, 353829

Peter Hougaard, 353296

Victor Bruun Fassbender, 354361

Vejledere:

Henrik Kronborg Pedersen

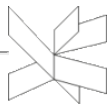
Søren Klit Lambæk

Antal Tegn: 69351

SoftWareInginør XR

2. Semester

05-03-2025 til 28-5-2025



1. Indledning

Dette semester har vi arbejdet med et projekt, der adresserer en stigende samfundsudfordring: den dalende fysiske aktivitet, især blandt unge. Med udgangspunkt i initiativerne: Bevæg dig for livet (bevaegdigforlivet, 2025) og Royal Run, har vi valgt at udforske en digital løsning, der kombinerer fysisk aktivitet og digital interaktion – en tilgang vi anser som særlig relevant i en tid, hvor unge i stigende grad engagerer sig i verden gennem deres mobiltelefoner.

Projektets mål har været at udvikle Royal GO - en mobilapplikation baseret på augmented reality (AR), der skal anvendes under Royal Run 2025 i Viborg. Appen er tiltænkt tilskuere og deltagere, som ikke nødvendigvis løber, men stadig ønsker at være aktive og engagere sig i begivenheden. For at motivere spillerne til bevægelse gennem leg, drager vi inspiration fra spil som Pokémon GO. Det gør vi for eksempel ved at bruger digitale incitamenter som: GPS-baserede aktiviteter, kosmetiske og belønninger.

Brugeren af Royal GO kan bevæge sig rundt i Viborg og finde virtuelle kronjuveler og en kongekrone, på forskellige steder i bybilledet. Disse objekter kan udløse minispil og belønninger i form af in-game valuta, som kan bruges til at købe kosmetiske AR-effekter såsom face filters. De sjove og spændende filtre er velegnet til selfies og kan deles på sociale medier. Strategien for Royal GO er at øge engagement og fysisk bevægelse gennem gamification og sociale elementer.

Projektet blev gennemført i en SCRUM-baseret ramme, hvor arbejdet blev opdelt i sprintforløb med faste stand-up møder. Denne struktur gav os mulighed for at følge fremdrift, tilpasse løbende og sikre en høj grad af gruppemedlemmernes involvering. Projektets udvikling- og samarbejdsproces blev dokumenteret gennem sprint-backlogs og logbøger, som samlet gav et overblik over forløbet. Versionsstyring og samarbejde blev håndteret via GitHub, hvor vi benyttede branches, inklusivt en validate branch, for at sikre en struktureret og dokumenteret udviklingsproces.

Vi har arbejdet med projektet i et problembaseret læringsmiljø (PBL), som har stillet krav til både faglig fordybelse og evnen til at samarbejde professionelt i en gruppe. Vores tilgang og metoder har løbende udviklet sig i takt med projektets fremskridt, og gennem dette arbejde har vi opnået værdifuld erfaring med både teknisk udvikling, samarbejdsdynamikker og refleksion over egen læring.

2. Gruppearbejde

I projektet arbejdede vi tæt sammen i en gruppe bestående af 4. Peter, Victor, Nikolaj, Patrick, hvor vi hver især bidrog med forskellige kompetencer og faglige styrker. Gruppen bestod af medlemmer med forskellige tekniske baggrunde, hvilket gav anledning til en dynamisk samarbejdsform, hvor opgaver og ansvarsområder blev fordelt i forhold til individuelle kompetencer og interesser.

Eventuelle konflikter blev løst konstruktivt gennem åben dialog og respekt for hinandens perspektiver. Vi oplevede generelt en høj grad af engagement, og der var ikke tydelige tilfælde af social loafing. Vi holdt regelmæssige statusmøder for at koordinere kode og sikre fælles fremdrift.

Nikolaj

Mit primære bidrag til projektet har blandt andet været at teste teknikken af på eventuelle løsninger for udviklingen af ny funktionalitet i et sprint. Herunder har jeg været dybt nede i programmeringen af koden, samt hjælpe til med at designe den generelle arkitektur af projektet, yderligere har jeg fungeret som sparringspartner for de andre gruppemedlemmer, og været med til at finde løsninger på de udfordringer der er opstået undervejs. Jeg har siddet med ansvar for den dybe forståelse af hvordan vi har kunnet få kombineret mange forskellige AR funktioner heriblandt Face Tracking, GPS- og kompasfunktionalitet.

Yderligere har jeg løbende fungeret som "den eksperimenterende" i gruppen, samt som sparringspartner til de skiftende "project owner" for at hjælpe dem med at tage beslutninger omkring hvilke implementationer der med fordel kunne kigges ind i for et givende sprint. Dette har lænt sig meget op af min røde personlighed, hvor jeg ofte ønsker stor indflydelse for hvordan et projekt skal være, dog har jeg forsøgt at dæmme disse tendenser til tider, for at give plads til udfoldelse fra de andre gruppemedlemmer.

Til sidst har jeg bidraget til gruppen med at sætte mig ind i teknikaliteten omkring at håndtere sådan et projekt med brug af GPS og Kamera på en korrekt, lovlig måde, da dette ville være et krav for hvis systemet skulle kunne udgives en dag, samt generelt at overholde GDPR-lovgivningen. Dette har jeg haft fordel ved, grundet tidligere arbejde i forsvaret, hvor gennemlæsning af "særlovgivning" og "reglementer" var en del af arbejdet, og derfor har jeg haft en god forståelse for hvordan lovgining skal fortolkes og håndteres i sådan et projekt.

Patrick

Mit primære bidrag lå inden for design og teknisk implementering af brugergrænsefladen samt funktionalitet i simulatoren Royal-Go. Jeg var ansvarlig for udviklingen af mobilapplikationens UI, hvor jeg designede startmenuen og implementerede navigation mellem funktionerne Skattejagt, Skattekammer og Butik i Unity ved hjælp af C#. Derudover skabte jeg en visuel identitet for applikationen, herunder logo-design og valg af farvepalette, med fokus på at skabe en sammenhængende og appellerende brugeroplevelse.

I samarbejdet arbejdede jeg desuden på udviklingen af butiksscenariet, hvor jeg udviklede et konfigurerbart shop-system, designet til mobilvisning, og integrerede det med spillets valuta og AR-komponenter. Jeg arbejdede iterativt og tilpassede løsningen undervejs, hvilket førte til vigtige læringer om systemintegration og runtime-datahåndtering i Unity.

For at styrke brugeroplevelsen og den langsigtede spilprogression implementerede jeg et point-system med principper fra SOLID-arkitektur. Systemet blev koblet til et leaderboard, hvor spillerens score vises og sammenlignes automatisk. Jeg stod for integrationen af API-baseret highscore-visning samt tilpasning af UI til det samlede design.

Jeg påtog mig rollerne som "Implementør", hvor jeg fokuserede på at omdanne ideer til funktionelle løsninger og bidrog med teknisk viden inden for UI og spiludvikling i Unity.

Victor

Jeg har primært i gruppearbejdet stået for bearbejdning og opbevarelse af gps og kompas-data. Meget af min tid har været brugt på at opstille arkitekturen for spillets databehandlingssystem. Dette navigationssystem er designet ud fra at lade andre dele af spillets funktioner, såsom instantiering af objekter på punkter, nemt tilgå et behandlet og letlæseligt datasæt.

Jeg har desuden været ansvarlig for et af disse systemer, altså visualiseringen af spillerens nuværende retning samt retningen af det næste punkt spilleren skal bevæge sig mod. Blandt dette system er også logikken for at fjerne punkter ved spillerens ankomst og udregne hvilket næste punkt er tættest på.

Til sidst har jeg sammen med Peter lavet implementationstests af diverse systemer i arkitekturen. Her har jeg været ansvarlig for at teste det visuelle system af spiller- og lokationsretninger.

Peter

Mit bidrag til gruppearbejdet kan opsummeres i tre punkter: forsøge at fuldføre de opgaver jeg har påtaget mig, være tilgængelig for de andre til hjælp / sparring, samt opfordre til en struktureret kommunikation.

Jeg har altid belært mine spejdere om at man først skal være dygtig til at være patrulje medlem, før man bliver leder, og det forsøger jeg at leve efter. Det betyder at jeg stræber efter at yde min bedste indsats, og fuldføre de opgaver jeg er blevet stillet, eller har taget. Hvis dette viser sig ikke at være muligt, har jeg taget kontakt til gruppen og diskuteret mulige løsninger.

Hvis andre har haft brug for sparring har jeg altid afsat tid, også selvom det er ubelejligt. Jeg har selv brug for at snakke mig til mange af mine løsninger, og vil derfor gerne være åben for at andre kan få mulighed for dette. I netop den forbindelse tog vi en beslutning om primært at arbejde hjemmefra, da hjælp og sparring hurtigt, og ofte, blev mere til hyggesnak.

Jeg har forsøgt at taget intitativ til at overholde vores daily scrum møder. En handling der ikke var svært, da alle bakkede op om det. Under mødet har jeg været refferant og ordstyrer, så vi alle kunne komme gennem mødet så nemt og effektivt som muligt, og have en aftale for den kommende dags arbejde.

Disse tre punkter: personlig bidrag, sparring, og daglig kommunikation lægger ikke langt fra min grøn-gule personlighed. Når det kommer til personligt bidrag ved jeg at jeg kan være flyvsk og let distraheret, så jeg gør en stor dyd ud af at få afstemt med de andre om præcist hvad jeg skal lave, så jeg ikke afviger alt for meget. Ellers så nyder jeg generelt at snakke med mennesker, så kommunikative roller falder mig let.

3. Projektopstart

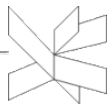
Projektets opstartsfase blev indledt med en fælles idéudvikling og diskussion af det overordnede problemfelt: fysisk inaktivitet blandt børn og unge, samt hvordan augmented reality (AR) kan anvendes til at fremme bevægelse i en moderne digital kontekst. Her blev to idéer udvalgt som mulige løsninger på den manglende fysiske aktivitet. Den første var løst baseret på konceptet af et "Escape-Room" hvor deltagerne skal løse gåder og andet for at slippe ud. Dog blev denne idé nedprioriteret grundet at det i større fald er en mere statisk aktivitet hvor brugeren ikke bevæger sig i et større område, et problem som den valgte idé overkom med et stort spilleområde.

For at sikre et godt samarbejde og prøve at opstille konfliktløsninger får de opstår, lavede gruppen en gruppekontrakt. Dette var også med til at forventningsafstemme i forhold til hvor meget arbejde det forventes for hver enkelt medlem at bidrage med.

Til fælleskommunikation, møder og resource-deling blev en Discord server oprettet. Her holdes de daglige scrum møder samt møder opsat på tværs af gruppemedlemmer i forhold til eventuel fejlfinding eller sparring. Discord serveren var det primære redskab til deling af kilder og information under skabelsen af problemfeltet hvor alt kunne samles ét sted før den endelige Github struktur var sat op.

For at skabe overblik over projektets omfang og brugernes behov, arbejdede vi med user stories. Dette gav os en struktureret tilgang til at identificere nøglefunktioner i appen, og det dannede grundlag for en mere præcis kravspecifikation. Brugen af user stories i opstartsfasen hjalp os med at fokusere på brugercentreret design og sikrede, at vi fra start havde en klar forståelse af, hvordan applikationen skulle anvendes i praksis.

Et vigtigt teknisk valg i opstartsfasen var beslutningen om at anvende Unity version 6.3.6f1 som udviklingsplatform. Dette valg blev truffet, da det på daværende tidspunkt var den mest stabile og opdaterede version med understøttelse af AR-funktioner, der var nødvendige for appens funktionalitet. Valget af platform blev støttet af tidligere erfaringer i gruppen og tilgængelig dokumentation, som gjorde det muligt at komme hurtigt i gang med prototyper og funktionelle tests.



Sideløbende med de tekniske beslutninger udarbejdede vi en overordnet tidsplan, hvor projektet blev opdelt i sprint-forløb med milepæle og opfølgningspunkter. Denne planlægning dannede grundlaget for vores SCRUM-baserede tilgang og hjalp os med at strukturere og styre projektets fremdrift fra starten.

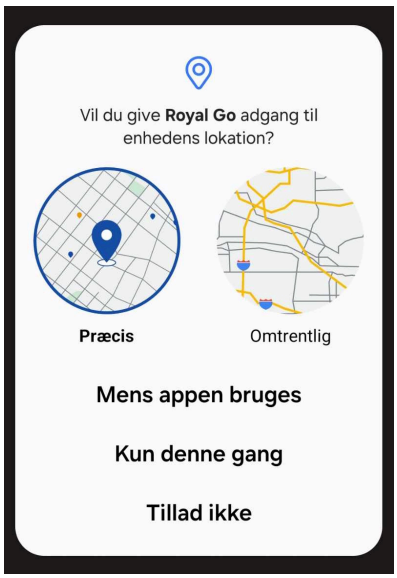
Med disse beslutninger draget, oprettes et Github repository samt en mængde "branches". Heriblandt Main (Det endelige resultat af projektet), Validate (Hvori nyskrevent kode bliver lagt til validering af anden gruppemedlem før push til Main) og en branch for hvert gruppemedlem til at lave prototyper, testing og implementation. Arbejde foregik primært individuelt under egne branches for at undgå "Merge Conflicts", altså problemer ved sammenfletning af arbejde, især Unitys metadata som kan volde store problemer og eventuel tab af arbejde.

Til rapportskrivning blev gruppen enige om at de ellers tilgængelige midler såsom Word og Google Docs ikke altid var lige attraktive. Med henblik på også at inkludere viden fået fra semesterets web-udviklingsfag blev det besluttet at lave rapporter ved hjælp af viden fået fra det nævnte fag, heriblandt html, css og javascript kode. I blandt javascript koden er logik for at omdanne dokumentets syntax og opbygning til at overholde de gældende krav for afleveringen. Rapporten, scrum-dokumentation blandt andet møder, diagrammer og bilag deles derfra via en seperat Github repository.

4. Projektudførelse

Tilgåning af GPS Lokation

Da applikationen gør brug af GPS-lokationer ved at tilgå GPS-signaler fra brugerens enhed, er det essentielt, at denne funktion håndteres i overensstemmelse med gældende lovgivning, herunder persondataforordningen (GDPR). I henhold til artikel 6, stk. 1 i GDPR (GDPR.DK ApS, 2025), kræves et gyldigt retsgrundlag for behandling af personoplysninger, herunder lokaliseringsdata. For at sikre, at behandlingen sker lovligt, tages der udgangspunkt i Digitaliseringsstyrelsens "Huskeregler for app-udbydere"(Huskeregler for App-Udbydere, 2025), hvor det blandt andet anbefales at være gennemsigtig omkring anvendelsen af GPS samt at indhente eksplicit samtykke ved opstart af applikationen.



figur 1. anmodnning om tilladelse til GPS brug

Implementeringen sker ved, at der i AndroidManifest-filen(bilag I4) angives, at tilladelse til adgang af lokaliseringsdata er nødvendig for, at applikationen kan fungere. Derudover anvendes scriptet AccessLocationPermission(bilag I1), som først kontrollerer, om brugeren overhovedet har givet tilladelse til, at applikationer må benytte GPS-signaler. Såfremt denne tilladelse foreligger, anmoder applikationen direkte brugeren om adgang til GPS-data for den specifikke app. som anført i figur 1. Først når begge tilladelser er givet, aktiveres GPS-sporingen ved hjælp af funktionen Input.location.Start() (Unity - Scripting API: LocationService, 2025).

Af hensyn til data-minimering og for at sikre, at brugerens lokation ikke spores unødigt, anvendes den indbyggede metode OnApplicationQuit(), hvor Input.location.Stop() (Unity - Scripting API: LocationService, 2025) kaldes. Herved sikres det, at sporing udelukkende finder sted under aktiv brug af applikationen, hvilket er i overensstemmelse med GDPR's principper om proportionalitet og dataminimering.

Brug af GPS signal

I den første iteration af applikationen skal systemet kunne få adgang til oplysninger om brugerens placering og sammenligne, om brugeren befinder sig på samme sted som en given position. For at opnå adgang til brugerens latitude og longitude kan metoderne input.location.lastData.latitude og input.location.lastData.longitude anvendes (Unity - Scripting API: LocationService, 2025), når funktionen input.location.start() aktiveres. Dette muliggør, at systemet med hver opdatering kan registrere brugerens aktuelle koordinater (latitude og longitude). Derudover bør der implementeres et system til positionssammenligning, idet målet er at udvikle et succesfuldt og langtidsholdbart software. For at sikre, at systemet er i overensstemmelse med SOLID-principperne, vil Haversine-formlen (GeeksforGeeks, 2025) blive anvendt. Denne formel beregner afstanden mellem to punkter på en globus baseret på forskellene i latitude og longitude, hvilket sikrer mere præcise resultater, især over større afstande.

Anvendelsen af Haversine-formlen vil yderligere muliggøre en fremtidig udvidelse af systemet til at dække større geografiske områder, hvis dette skulle blive nødvendigt. Når afstanden er beregnet ved hjælp af Haversine-formlen, kan den sammenlignes med en forudbestemt maksimal afstand, som systemet accepterer. Denne fremgangsmåde kan nemt udvides til at inkludere funktionalitet, der muliggør instantiationen af objekter på specifikke positioner.

Navigationssystem

Navigationssystemet er designet ved hjælp af Facade-mønsteret, til at samle de mange unikke subklasser der er nødvendige. Det betyder at klassen NavigationsSystem benytter facaden til at klasserne CompasSystem, GPSSystem, DirectionCheck og PositionCheck, med alt relevant data nemt og simpelt kan tilgås.

For at undgå en opadgående afhængighed benyttes Observer-mønstret til at sprede klassen NavigationSystemData til alle lyttere. Dennes publisher bruger en timer til at refreshe NavigationsSystemet, og dermed producere NavigationSystemData, som indeholder alt relevant data fra NavigationsSystem.

Eftersom navigationssystemet er meget vigtig for kernen af spillets koncept, skal der være ekstra opmærksomhed på ikke at skabe en "Gudeklasse". En måde hvorpå der forsøges at styre dette på er ved ikke at have noget konkret udregning i NavigationSystem, blot samling af data fra flere steder.

For at simplificere outputtet fra forskellige klasser har de oftest fået en separat klasse der håndterer deres inputs. Fx GetGPSData fra GPSSystem (Bilag I18) retunerer klassen GPSData, der er en klasse der indeholder horisontal præcision, længde- og breddegrader. Der blev diskuteret hvor vidt det var bedre at kalde de individuelle funktioner hver for sig, over at oprette en helt ny klasse for at have dataen, da det virkede lidt som at ”hælde fra en spand, til en anden”. Den endelige beslutning blev truffet da der blev enighed om at det stemte mest overens med single responsibility at have en enkelt public funktion der retunerede et sæt data.

I diskussionerne før facade mønstret blev endeligt besluttet blev der udforsket brugen af et decorator mønster. De ligner meget hinanden ved at bruge aggregation, i stedet for nedarvning, og de har begge adgang til mange classer. På samme måde kan begge mønstre håndtere mange unikke retur typer og forskellige kald af funktoner. Den endelige beslutning blev truffet ud fra at et decorator mønster er mere egnet til at udvidde en funktion i runtime, noget som projektet ikke havde behov for, da facaden starter med alle de klasser den har brug for. Derudover er det også tiltænkt at der kun er en enkelt NavigationSystem aktiveret i scenen ad gangen, så hele nedarvnings princippet decorator mønstret faciliterer er ikke nødvendigt.

Indsamling af lokations- og kompasinformation

For at imødekomme principperne bag Single Responsibility, som er en del af SOLID-arkitekturen, er indsamlingen og den efterfølgende behandling af lokations- og kompasdata adskilt i separate klasser. Denne opdeling muliggører en mere effektiv anvendelse af Unitys indbyggede "MonoBehaviour"-package, idet databehandling og tilhørende beregninger kan håndteres uden for MonoBehaviour. Dette bidrager til en reduceret belastning af brugerens enhed og sikrer en mere ressourceeffektiv drift.

Databehandling uden for MonoBehaviour

Anvendelse af Haversine-formlen til afstandsberegning

I forbindelse med betinget instansiering af kongekroner skal der verificeres, om spilleren befinder sig inden for en given radius af et referencepunkt. Denne afstandsberegning udføres ved hjælp af Haversine-formlen og er implementeret uden for MonoBehaviour i klassen "PositionCheck"(bilag I50).

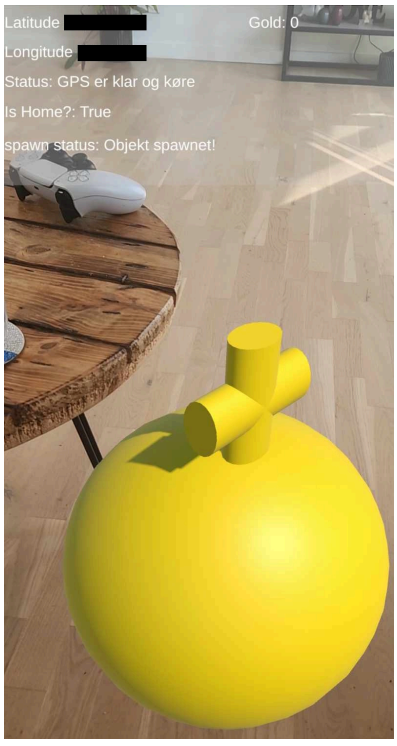
Klassen tilbyder en offentlig metode, som returnerer en bool værdi afhængigt af, om spillerens position og objektets position overholder den givne radius.

Beregning af retning ved hjælp af bearing-formlen

Udover afstanden skal også spillerens orientering verificeres for at muliggøre instansiering af kongekroner. Dette gøres ved anvendelse af bearing-formlen, som beregner vinklen mellem to punkter på en sfærisk overflade. Implementeringen foretages i klassen "DirectionCheck"(bilag I12), som er adskilt fra MonoBehaviour. Klassen indeholder en metode, der returnerer en bool afhængigt af, om vinklen mellem spillerens og objektets position befinder sig inden for en defineret grænseværdi for acceptabel retning.

opstart af augmented reality (kronjuveler)

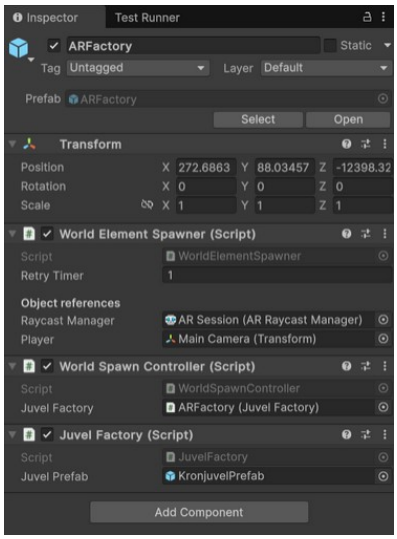
Efter at have etableret de fundamentale elementer til opbygningen af vores GPS-baserede system, blev næste skridt at implementere en basal form for augmented reality (AR). I denne forbindelse blev udviklingen af de tilfældigt genererede “kronjuveler” påbegyndt. Disse objekter er placeret i nærheden af spilleren med det formål at gøre dem let tilgængelige uden afhængighed af præcise GPS-koordinater.



figur 2. DevArt

For at opnå dette blev der udviklet et system baseret på plane-detection. Systemet anvender en raycasting-teknik, hvor der med faste tidsintervaller - defineret mellem to inputværdier i sekunder udsendes en ray fra telefonens kamera. Denne ray fortsætter, indtil den registrerer et plane med en passende størrelse, hvorefter et præfabrikeret objekt som vist på figur 2 instansieres på den fundne position. På sigt planlægges dette system optimeret ved brug af objekt-pooling, Det nuværende system instansierer dog stadig objekterne dynamisk i nærheden af spilleren.(Bilag I31)

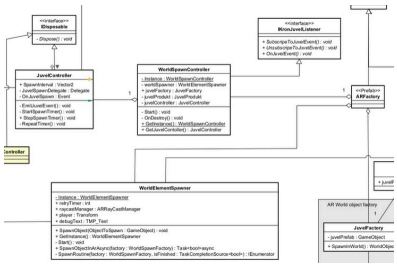
Instantieringen bliver gjordt ved brug af factory mønsteret som vist på figur 3 der i hele projektet gør grundlag for instantiering af elementer i AR-verdenen. Dette er også med til at gøre det nemt og hurtigt senere i processen at udvide systemets funktionalitet med henblik på instantiering af andre elementer, for eksempel Skatteeventet. Alle scripts som forholder sig til dette factory pattern(bilag I32, I67, I57, I33, I21, I28, I58) bliver dernæst gæmt i en prefab som gør det muligt at ændre diverse variable, såsom typen af element som instantieres.



figur 3. AR Factory

Ansvarligt for alt hvad der instantieres i scenen er "World Spawn Controlleren". (bilag I66) Den fungerer i basis som en presenter for 3D-scenen. Scriptet er i sig selv ikke synderligt komplekst men afhænger derimod af observer mønstre fra resten af modellen til at afgøre opdateringen af AR-verdenen. Scripts såsom "Juvel Controller"(bilag I31), som holder styr på instantieringen af juveler, startes af World Spawn Controlleren.

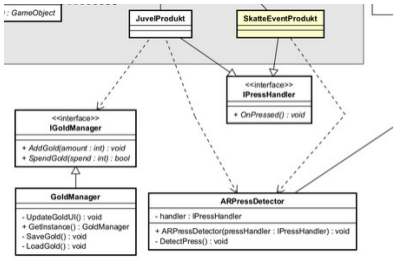
"World Element Spawner"(bilag I65), som fremvist på figur 4, er ansvarlig for at finde det rigtige tid og sted at spawne et element ind i verdenen. I AR-verdenen er der behov for en reference / flade at instantiere noget på, i forbindelse med dette er der to vigtige ting at huske: For det første skal der være et sted at spawne, og for det andet, hvis der ikke er, skal spillet forsøge at spawne når der engang bliver et sted. Det er denne logik som World Element Spawneren står for, igennem funktionen SpawnObjectInArWorld().



figur 4. World Element Spawner

Efter implementeringen af denne funktionalitet var næste skridt at definere interaktionen mellem spilleren og kronjuvelerne. Dette blev realiseret ved, at systemet registrerer spillerens tryk på skærmen og udsender en ray fra dette punkt. Hvis ray'en rammer et objekt, der har det korrekte tag (dvs. identificeres som en “kronjuvel”), aktiveres en funktion, der tilføjer en vis mængde guld til spillerens konto (bilag I33). Objektet destrueres derefter (eller senere i udviklingen returneres til objekt-poolen).

Implementationen er gjordt gennem interfacet "IPressHandler"(bilag I28) som har funktionen OnPressed() som parameter i ARPressDetectoren, der nedarver fra MonoBehaviour og som derfor skal eksistere på et GameObject. Ved instantiering af juvelen tilføjer JuvelProdukt, figur 5, ARPressDetectoren til GameObjectet som det befinder sig på.



figur 5. JuvelProdukt

I forlængelse heraf blev der også implementeret et simpelt valutasystem(bilag I17), som forbereder spillet på en fremtidig økonomifunktion. Systemet anvender Singleton Design Pattern for at sikre, at den samme instans af valuta - kaldet “guld” - anvendes på tværs af alle scener og spilsituationer. Da guld blot repræsenteres af en integer-værdi, gemmes denne ved hjælp af PlayerPrefs, hvilket sikrer, at spillerens valuta bevares på tværs af spilsessioner - også efter at applikationen er lukket. Singletonen tilgås fra JuvelProdukt ved hjælp af interfacet "IGoldManager" (bilag I24).

Samlet set danner disse systemer et grundlag for både spillets AR-elementer og økonomiske mekanismer, og de planlægges løbende optimeret med hensyn til ydeevne og brugervenlighed.

Lokationssporing og Spilområde

I denne nyupstartede iteration af systemet introduceres muligheden for sporing af specifikke positioner. Dette realiseres gennem en ny objektstruktur kaldet Location(bilag I39). Et Location-objekt indeholder tre variabler: en string, der repræsenterer navnet på positionen, samt to floats, som angiver henholdsvis breddegrad (latitude) og længdegrad (longitude). På baggrund heraf oprettes en liste, Locations, der samler flere Location-objekter og danner grundlag for videre funktionalitet, såsom søgning efter specifikke positioner.

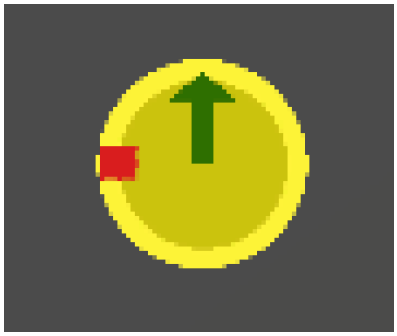
Yderligere introduceres konceptet om et "spilområde" (game area) (I68), hvilket definerer det geografiske område, inden for hvilket spillet skal foregå. Hvis en spiller bevæger sig uden for dette område, modtager vedkommende en besked, der informerer om, at vedkommende er uden for det tilladte spilleområde. For at bestemme spilområdets centrum anvendes listen af Locations. Systemet itererer gennem samtlige Location-objekter og beregner gennemsnittet af deres latitude- og longitude-koordinater. Resultatet bliver et centralt punkt, som repræsenterer midten af spilområdet. Ved hjælp af dette punkt kan systemet foretage en afstandsberegning mellem spilleren og centrum. Hvis afstanden er mindre end en foruddefineret radius, betragtes spilleren som værende inden for spilområdet; overskrides denne radius, anses spilleren for at have forladt området, og vil derved bliver informeret vedrørrende dette.

Bestemmelse af næste lokation

Når spilleren når den tilsigtede lokation og kongekronen initisaliseres, er det nødvendigt for systemet at skifte målet for spilleren til et nyt. Både for at undgå yderligere end 1 initialisering af kronen men også for at spilleren kan udforske alle punkter på listen. Til dette er der implementeret en funktion i navigationssystemet(bilag I45) kaldet "RemoveAndSetNewTarget()". Her slettes først det nuværende "target" fra listen over placeringer før det nye bliver valgt. Funktionen udnytter den førnævnte klasse "positionCheck"(bilag I50) som bruger Haversine-formlen til at give en afstand mellem punkter. Med dette bruges en for-løkke til at køre i gennem alle punkter i listen over placeringer og finder den mindste afstand fra spillerens nuværende position.

Visualiering af spiller- og lokationsretninger

For at kunne indikere en retning for spillerne at gå, kan en blanding af det nuværende mål og dets position, spillerens enheds kompas-retning og retningen til målet fra spillerens position bruges. Alt informationen kan tilgås ved hjælp af event systemet som trækker data fra navigationssystemet(bilag I45). I en UIPresenter kaldet "DirectionPresenter"(bilag I13) opdateres to UI elementer med rotationsværdierne for spillerens orientation og retningen til næste mål. Pilen indikerer spillerens retning mens den røde indikator viser retningen til målet.


figur 6. DirectionPresenter

Logo-design

Projektet omfattede desuden design af applikationens visuelle identitet gennem udviklingen af et logo. Logoet blev udformet med en kongekrone placeret over navnet Royal-Go, sat i en serif-skrifttype for at understøtte et klassisk og eksklusivt udtryk. Farvevalget blev foretaget med henblik på at forstærke appens tematiske fokus på luksus og eventyr. Følgende farver blev anvendt: guld (#FFD700), kongeblå (#4169E1) og baggrundsblå (#3E7ACB). (Se vist på figur 7)

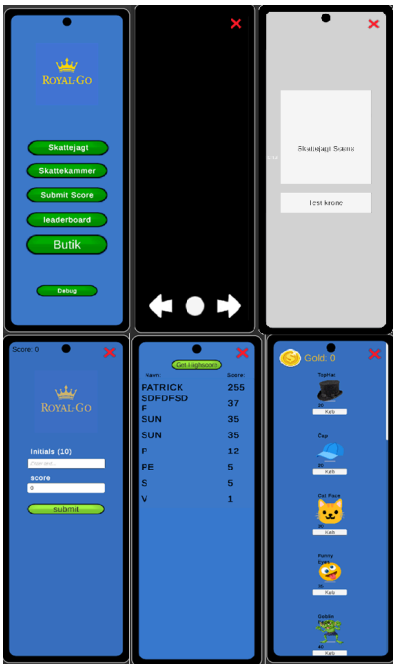
figur 7. Logo

Sammenfattende blev der lagt vægt på at sikre sammenhæng mellem æstetik, funktionalitet og tematik, således at designvalg og tekniske løsninger understøttede applikationens overordnede formål og brugervenlighed.

Simulator (UI):

Udviklingen af mobilapplikationen Royal-Go's UI blev gennemført med fokus på design, brugergrænseflade og visuel identitet. I udviklingsfasen blev der udarbejdet en funktionel og intuitiv brugergrænseflade målrettet mobile enheder. Startmenuen blev designet til at give adgang til fem centrale funktioner: Skattejagt, Skattekammer, Submit Score, Leaderboard og Butik. Navigation mellem disse scener blev realiseret via et Scripts ” MainSceneChanger”(bilag I41) i Unity, der kontrollerer sceneovergange baseret på brugerens input.

Et simpelt og genkendeligt design blev anvendt med tydelige knapper, grønne farveindikatorer for interaktive elementer og ikon baserede navigationssymboler, hvilket understøtter brugervenlighed på mobile enheder (se figur 8). Et indkøbskurvsikon blev i starten tilføjet som en visuel og funktionel knap til butiksscenen for at styrke brugerens mentale model af appens funktioner.



figur 8. UI

Selve funktionaliteten blev bibeholdt, men det grafiske ikon blev ikke videreført i det endelige produkt, da det ikke passede ind i den visuelle helhed og appens øvrige designstil. UI’et anvender fortsat en vertikal struktur med letgenkendelige knapper og tekst, der guider spilleren gennem handlinger som køb, scoreindsendelse og visning af highscore.

Butiksscenen er designet med et simpelt og pædagogisk layout, hvor hvert kosmetisk item (f.eks. hat, maske, ansigt) vises med billede, pris og en “Køb”-knap. Guldmængden vises øverst i interfacet, hvilket giver brugeren konstant feedback på deres valuta. Scroll View, Vertical Layout Group og Content Size Fitter blev anvendt til dynamisk tilpasning af butiksindholdet, så det fungerer hensigtsmæssigt på skærme med forskellige opløsninger.

I Leaderboard-delen vises spillernes navne og scores med brug af TextMeshPro-komponenter og en dynamisk opdatering via en ekstern API. Submit Score-scenen er blevet tilpasset, så spillerens score automatisk hentes og vises uden manuel indtastning, hvilket forbedrer både dataintegritet og brugeroplevelse.

Der blev også integreret en Debug-knap i Startmenuen. Denne knap gjorde det muligt at teste funktionalitet hurtigt under udviklingsfasen og var et effektivt redskab til fejlfinding og validering af funktioner i realtid.

Til den tekniske implementering blev C# anvendt. For at muliggøre hurtig prototypeudvikling blev navigation og funktionalitet hardkodet, velvidende at denne metode er mindre skalerbar i større produktionsmiljøer, men egnet til en effektiv tidlig version. Denne tilgang gjorde det muligt hurtigt at afprøve og evaluere interaktioner og flow mellem UI-komponenter.

Brugergrænsefladen er udformet med enkle former og farver for at sikre visuel klarhed, især på mindre skærme. De anvendte designprincipper – som konsistens, visuel feedback, og minimal kognitiv belastning – understøtter en brugervenlig og funktionel oplevelse, hvilket har været centralt i designstrategien for Royal-Go. (Alle nævnte elementer er illustreret i figur 8)

Brugergrænseflade (UI)

Brugergrænsefladen er designet ved hjælp af Model-View-Presenter (MVP) mønsteret. Her tænkes der på view delen som klasserne fra Unitys UnityEngine.UI namespace, hvilket er hvad der bruges til at designe udseendet til i Unitys builder. Presenteren bliver derfor primært vedhæftet det GameObject som indeholder det UI view den skal arbejde med.

Den primære kontakt til presenteren fra modellen vil være ved gennem events, for at få så løs en kobling til modellen som muligt.

Udvikling af ButikScene

Under udvikling af en shop-funktion i Unity, hvor spilleren kan købe kosmetiske genstande for en opsamlet valuta (guld). Udviklingsprocessen har bestået af en iterativ tilgang, hvor funktionalitet gradvist blev udvidet og testet i sammenhæng med brugergrænseflade og gameplay-integration.

Projektet blev startet med et færdigt "gold script"(bilag I17), der håndterede spillerens valuta, inklusive lagring og visning af guld. Dette script dannede fundamentet for hele shop-systemet. Dernæst blev et ShopManager-script(bilag I56) udviklet med målet om at undgå hårdkodning og i stedet lade shop-items konfigureres via Unity-editoren. Hvert item kunne defineres med navn, pris og billede, hvilket gjorde løsningen både fleksibel og skalerbar i forhold til fremtidig udvidelse.

Gold-systemet(I17) blev senere omstruktureret til at følge en mere SOLID-venlig arkitektur. Et interface, IGoldManager(I24), blev introduceret for at abstrahere logikken for at tilføje, bruge og hente guld. GoldManager-klassen(I17) implementerer dette interface og indeholder ansvar for datahåndtering (PlayerPrefs), UI-opdatering og Singleton-initialisering. Denne opdeling giver en klar separation af ansvar og øger genbrugelighed og testbarhed i koden.

UI blev designet med mobilanvendelse i tankerne. Herunder blev en Scroll View konfigureret korrekt til mobil, og shop-items blev lagt ind som prefabs med Vertical Layout Group og Content Size Fitter, så visningen automatisk tilpasser sig forskellige skærmstørrelser. Derudover blev der arbejdet med responsiv opbygning, hvor knapper og layout automatisk justeres afhængigt af opløsning og format, hvilket sikrer en ensartet oplevelse på tværs af enheder.

Købsfunktionen blev koblet til ShopManager(bilag I56), så det automatisk valideres, om spilleren har nok valuta til at gennemføre et køb. Systemet blev desuden udbygget, så det let kunne udvides med yderligere logik – eksempelvis opdatering af en kosmetik-liste, eller integration med andre scener og systemer i spillet.

Udvikling og Justering af AR-Produkter til Virtuel Butik

Efter afslutningen af den indledende udvikling af butikken blev fokus rettet mod implementeringen af de produkter, som brugerne kunne købe og interagere med. Den oprindelige intention var at benytte Full Body Tracking til at skabe virtuelle "kostumer", som brugerne kunne iføre sig og tage billeder af i augmented reality (AR). Efter adskillige timers arbejde viste det sig dog, at Unitys dokumentation - i det med småt - angav, at fuld kropssporing udelukkende understøttes på iOS-enheder((Body Tracking | Unity MARS | 1.3.1, 2025)).

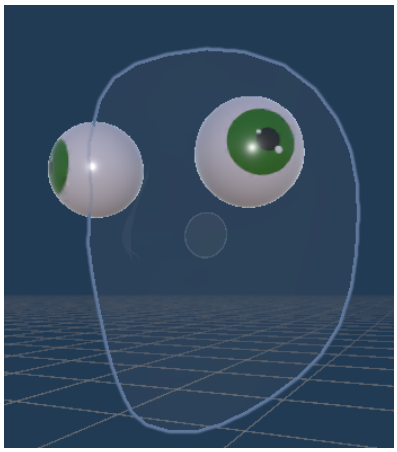
Da projektets målplatform udelukkende anvender Android-enheder og gør brug af Google ARCore, var denne funktionalitet desværre ikke anvendelig. Projektgruppen måtte derfor revurdere strategien og vendte tilbage til tegnebrættet. Det blev besluttet at anvende Face Tracking som alternativ.

Integration med ARFaceList

Som en del af shop-funktionaliteten blev der arbejdet på at integrere købte AR-face-prefabs direkte i en ARFaceList(bilag I5), som bruges af AR-systemet i Skattekammer-scenen. Løsningen gjorde det muligt for spilleren at købe kosmetiske ansigtsgenstande (f.eks. masker), som derefter blev tilføjet automatisk til en liste (ScriptableObject). Denne liste blev brugt af FacePrefabSelector(bilag I14) i AR-scenen til at vise og skifte mellem de købte ansigter. Shoppen blev dermed direkte koblet til AR-oplevelsen, og implementeringen understøttede dynamisk opbygning uden hårdkodning.

Denne tilgang tillod stadig brugeren at tage billeder af sig selv i AR, hvilket stemte overens med kravspecifikationen, hvor metoden til at tage billedet ikke var nærmere defineret.

For at påbegynde implementeringen af Face Tracking blev Unitys Learning Path(Create a Basic Face Filter - Unity Learn, 2021) for teknologien gennemgået. Herefter blev der udviklet en klasse til håndtering af ansigtsfiltre. Denne klasse gjorde det muligt at tilføje og bladre mellem forskellige filtre. For at gøre systemet mere skalerbart og integrerbart med butikken, blev filtrene struktureret som ScriptableObjects, der indeholder en liste over tilgængelige filtre. En metode til dynamisk at tilføje nye filtre, som figur 9, blev tilføjet, hvilket gør det muligt at knytte købte varer til filtrene, således at et filter bliver tilgængeligt for brugeren efter køb.


figur 9. ARFace

Efterfølgende blev der udviklet en funktionalitet til at tage billeder og gæmme dem i brugerens kamerarulle. Et tidligt teknisk problem bestod i, at UI-elementerne, som anvendes til at tage billeder, fortsatte med at være synlige på det færdige billede. Dette blev løst ved at konvertere fotofunktionen til en IEnumerator-metode, som sikrer sekventiel udførelse. Inden selve billedet tages, sættes UI-elementernes alfaværdi til 0 (gennemsigtige), og efter billedet er gemt, sættes alfaværdien tilbage til 255 (fuldt synlig). Billedet gemmes via en platformsspecifik filsti tilpasset Android/Samsung-enheder, og brugeren modtager samtidig en Toast-meddelelse med bekræftelse af, at billedet er gemt(bilag I55).

Afslutningsvis blev AndroidManifest.xml(bilag I4) opdateret, så applikationen eksplicit anmoder om tilladelse til at benytte kameraet - analogt med hvordan tilladelser til lokationstracking håndteres. Dette skridt er taget for at sikre overholdelse af gældende lovgivning vedrørende brugen af kamera og persondata.

Implementering af tværgående funktionalitet mellem scener: Butik og AR-ansigtsfiltre

I dette sprint blev der fokuseret på at udvikle funktionalitet, der forbinder eksisterende systemer på tværs af scener, nærmere bestemt interaktionen mellem butiksscenen og "skattekammeret", hvor brugeren har mulighed for at benytte ansigtsfiltre i et AR-miljø. Formålet var at gøre det muligt for brugeren at købe ansigtsfiltre i butikken og efterfølgende anvende disse i skattekammeret til at tage billeder.

Implementeringen blev realiseret gennem en metode, der først validerer, om brugeren har tilstrækkelige midler i form af "guld" (håndteret via et eksisterende Gold-script) til at foretage et køb. Herefter undersøges det, om den valgte vare er gyldig. Hvis begge betingelser er opfyldt, tilføjes det pågældende ansigtsfilter (repræsenteret som et prefab) til en liste over tilgængelige filtre for brugeren. Denne funktionalitet blev realiseret ved hjælp af scriptet ARFaceFilters, som håndterer brugerens samling af filtre.

Et væsentligt aspekt af denne opgave var at sikre persistens, så købte varer forbliver tilgængelige på tværs af sessioner. Da Unitys PlayerPrefs og JSON-format ikke understøtter direkte lagring af GameObject-referencer, blev der i stedet valgt en løsning, hvor navnene på de købte filtre gemmes som strings i JSON-format. Disse gemmes i PlayerPrefs, og ved genstart af applikationen indlæses navnene og matches mod en statisk liste over alle tilgængelige ansigtsfiltre(bilag I5) i systemet. Ved et match rekonstrueres brugerens samling, hvilket sikrer, at købte varer forbliver tilgængelige.

Når et køb er gennemført, fratrækkes den relevante mængde guld fra brugerens saldo. Det købte produkt fjernes derefter fra butikkens udbud, og butikken opdateres, så kun ikke-købte varer vises. Købsstatussen gemmes ligeledes i PlayerPrefs, hvilket forhindrer brugeren i at købe den samme vare mere end én gang.

Denne løsning understøtter således både funktionel integration mellem separate scener og sikrer datavedholdelse for brugerens købte indhold, hvilket er centralt for en konsistent brugeroplevelse.

AR World Scene

AR world scenen er alt hvad der vises af 3d rum på telefonen, når AR er aktiveret. For at have et centralt sted at tilføje og fjerne ting fra denne scene er Factory mønsteret oplagt.

Klassen WorldSpawnController (bilag I66) er ansvarlig for oprettelsen af instantieringen af 3d modeller fra Skatte event og Juvel event i AR scenen. Factory controlleren (bilag I32) har subscribet til en række events, der signalerer hvad og hvor noget skal spawne. Factory mønsteret sikrer, at der er en nem og stabil måde at udvide systemet i fremtiden, fx hvis der skulle implementerers en AR-rutevejledning.

Der var ikke specielt meget diskussion om hvordan dette burde implementeres, da factory mønstret er åbenlyst velegnet til formålet.

Implementering af miniGame System

I dette sprint blev det besluttet at fokusere på at færdiggøre selve kernen af spiloplevelsen - det, vi betegner som "skattejagten". Den primære målsætning var at etablere funktionaliteten omkring interaktionen med en kongekrone, som fungerer som udgangspunkt for en række minispil.

Når spilleren interagerer med kronen (bilag I69), skal et tilfældigt minispil aktiveres, hvilket, ved succesfuld gennemførelse, belønner spilleren med en vis mængde guld.

For at sikre systemets fleksibilitet og fremtidige udvidelsesmuligheder blev der valgt en arkitektonisk løsning baseret på Strategy Pattern. Denne tilgang muliggør nem integration af nye minispil uden behov for at ændre den eksisterende logik. Et centralt element i implementeringen var en MiniGameManager (bilag I44), som automatisk identificerer og registrerer alle spil, der implementerer det definerede IMiniGame interface (bilag I27). Ved aktivering udvælges et af disse spil tilfældigt.

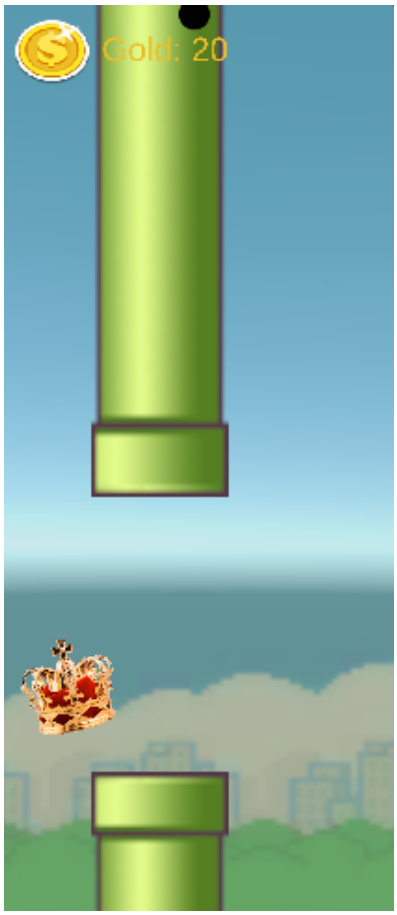
Eftersom alle spilkomponenter er afhængige af Unitys MonoBehaviour, blev der oprettet en abstrakt klasse, MiniGameBase (bilag I42), som fungerer som bindeled mellem IMiniGame (bilag I27) og de konkrete spiltyper. Denne løsning muliggør, at minispil både kan nedarve fra MonoBehaviour og samtidig overholde IMiniGame interfacet(bilag I27).

Valg og Implementering af Minispil

Til at begynde med blev der udviklet tre minispil, som alle tager udgangspunkt i kendte spilmekanikker for at sikre høj brugerforståelse og lav indlæringskurve. Disse er:

- **KingClicker:** inspireret af "Cookie Clicker
- **FlappyKing:** inspireret af "flappy Bird
- **CatchKing:** inspireret af klassiske "dropper"-spil

Alle minispil blev implementeret som UI-elementer, hvilket reducerede den tekniske kompleksitet, men medførte visse begrænsninger i brugen af fysiske komponenter såsom colliders og transforms.



figur 10. FlappyKing

KingClicker

Dette var det første spil, der blev implementeret. Spillet går ud på, at spilleren optjener guld ved at klikke gentagne gange på en kongekrone inden for en begrænset tidsramme. Tidens udløb vises via en grafisk UI-slider. Spillet er simpelt i sin mekanik, men tjener som en effektiv introduktion til minispilssystemet.(bilag I34, I61)

FlappyKing

Dette spil er en UI-baseret version af "Flappy Bird", som vist på figur 10, hvor spilleren styrer en faldende kongekrone ved at trykke på skærmen for at få den til at stige. Målet er at navigere gennem rør og optjene guld. En teknisk udfordring i dette spil var, at UI-elementer i Unity ikke understøtter klassiske fysiske komponenter som colliders. Derfor blev logikken implementeret ved hjælp af RectTransform overlap-tjek frem for fysiske kollisioner. På trods af dette lykkedes det at skabe en oplevelse, der minder om originalen. (bilag I15, I16, I47, I48)

CatchKing

Det sidste spil, der blev udviklet i denne sprint, var CatchKing. Her falder både kroner og bomber ned over skærmen, og spilleren skal forsøge at samle så mange kroner som muligt uden at røre bomberne. Spillet benytter sig af en Object Pooling-mekanisme til at håndtere de faldende elementer effektivt. En udfordring her var at sikre korrekt interaktion mellem objekterne i puljen og brugerens input, hvilket dog blev løst tilfredsstillende. (bilag I7, I8, I40, I49)

Integration og Visuelle Forbedringer

Efter implementeringen af de individuelle spil blev MiniGameManager(bilag I44) integreret med kongekronen i spilverdenen. Det betyder, at hver gang en krone genereres i spillet, medfølger funktionaliteten til at starte minispil direkte ved interaktion. Derudover blev der udviklet en separat klasse, der håndterer rotationen af kongekronen (bilag I70) for at give en animeret og mere iøjnefaldende præsentation, hvilket bidrager til et mere levende og dynamisk visuelt udtryk.

Det samlede resultat af sprintet er et fleksibelt og skalerbart minispilssystem, der gør det muligt for udviklere nemt at tilføje nye spil blot ved at implementere det eksisterende interface og arve fra den abstrakte MiniGameBase klasse(bilag I42). Denne arkitektur sikrer en nem vedligeholdelse og udvidelse af systemet i fremtidige iterationer og understøtter videreudviklingen af den samlede spiloplevelse.

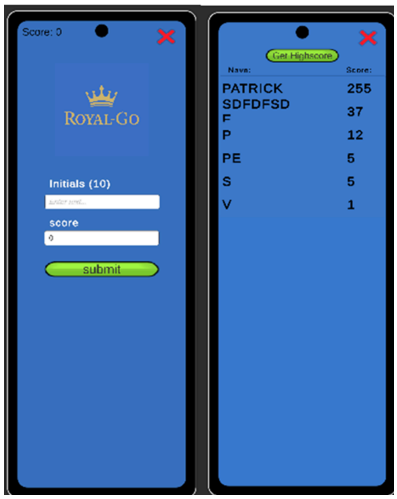
Implementering af point-system (ScoreTracker)

For at udvide shop-systemet og øge spillerens motivation blev et nyt pointsystem implementeret (bilag I19) sideløbende med det eksisterende guld. Idéen bag systemet var, at spilleren optjener point hver gang der optjenes guld, men uden at miste disse point ved køb. På den måde fungerer point som en form for "samlet optjent guld", som giver et billede af den langsigtede progression.

Systemet blev designet ud fra SOLID-principper – særligt Single Responsibility og Dependency Inversion. Logikken blev derfor delt op i interfaces: IAddScore(bilag I20) og IGetScore(bilag I23), og selve tracking-klassen ScoreTrackerSeparated blev isoleret fra UI. UI-delen blev håndteret af en separat ScoreTrackerPresenter(bilag I52), som kommunikerer via SetPresenter metoden. Denne opdeling gør systemet lettere at vedligeholde og udvide, f.eks. hvis man senere vil vise point på flere UI-elementer eller skifte lagringsmetode.

Leaderboard og integration med point-systemet

Et leaderboard blev tilføjet som en afsluttende del af point-systemet, så spilleren kan sammenligne sin score med andre. Dette system blev bygget ud fra en Unity-pakke leveret af Pelle, som indeholdt scripts som SubmitScorePanel(bilag I62), ScoreView(bilag I54), ScoreList(bilag I51), LeaderBoard(bilag I36), HighscoreData(bilag I19) og AddScore(bilag I3). (Se illustration i figur 11)



figur 11. Leaderboard

Leaderboardet blev integreret med det eksisterende ScoreTrackerSeparated(bilag I53) og ScoreTrackerPresenter-system(bilag I52) ved at ændre SubmitScorePanel-scriptet(bilag I62). I stedet for manuelt indtastede scores hentes spillerens faktiske score nu automatisk og vises i et låst felt. Dette bidrager til en mere brugervenlig oplevelse og forbedrer dataintegriteten. Til integrationen blev der benyttet en API Key og et Game ID udleveret af Pelle, som konfigureres i HighscoreData(bilag I19) for at muliggøre ekstern lagring og visning af highscores.

API Key:

6d5ad9adc2e3ecc525b288b9af5719f0b6b082352715ab8f8efa03daa012e6c3

ID:

SEP2-AR-RRI

UI'et fra pakken blev også tilpasset, så det visuelt passede med resten af projektet og fungerede på mobil. Selvom vi oprindeligt vurderede, at opgaven ville tage omkring 100 story points, viste det sig, at den reelle arbejdsbyrde var tættere på 16 story points, da koden i høj grad allerede var givet og kun krævede integration og justeringer. (Se figur 11)

Unit- og integrationstesting

Med hensigt på regelmæssigt testing, især før push af ændringer, blev implementationen af unit tests startet tidligt i projektet. Implementationen var anderledes end blot at unit teste c# scripts og der var derfor en læringskurve i forhold til vaner og routiner. En stor grund til dette var at mange af de inputs som spillet gør brug af befinder sig i MonoBehaviour klassen som er del af det aktive spil miljø. Disse inputs kan unit testing ikke tilgå og der blev derfor brugt mere tid på integrationstest i form af Unitys "PlayMode" tests, som inkluderer oprettelse af et spilmiljø.

Skattejagtsscenen bliver testet for følgende funktioner: Instantiering af scenen, opstart af navigationssystemet, signal fra juvel controllere, signal fra skatte event controlleren, instantiering af juveler, instantiering af skatte event, tilføjelse af guld og point og opdatering af visuel repræsnetation af spiller- og lokations retninger. Efter indledende fejlede forsøg lykkedes det at opsætte et simuleret test miljø, som AR suiten i Unity genkender. Dette muliggjorde yderligere at teste plane-detection samt synkronisering med orientering i AR.

Oprindeligt var alle tests defineret for sig selv, men grundet brugen af prefabs i scenen fejlede nogle af disse grundet flere instantieringer af objekter som ellers ikke ville forekomme. Derfor blev alle tests samlet i én, som sikrede at scenen og andre prefabs kun instantieres én gang hvilket undgik unødvendige fejl på tests. Testene udnytter funktionen WaitUntil() som muliggører at vente til et bestemt event sker før fortsættelse af test, hvilket gav bedre mulighed for at simulere forskellige situationer.

Fejlhåndtering

Kompas komplikationer

Den oprindelige implementation af kompasset gjorde brug af Unitys gamle "Input" system, det samme som projektet udnytter til GPS-data. Dog ved det endelige build returnede Input funktionen "magneticHeading", som koden ellers var skrevet rundt om, altid 0 uanset hvordan enheden blev holdt. Ved nærmere eftersyn viste det sig at kompas funktionaliteten var forældet og ikke længere understøttet. I denne sammenhæng blev gruppen enig om at terminere det daværende sprint (7. Sprint) for at fokusere på at lave en løsning til denne vigtige del af projektet.

Herfra blev skiftet lavet til det nye input system. Efter kig i dokumentationen blev det dog tydeligt at kompas implementationen ikke var overført til det nye input system, hvor der i stedet var en funktion der kaldte en sensor, MagneticFieldSensor, som registrer der omkringværende ambiente magnetfelt af enheden. Sensoren returnerer en Vector3, der beskriver telefonens placering ud fra magnetfeltet, hvor x, y og z repræsentere henholdsvis nord / syd aksen, øst / vest, og op / ned.

Magnetfeltssensoren har en række fordele over det 2-dimensionelle kompas. Primært den at beregning af telefonens relation til nord er muliggjordt i enhver position. Dette betyder at brugeren undgår at skulle holde enheden vandret for at finde den korrekte nord. Inklusionen af en tredje akse, z, giver også mulighed for at bestemme orientationen af skærmen, som dog ikke endte med at blive relevant for projektet.

Implementationen af kompasset afhænger, som ses på figur 12, af atan2 funktionen. Denne funktion returnerer vinklen mellem to punkter, baseret på en fuld cirkel. Implementationen er bygget ud fra guiden "how to convert magnetometer data into compass heading" (Bleything, 2023). Der oprindeligt er baseret på en Arduinos magnetfeltssensor. Efter eksperimentation blev det tydeligt at denne guides x og y akse er byttet om i forhold til retur typen af Unitys brug af sensoren. Dette betød at retningen af øst og vest på enheden var omvendt, hvilket blev løst ved at vende fortegnet på x-aksen.

```
double headingRaw =
    Math.Atan2(
        MictoTeslaToGauss(magneticField.x * -1),
        MictoTeslaToGauss(magneticField.y)
        ) * (180 / Math.PI);

if (headingRaw < 0)
{
    return headingRaw + 360;
}
else if (headingRaw > 360)
{
    return headingRaw - 360;
}
return headingRaw;
```

figur 12. Kompas kode

Efter kodningen af databehandlingen af sensorens returnering var færdig var implementationen gnidningsløs. Arkitekturen havde blot et enkelt punkt som afhang af kompas-retningen.(bilag I9, I10, I11)

Efter grundigere tests blev det dog tydeligt at udregningerne af retningen var fejlagtige. På trods af en liste af unit tests der bekræftede systemets funktionalitet. Efter eksperimentering bundede fejlen ud i en afrundingsfejl, som skyldes brugen af floats til positioner hvor forskelen mellem punkter blev for lille til præcis udregning. De førnævnte tests udnyttede alle hele float værdier og stødte derfor ikke ind i problemet. Løsningen på dette blev at skifte datastrukturen ud med double værdier samt at bruge et Math-namespace som håndterede radianer i doubles i stedet for floats, da Unitys indbyggede math system regner dem i floats. Ændringen blev lavet ved hjælp af Microsofts sprog model "Copilot" da der var tale om et systematisk og objektiv ændring af data typer som sprogmodeller excellerer i. Efter unit testing med ikke-runde double værdier var kompas systemet implementeret og funktionelt.

Afstand til punkt - Debug

Sent i projektet blev det nødvendigt at tilgå afstanden fra den nuværende position, til et bestemt koordinat. Brugen af MVP, observer og facade mønsteret gjorde det nemt tilgængeligt at tilgå informationen. Facaden tillader et simpelt adgangspunkt til den ellers omstændige funktion for afstanden mellem to punkter. Herfra var det blot at tilføje afstanden til NavigationSystemData i navigations systemet, så den gennem observer mønstret bliver distribueret til debug presenteren. Dette tillod gruppen at løbende tjekke værdien for afstanden til næste punkt, og hjalp med fejlfinding at det system.

Konstant instantiering af kroner

Ved test af instantiering af krone-prefabet, som skal ske én gang ved et bestemt punkt og enhedsretning, skete instantieringen konstant. Udover at dette ikke var den ønskede opførsel førte det også til problemer ved interaktion med objekterne da logikken bag minigames var designet ud fra et maks af 1 krone ad gangen. Der er et par måder at løse det på, fx ved at lave kronen til en enkelt instans så factorien ikke producerer mange forskellige, eller udvidde locationsmanageren til at kunne holde styr på om en location har spawnet en kongekrone eller ej. En endelig løsning er endnu ikke besluttet. På lang sigt vil der højst sandsynligt være brug for flere informationer fra locationsmanageren og derved vil udviddelsen af denne være optimal. Samtidigt, kan der opstå tvivl om en bedre løsning ikke kunne være at have en separat manager til locations, som kun tilgås når spillet kører.

Grundet denne tvivl om logikken bør være på locationsmanageren, endte løsningen med at factoriet holder overblik over om en krone er spawnet eller ej. Dette betyder at det senere vil være nemt at udvide factoriet med en object pool hvis instantieringen af kroner på flere lokationer samtidigt bliver nødvendigt.

WhiteBox test af systemet

I forbindelse med at der løbende er blevet udviklet Whitebox tests til de forskellige dele af systemet, er der også opstået et behov og en interesse til at få et eksternt test panel til at teste systemet. her er der blevet udført test af en enkelt person uden for Scrum Gruppen, "Sunneva Ros Hlynsdóttir", som har testet systemet og givet feedback på dette. Dette er dog gjordt med bevidsthed omkring faldgruber for "testpanellet" da det er en person som har en relation til et medlem i scrum gruppen, samt at testene kun er udført af en enkelt person. Dog er testene udført efter bedste even om at holde sig neutral til resultatet, samt at systemet er blevet gået igennem i samtlige test på 2 forskellige android enheder, for at være sikker på at resultatet ikke kun ville virke korrekt på én enhed.

5. Personlige refleksioner

Nikolaj

Jeg synes personligt jeg har kunne bidrage med en del til projektet, da jeg har haft stor drivkraft og motivation til at få det til at lykkedes. Blandt andet, føler jeg at jeg har været medvirkende til at skubbe projekt meget frem løbende og sørget for at vi kom videre i proccessen. Der har dog været både fordel og ulemper ved dette, blandt andet ville jeg gerne have haft at jeg fra start af havde lagt noget mere fokus på arkitekturen i projeket og sørget for at det jeg lavede tidligt i forløbet var SOLID, da det først er noget jeg begyndte af havde en bedre forståelse for senere i forløbet, hvilket betyd at ting jeg lavede tidligt i forløbet skulle laves om. Der er mine "røde" personligheds træk virkeligt kommet til udtryk. Yderligere har der også været flere fordele og udlemper i forbindelse med at jeg har den "røde" personlighed, da vi løbende har valgt at rotere mellem de forskellige roller i gruppen, hvilket til tider haft en negativ indflydelse på mig, da jeg ofte har siddet med en plan for hvordan noget skulle være i mit hoved, men så har en projekt owner valgt en anden løsning, så der har jeg skulle tilpasse mig til det, hvilket har været en udfordring men jeg synes at jeg har været god til at give plads i gruppen og tilpasse mig til det. Således at jeg ikke har brudt for meget igennem og taget styringen med det hele, dette er dog noget jeg har skulle aflære mig selv, da det ligger dybt i mig, grundet den kultur i forsvaret jeg kommer fra, hvor jeg i mit tildligere job som sergent har været vant til at tage styringen og have det overordnede ansvar. Dog syntes jeg at gruppen har været gode til at samarbejde og jeg har ikke en følelse af at der har været noget hvor jeg har taget styringen over hovedet på andre.

Igennem forløbet har jeg klart fået en bedre forståelse for værdien af både at have en god arkitektur og brugen af design patterns, da det har gjort at det krævede markant mindre ændringer i det kode jeg har lavet efterfølgende, samt det har været nemmere for gruppen både at kunne forstå og tilslutte deres egen kode til det jeg har lavet.

Særligt i den sidste del af forløbet har jeg kunne mærke mine røde personlighedstræk, da det er svært for mig at stå inde for et projekt som ikke er "færdigt", så det har medført mange timers overarbejde for at få nået så meget af projektet som muligt. Specielt i forbindelse med at i vores sidste sprint oplevede vi at en meget essentiel del af vores system ikke virkede, der var det specielt hårdt for mig at være i, da det var en del af projeket som jeg kun har været delvist involveret i og derfor ikke har kunnet være en drivende kraft i at få problemerne til at blive løst, det gav mig en følelse af hjælpeløshed, da jeg kun har kunne bidrag med enkelte ting til løsningen af problemet. og har måtte stole på at de vedrørende i gruppen har kunne været i stand til at løse det. Dette har dog betydet at jeg har taget meget ansvar i at få udarbejdet en masse andre dele af projketet, imellem tiden.

Generalt syntes jeg at vi har haft en god dynamik i gruppen og at vi har været gode til at samarbejde og hjælpe hinanden, til at komme så langt i projektet som vi er resulteret i at være kommet.

Patrick

Under arbejdet med projektet oplevede jeg, hvor vigtigt det er at udvikle systemer, der er fleksible og lette at vedligeholde. Jeg havde tidligere en tendens til at hardkode funktioner, som fx sceneskift, men under dette projekt blev det meget tydeligt for mig, hvor hurtigt det bliver uoverskueligt i større applikationer. Jeg lærte, at det er langt mere hensigtsmæssigt at arbejde med mere dynamiske løsninger, fx via event-systemer eller konfigurerbare objekter. Det er noget, jeg helt klart vil implementere tidligere i fremtidige projekter.

Jeg fandt også ud af, hvor værdifuldt det er at bruge Unitys UI-system med Scroll View, Vertical Layout Group og Content Size Fitter. Det hjalp mig med at lave en brugerflade, der tilpasser sig forskellige skærmstørrelser – især med mobiltelefoner i tankerne. Det gav mig konkret erfaring med, hvordan man kan tænke responsivt design ind fra starten, og jeg blev bedre til at arbejde struktureret med prefabs og editorbaseret konfiguration.

Under arbejdet med at forbinde shop-funktionen med AR-face-funktionaliteten lærte jeg meget om, hvordan man kan arbejde med ScriptableObjects og dynamisk datahåndtering i Unity. Det var spændende at prøve på at udvikle en løsning, hvor købte items direkte kunne integreres i AR-oplevelsen i en anden scene, uden at skulle ændre eksisterende scripts. Men på trods af at løsningen teknisk virkede, valgte vi i sidste ende en alternativ metode, der passede bedre til projektets mål. Det var en værdifuld proces, da jeg fik praktisk erfaring med designvalg og lærte at tage højde for både funktionalitet og brugervenlighed i en helhedsorienteret løsning.

Implementeringen af pointsystemet gav mig værdifuld erfaring med at tænke i mere strukturerede og skalerbare løsninger. Ved at benytte interfaces og en adskilt Presenter-klasse lærte jeg, hvordan man kan holde sin logik ren og adskilt fra visningen. Det føltes som et skridt i retning af mere professionel softwarearkitektur. Jeg blev også opmærksom på, at selv små systemer – som f.eks. et simpelt scoremodul – kan drage fordel af SOLID-principper, især hvis projektet vokser. Jeg vil tage denne læring med videre til fremtidige projekter, hvor det at strukturere kode tidligt kan spare meget arbejde senere.

Arbejdet med leaderboardet var en god øvelse i at integrere andres kode i vores projekt. Det krævede forståelse af, hvordan de eksisterende scripts fungerede, samt hvordan jeg kunne koble dem sammen med mit eget point-system. Det gav mig også en bedre forståelse for, hvordan man skriver kode, der er let at integrere – f.eks. ved brug af interfaces som IGetScore.

Overordnet føler jeg, at jeg har udviklet mig meget i forhold til at tænke mere professionelt og langsigtet i min tilgang til både UI-design og opbygningen af systemer. Jeg har fået større forståelse for vigtigheden af at adskille ansvar i koden, arbejde med genanvendelige komponenter og tænke i skalerbare løsninger fra starten. Jeg har lært af mine fejl og set konkret, hvordan små valg tidligt i processen kan gøre stor forskel senere. Det har givet mig både selvtillid og motivation til at arbejde endnu mere målrettet med at skrive robust, fleksibel og vedligeholdelsesvenlig kode i fremtidige projekter.

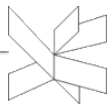
Victor

Jeg mener at jeg har bidraget godt til gruppearbejdet. I forhold til kvantitativt hvor meget kode jeg har skrevet virker det ikke som så meget, men mit arbejde indenfor strukturen og arkitekturen af databehandlingen har hjulpet til at gøre projektet mere overskueligt og stemme mere overens med SOLID-principper. Jeg føler jeg har været god til at tilbyde indsigt og hjælp i andres kode og derved hjælpe med at troubleshoote. Jeg har desuden stået for validering af størstedelen af scripts der indgår i spillet og har derfor dannet mig et godt overblik over projektet og dets indhold.

Den vigtigste viden jeg har opnået gennem projektarbejdet er hvor vigtig struktur og ordentlig planlægning er for større projekter. I starten begav jeg mig ud i at lave hvad jeg tænkte var gode systemer men da det senere blev problematisk at koble det op til resten af systemer blev det tydeligt at vi var nødsaget til at sætte os sammen og strukturere projektet.

Arbejdet med implementationstests var spændende og gav også ny læring og viden indenfor hvordan det bliver gjordt i Unity. Men det blev også tydeligt at det ikke altid var tilstrækkeligt da diverse systemer og data-typer ikke altid fungerede ligeledes ved et build til mobilen. I kommende projekt der udnytter en anden platform end PC, hvor der bliver udviklet på, vil jeg mene der skal prioriteres at teste på enheden tidligere og løbende i processen.

Generelt set synes jeg der har været et godt samarbejde i gruppen hvor alle har været gode til at påtage sig unikke roller, hvor nogen har specialiseret sig mere i bestemte systemer andre har lavet flere generelle systemer og nogle har stået mere for det struktuerelle og dokumenteringen af det. På den måde har vi kunne nå relativt langt med projektet mens det har været vel struktueret, dokumenteret og stemt overens med SOLID-principper.



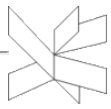
Peter

Jeg vil mene at jeg har fået et stort udbytte af dette semester projekt. Gennem hele processen har jeg joket med at jeg ikke har skrevet en eneste funktionel del af spillet, hvilket ikke er ret langt fra sandheden. Jeg har fokuseret på at re-factorere eksisterende stykker kode, og sørge for at arkitekturen og funktionerne overholder SOLID principperne bedst muligt. I den process, har jeg fået stor respekt for brugen af klassediagrammet, både som et værktøj til at kommunikere med mine medprogrammører, men også til at udvikle og løse arkitektoniske udfordringer. Brugen af mønstre har været enorm belejlig til både kommunikation, og til implementation af nye funktioner. Det har været tydeligt hvordan kode skal sammensættes, og hvad man kan forvente at få af kode.

Hen mod slutningen begyndte vi at implementere flere test. Det tog en del tid at finde ud af hvordan unit- og integrationstest-systemet i Unity fungerede, men da vi havde sat de første test op, blev det et uundværligt værktøj til at se om der var nogle umiddelbare problemer i den nye kode. Jeg vil helt sikkert benytte flere test, og endda eksperimentere med en test orienteret udvikling, da jeg i udviklingen af testene fandt mange huller / edge cases, der ellers ville have været svære at fejlsøge.

I vores gruppe besluttede vi os for at arbejde hjemmefra, da vi alle har en slem tendes til at blive distraheret oppe i klassen. I starten var det svært at koncentrere sig hjemme, og jeg blev ofte frustreret over kode, eller tekst eller andet. Derfor endte jeg tit med en følelse af ikke at have lavet noget, og generelt frustreret når min familie kom hjem. Efter et par dage med dette satte jeg mig for at få en mere organiseret arbejdsgang, og købte et spil jeg længe har overvejet ”Ithya: Magic Studies” der er en gamificeret udgave af Pomodoro metoden. Efter at have brugt den et stykke tid, føler jeg af at jeg sidder mindre fast, arbejder hurtigere, koncentrerer mig bedre, og vigtigst: jeg er i bedre humør når min familie kommer hjem. En bonus er at jeg åbenbart automatisk rydder lidt op omkring mig i de 5minutters pauser der er, så min arbejdsstation har aldrig været mere ryddelig.

Hvis jeg skal vælge to ting jeg har lært i dette projekt, som jeg vil tage med videre til det næste er det: Flere unit tests – tidligere og sæt en meget mindre ambitiøs kriterie for working product.

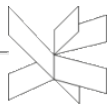


6. Refleksioner over vejledning

Vi har i gruppen generelt ikke modtaget særlig meget vejledning. Vi har haft en fornemmelse af at sidde dybt begravet i arbejdet, og har ikke haft en systematisk tilgang til kontakten af vejledere. I de tilfælde vi har haft, har det opstået nærmest spontant fordi vi lige faldt i snak om det.

Dette er dog med en enkelt undtagelse – Pelle. Eftersom Pelle er vores vejleder med mest erfaring i Unity, har det været ham vi har kontaktet når ingen af os har kunnet finde en løsning. Han har været enorm hjælpsom, og haft en karakter mere som support-line end vejleder. Når vi har kontaktet ham over discord har han været altid været parat til at give gode råd. Fx da vi havde problemer med assemblies, her havde vi i en hel dag forsøgt alt hvad vi kunne for at løse et namespace problem, og Pelles hjælp og indsigt var uundværlig.

Når det kommer til spørgsmål i forhold til metode - SWE, har vi ikke været specielt opsøgende. Vi har primært benyttet klasse diagrammet til at arbejde ud fra, og derfor ikke haft reelle spørgsmål til de andre UML diagrammer. Indtil få dage inden aflevering. Vi må indse at vi står i en situation hvor det vil have været fordelagtigt at have begyndt diagrammerne tidligere, for at kunne have noget generel vejledning i forhold til projektet arkitektur og dokumentation.



7. Konklusion

Overordnet set har samarbejdet foregået uden stor konflikt samt tilbageslag, hvilket har resulteret i et produkt som tillader brugeren at finde kronjuveler tilfældigt omkring samt kongekroner ved forudbestemte lokationer og derved få gevinster i form af guld som kan bruges til digitale-kosmetiske gevinster i form af ansigtsfiltre.

Brugen af scrum samt arbejdet ud fra arkitektoniske principper såsom SOLID, har ført til en vel planlagt og rutineret arbejdsprocess og et resulterende produkt som er veldokumenteret og let at redigere. Yderligere, brugen af design-mønstre såsom Facade og Strategy har henholdsvis gjordt fejlfinding og ændring af eksisterende systemer, såsom kompasset, lettere men også gjordt tilføjelse af yderligere indhold, her i form af minigames, til en streamlinet oplevelse for udviklere. Desuden har scrum givet værktøjer til håndtering af tilbageslag samt planlægning af resten af projekt perioden, hvilket blev yderst brugbart ved de førnævnte kompas-komplikationer hvor det hurtigt blev nødvendigt at terminere og planlægge en ny sprint samt at holde overblik over mængden af arbejde lavet og manglet.

Fejlhåndtering af projektet har været tilnærmelsesvis simpelt grundet de løbende unit- og integrationstests som blev opsat. Dette var en vigtig del af arbejdsprocessen som tillod at teste alle nye funktioner grundigt, specielt funktioner som gjorde brug af førhen-skrevent kode, inden funktionerne blev pushet til et højere niveau af branch i Repositoriet. Udover dette gav inklusionen af en Validation branch mulighed for at dele ansvar for kode blandt udviklere i gruppen og derfor sikre at alt endeligt i projektet mindst har haft to sæt øjne på sig.

Det endelige produkt har stadig nogle mangler, såsom en implementeret løsning til den konstante initialisering af kroner. Her drages der igen fordel af brugen af design-mønstre hvor det brugte Factory-mønster nemt kan udvides til at inkludere et object pool som kunne byde en løsning. Factory mønsteret er blandt de andre mønstre og brugen af SOLID med til at skabe et endeligt produkt som er designet ud fra altid at kunne udvides yderligere på funktionalitet.

8. Referencer

bevaegdigforlivet
2025.
Bevæg dig for livet
Bevæg dig for livet.
https://www.bevaegdigforlivet.dk/

GDPR.DK ApS
2025.
Artikel 6 – Lovlig behandling - GDPR.DK.
GDPR.DK ApS.
https://gdpr.dk/databeskyttelsesforordningen/kapitel-2-principper/artikel-6-lovlig-behandling/
z?gad_source=1&gad_campaignid=
20728534545&gbraid=0AAAAAC7tfkPWRYLVdAJSiWTfJJYoqRPjE&gclid=
EAIaIQobChMI07CppZDDjQMVPASiAx19ewCJEAAYASAAEgKXXvD_BwE

Digitaliseringsstyrelsen
2025.
Huskeregler for app-udbydere
Huskeregler for app-udbydere
https://digst.dk/tilsyn/sporingsteknologiomraadet/vejledninger-til-tjenesteudbydere/huskeregler-for-app-udbydere/

Unity
2025.
Scripting API: LocationService
Scripting API: LocationService
https://docs.unity3d.com/6000.1/Documentation/ScriptReference/LocationService.html

GeeksforGeeks
2025.
Haversine formula to find distance between two points on a sphere
Haversine formula to find distance between two points on a sphere
https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

Unity
2025.
Body Tracking
Unity MARS
https://docs.unity3d.com/Packages/com.unity.mars@1.3/manual/BodyTracking.html

Unity
2021.
Create a basic face filter
Unity MARS
https://learn.unity.com/pathway/mobile-ar-development/unit/create-a-basic-face-filter-tutorials?version=2021.3

Bleything, T.
2023.
How to Convert Magnetometer Data into Compass Heading
Digilent Blog
https://digilent.com/blog/how-to-convert-magnetometer-data-into-compass-heading/