**VIA University College**

**Spil2-Rapport**

**Nikolaj Bræmer Christen, 354322**

**Patick Overgaard Blauert, 353829**

**Peter Hougaard, 353296**

**Victor Bruun Fassbender, 354361**

**Vejledere:**

**Henrik Kronborg Pedersen**

**Søren Klit Lambæk**

**Pelle Sølvkjær Christensen**

**16.216 tegn**

**Software ingeniør**

**2. semester**

**28-05-2025**

**Indhold**

Contents

[Introduktion 1](#_Toc199143534)

[Teknisk arkitektur 1](#_Toc199143535)

[Anvendelse af AR-teknologi 4](#_Toc199143536)

[Sensorer og hardware 5](#_Toc199143537)

[Input og interaktion 5](#_Toc199143538)

[Optimering og performance 6](#_Toc199143539)

[Konklusion og refleksion 7](#_Toc199143540)

# Introduktion

Royal GO, er et real world gps baseret spil, der har til formål at fremme fysisk aktivitet blandt børn og unge i Danmark ved at kombinere jagten på kongekroner og juveler med augmented reality (AR) teknologi. Spillere motiveres til at bevæge sig mere ved at tilbyde en engagerende og interaktiv oplevelse, hvor de kan samle virtuelle skatte ved specielle forudbestemte seværdigheder.

Målgruppen for dette spil er primært børn og unge, i forbindelse med Royal Run arrangementet. Eftersom børn ofte oplever verden gennem deres mobiltelefoner, og har en tendens til at være fysisk inaktive (Move for Life, u.d.), forsøger vores spil at engagere børn til at motionere mere, ved at tale til dem på en måde de forstår, og gennem et medie de er bekendt med. Begivenheden Royal Run er dog primært et sted hvor voksne mødes omkring det at dyrke motion, men har taget deres modvillige børn med. Vores spil, Royal Go, forsøger derfor at skabe et segment der også kan tiltale disse børn.

For at gøre oplevelsen mere lokalt forankret, bruger vi AR-teknologi til at skabe en hybrid oplevelse, hvor spilleren kan interagere med både fysiske og virtuelle verden samtidig. Helt konkret skal spilleren afsøge den virkelige fysiske verden, i et prædefineret område bestående af lokale seværdigheder, for at finde digitale skatte.

# Teknisk arkitektur

Vores tekniske arkitektur har taget udgangspunkt i vores domænemodel (se Bilag 1: Domæne model). Selvom der er mange funktioner, er mange af dem supporterende i forhold til spillet. Butikken har fx en meget mere simpel opbygning end selve spillet. Derfor er de to primære arbejdsheste i vores arkitektur: Navigationssystemet, og AR-systemet.

Navigationssystemet er ansvarligt for at opdatere alle relevante oplysninger i forhold til den spillerens- og seværdigheders orientering i den fysiske verden, og samle dette til en pakke af relevant information. AR-systemet er derimod ansvarligt for at håndtere den digitale verden der opleves gennem kameraet. Vi har forsøgt at holde brugen af Monobehaviours til et minimum, da de gør det besværligt at Unit teste vores scripts. Derfor vil man kunne læse på vores overordnede klassediagram (se Bilag 2: Overordnet klassediagram) at langt størstedelen af vores model er ren c# klasser. Hvor vidt dette påvirker vores performance i et lille spil som vores er ikke relevant, det har dog gjort det væsentligt nemmere at benytte UML værktøjer.

Et billede, der indeholder tekst, diagram, Plan, Parallel

Indhold genereret af kunstig intelligens kan være forkert.

Figur 1 Navigations system I klassediagram

#### Navigations system:

Navigationssystemet er designet ud fra to mønstre: Facade- og observer mønstrene. Eftersom Navigation systemet bruger flere forskellige kilder, og benytter flere klasser til at udføre udregninger har vi valgt at bruge facade mønstret. Her kan vi forbinde mange forskellige klasser under en enkelt klasse, der så repræsenterer en simpel adgang til alt relevant data i form af klassen: NavigationSystemData. Her bliver de mange inputs og bereginger kogt ned til en håndfuld nødvendige data: position, retning, samt retning og længde til mål.

En faldgrube vi skal være opmærksom på når vi benytter facade-mønsteret, er at vi ikke skaber en gudeklasse. Altså en klasse hvor alting sker i fordi det lige var nemmere at hard-kode en at lave en ordenlig struktur. Derfor har vi været meget opmærksome på at minimere ansvaret for navigation klassen til netop kun at behandle og præsentere geografisk information, og så have andre klasser til at indsamle, og beregne på denne data.

Efter alt navigations dataen er parat bruger vi observer mønstret til at distribuere det ud i systemet. Det gør vi fordi navigations systemet ligger langt nede i afhængigheds-hierarkiet i arkitekturen, og ikke behøver nogle inputs fra brugeren. Alt info den har brug for, får den gennem telefonens sensorer, og kan derfor nøjes med en én-vejs kommunikation. Derudover, benyttes dataen på alle niveauer af afhængigheds- hierarkiet. Ved at bruge en observer kan Navigation system dataen på simpelt vis distribueres til fx MVP mønstret.

#### AR-systemet

AR-Systemet bliver primært håndteret af Unity, og at interagere med det er ikke så anderledes fra hvordan man normalt vil interagere med en scene i Unity Den største forskel er at når systemet skal testes bliver vi nødt til at sætte et simuleret test-miljø op. Måden vi har interageret med AR-systemet er primært ved at instantiere objekter ind i den virtuelle verden, når en spiller har befundet sig på det rigtige punkt samt kigger i den rigtige retning i virkeligheden.

Spawn element in AR world er den klasse vi benytter til at spawne objekter i den digitale AR scene. Den benytter mønstrene singleton og factory til at gøre den nemt tilgængelig og skalerbar. Det betyder at alle ting der skal spawne i AR scenen, bliver nød til at have både en konkret factory og produkt, hvilket øger kompleksiteten. I vores tilfælde har vi kun 2 forskellige elementer der behøver at spawne: Kronjuveler, og kongekroner. Disse to elementer har vidt forskellig interaktions karakter og spawn kritererier, det giver derfor mening at have to seperate factories der kan sætte dem op. Et billede, der indeholder diagram, Plan, Teknisk tegning, Rektangel

Indhold genereret af kunstig intelligens kan være forkert.

Figur 2 Ar World Object Factory pattern I Klassediagram

Til at arbejde sammen med element spawneren, er der en controller. Denne controller består igen af andre controllere, som styrer de henholdsvis kronjuvelers spawnkriterier samt kongekroners. Selvom det virker som mange klasser, har det vist sig at være meget brugbart at have klasser der blot samler inputs fra andre, og så tager Et billede, der indeholder diagram, Plan, tekst, Teknisk tegning

Indhold genereret af kunstig intelligens kan være forkert.stilling til hvordan de præcist skal udføres.

Figur 3 Minigame Strategy pattern I Klassediagram

Til udarbejdelsen af vores minispil systemet er der brugt et strategy-pattern, som tillader at vi let og overskueligt kan tilføje flere spil ind i systemet, uden at det vil bringe mere kodning ind andre steder i systemet, her vil et nyt spil blot skulle nedarve fra vores abstrakte klasse, og så er der lavet en metode for at gå igennem alle nedarvningerne og starte et enkelt af dem, dette gør spillets struktur mere solid, specielt med henblik på open Close princippet.

#### Solid

I vores arkitektur design har vi brugt lang tid på at diskutere hvordan vi bedst kan strukturere det i forhold til SOLID principperne. I disse diskutioner har vi i første omgang lagt størst vægt på at udforske hvordan eksisterende mønstre kan bruges til at danne fundament for vores system. Til at supportere arkiteturen omkring disse mønstre, lagde vi i første omgang stor vægt på S – Single responsibility.

Måden vi har forsøgt at benytte single responsibility kan fx ses i controllersystemet til element spawneren. Her er der en hoved controllerens ansvar at controllere det der skal spawne AR scenen. For at kunne udføre dette ansvar er den afhængig af andre controllere som kontrollerer hvornår del-elementerne skal aktivere. Disse ansvar kunne nemt have været lagt i en enkelt klasse, men vi har delt dem op i underansvar, netop for at overholde SOLID principperne.

Vores primære strategi til at overholde Open closed princippet har ikke været en stor bekymring for os, da mange af vores funktioner er meget forskellige, og vi generelt ikke har benyttet særligt meget nedarvning. Når det har været relevant at nedarve, fx da vi implementerede factory mønsteret, har vi brugt en abstrakt klasse, og derved tilføjet den ekstra funktionalitet i det konkrete script. På samme måde har vores sparsomme brug af nedarvning gjort det nemt at overholde Liskovs princippet.

Interface Segregation princippet har vi benyttet langt de fleste gange hvor kontakt til andre klasser har været nødvendig. Fx kom det os til gode da vi fandt ud af at kompas-sensoren ikke fungerede i Unity 6. Her var implementeringen af et helt nyt kompas system enorm nem, da det blot var at tilføje et interface.

Dependency inversion princippet fik vi opfyldt tidligt ved at benytte observer mønstret til vores høj-afhængigheds del af vores system: Navigations systemet. Ellers har vi generelt forsøgt at forbinde høj afhængigheds systemer med singletons og interfaces.

# Anvendelse af AR-teknologi

Som tidligere nævnt er AR systemet en central del af vores spil. I dette afsnit vil vi gå en smule mere i dybden med hvordan vi rent teknisk benytter det og hvilke kriterier der skal være opfyldt for at kunne bruge det.

Vores spil benytter plane-detection til at referere til den virkelige verden. Dette gør vi da vores spil som udgangspunkt er designet til at finde sted i en by, hvor plane flader er rigelige. Derudover, vil andre detectionsmetoder som billedsporing ikke give mening for os, da vi er afhængige af gps lokationer, hvilket gør det overflødigt også at have fysiske billeder hængt op for spilleren.

Når tiden er til at instantiere et objekt i spil scenen, bruger vi en Coroutine til at tjekke for planer, og derefter instantiere hvis der findes en. Denne metode fortsætter til der er fundet en egnet flade at spawne på, hvorefter der tages udgangspunkt i spillerens position i den digitale verden til at placere det instantierede objekt.

For at kunne interagere med spawnede objekter har vi oprettet klassen: ARPressDetector, der caster en ray fra spillerens skærm, og vinkelret ind i den digitale verden. Herefter, itererer den gennem alle ramte objekter og søger efter om der er et relevant tag. Hvis der bliver fundet et match, kalder den et handler interface, hvor resten af interaktionen processeres i en separat OnPress() funktion.

# Sensorer og hardware

Royal Go er baseret på gps lokationer og retning, og er derfor afhængig af at den enhed der spilles på har adgang til både gps og kompas sensorer. For at kunne tilgå disse inputs i Unity er det dog nødvendigt først at konfigurerer systemet til at kunne tilgå dem. Dette gøres gennem android-manifestet, hvor få linjer kode skal tilføjes for at aktivere tilladelserne. Af de to sensorer har gps-lokationer været den nemmeste at benytte, da den gav længde og bredde grader direkte, mens dataen fra magnetic field sensoren var omstændig at convertere til et brugbart format.

#### Kompas

Eftersom Unity 6 ikke supporterer det gamle kompas, men nu har en magnetisk felt sensor, er det nødvendigt at lave et kompas fra bunden af, fra en Vector3. Den magnetisk felt sensor returnerer tre akse, nord/syd, øst/vest og op/ned. For at kunne omsætte dette til et brugbart resultat benyttede vi atan2 til at få relationen mellem disse i radianer, som vi så omregnede til grader (Bleything, 2023).

En vigtig ting at i forhold til geologiske positioner er at de indeholder enormt store tal. Derfor er det nødvendigt at benytte doubles i stedet for floats når den relative vinkel mellem en spillers position og en anden location skal beregnes.

#### Lokation

Til at kunne guide spilleren rundt er der også gjort brug af GPS signalet fra telefonen, dette hentes ned fra input systemet, som giver os spillerens nuværende Latitude og Longitude position, disse ”Lat/Long” koordinater, bliver så sammenlignet ved hjælp af haversine formlen (geeksforgeeks, 2022), som udregner afstanden mellem to punkter på en kugle, hvor vi har sammenlignede med nogen float værdigere, som er skrevet ind som de ”lat/long” for det lokationer som spilleren skal være, dette har resulteret i at vi kan sende spilleren rundt i verden til forskellige steder vi ønsker dem at se.

#### Manifestet

Eftersom at vores spil, køre på android telefoner, og vi gør brug af forskellige sensorere i telefon, har det været nødvendigt at kigge ind i Android Manifestet, for at håndtere at gøre disse ting på en lovlig måde, manifestet fungere som en slags konfigurationsfil for android om hvordan apps skal opføre sig, derfor har vi specifikt været inde og arbejde med den del af manifestet som specifikt håndterer tilladelser, hvor vi har tilføjet en ”user permission” til brugen af kamera og location data, dette er gjordet i henhold til brugen af Persondataloven omkring GPS sporing (juralia, 2025)

## Input og interaktion

I vores spil bruger vi udelukkende raycasts til at vurdere om et AR element er trykket på eller ej. Hvis vi skulle implementere flere interaktioner gennem AR vil et oplagt sted at gøre det vare i vores minigames. Som det er lige nu, er alle minigames UI baseret og bliver implementeret gennem strategy mønsteret. At lave et AR spil hvor man fx skal rotere en nøgle til at passe til en hængelås der vogter en kongekrone kunne være en interessant mulighed, både teknisk og fortællingsmæssigt. Vi har arkitekturen på plads til nem at kunne implementere, eller udvide det eksisterende minigame system.

## Optimering og performance

Efter at have testet vores spil i profileren, har vir konkluderet at det sagtens vil kunne køre med 60fps. Selv efter at have provokeret det mest ressource intense modul – AR modulet, ville vi stadig kunne køre med 60fps.

Ud fra et screenshot fra profileren (se Figure 1) er det tydeligt at se at spillet langt det meste af tiden er godt under 60fps. Det bliver også tydeligt at 59.6% af systemets ressources bliver brugt på at kører editoren, altså processer der ikke vil være til stede i det endelige build. Det er også dette editorLoop der er skyld i de store spikes.

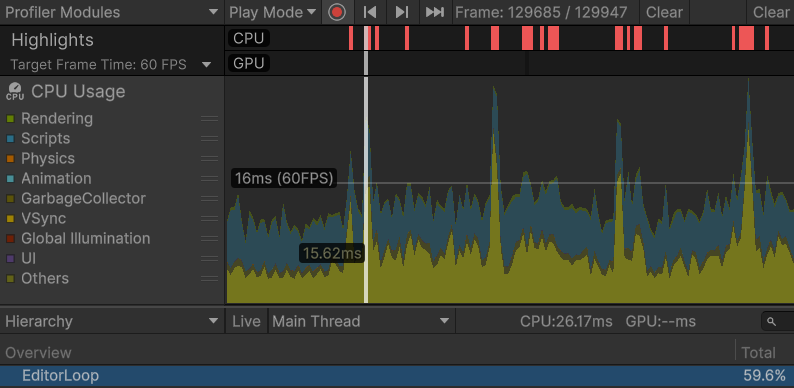


Figure 4: Screenshot fra profiler under den mest krævende del af spillet.

Sorterer vi EditorLoop fra i vores profiler får vi altså et resultat der viser at vores spil vil kunne køre med en hastighed på omkring 100fps.

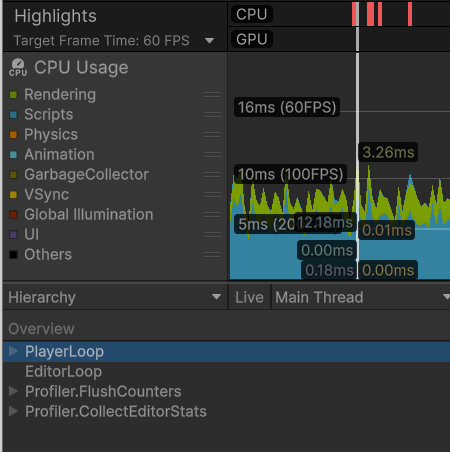


Figure 5: Profiler uden editorLoop

Det vil dog være helt unødvendigt at have spillet til at opdatere så hurtigt. For det første er der tale om et håndholdt spil man sikkert kommer til at spille på farten. I det tilfælde kan man argumentere for at man ville kunne slippe afsted med en væsentlig lavere refresh-rate. For det andet bliver Navigations systemet opdateret med 300ms intervaller, så på 80% af de viste frames vil der ikke blive beregnet relevant ny gameplay information.

Ud fra ovenstående observationer har det været svært at retfærdiggører hvorfor der skulle prioriteres mandetimer til optimering. Optimering er dog holdt i baghovedet når det kommer til de løsninger vi har implementeret. Fx forsøger vi at holde instantieringer til at ske i opstart fasen ved at object pools, fx bruger SkatteEventFactorien en object pool til dens (ene) produkt. Og vi har prioriteret at bruge mindre datatyper hvor vi kunne, således at det kun er kompasset der bruger double baseret matematik. Men disse tiltag har så lille en påvirkning at det ikke har været muligt at skimte dem i profileren.

## Konklusion og refleksion

Vi har opnået en væsentlig teknisk indsigt, især i anvendelsen af AR-teknologi, GPS- og kompasintegration. Vi har fået en dybere forståelse for vigtigheden af en robust arkitektur baseret på SOLID principperne. Efter at have arbejdet bevist med arkitekturen af systemet har det været meget mere overskueligt at implementere nye funktioner og fejlsøge kode. Udfordringer med Unity 6’s Kompas sensor og håndtering af store geologiske data har tvunget os tilpasse og bygge skræddersyede løsninger, erfaringer vi garanteret vil gavne af i fremtiden

Hvis projektet skulle startes forfra, ville vi prioritere en endnu tidligere implementering af et simuleret testmiljø for AR-systemet, og flere – og grundigere – unit tests. Vi ville også overveje at forbedre arkiteturen på vores supporterende klasser, fx world controlleren og dens under controllere må kunne designes bedre / mindre komplekst. Disse ændringer ville forbedre både udviklingsprocessen den tekniske arkitektur bevarer skalerbart og effektivt.

**Bilag**

1. A diagram of a person's name

   AI-generated content may be incorrect.Andre diagrammer

Bilag 1: Domæne model

1. Klassediagrammer.

Bilag 2: Overordnet klassediagram



# Referencer

Bleything, T. (2023). *digilent*. Hentet 22. 05 2025 fra https://digilent.com/blog/how-to-convert-magnetometer-data-into-compass-heading/

geeksforgeeks. (2022). *geeksforgeeks*. Hentet 28. april 2025 fra https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

juralia. (2025). *juralia*. Hentet 26. 05 2026 fra https://juralia.info/regler-for-gps-sporing/

Move for Life. (u.d.). *royalrun*. Hentet 25. 05 2025 fra https://www.royalrun.dk/om-royal-run/about-royal-run/