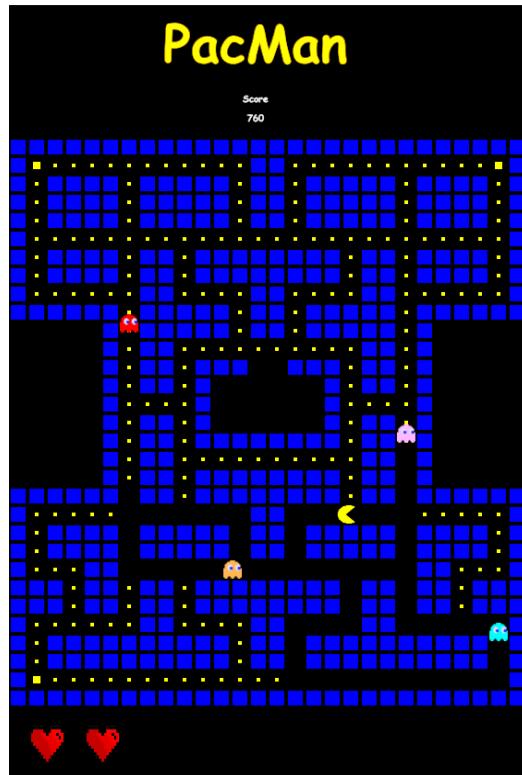


DM575 - Objektorienteret programmering

May 30, 2025



Mathias B. Jensen
Nikolaj R. Madsen
Valdemar B. Reib

Institut for Matematik og Datalogi

Syddansk Universitet

Indhold

1	Introduktion	3
2	Design	3
3	Implementation	5
3.1	Entity stamtræ	5
3.2	AI (Spøgelser)	6
3.3	Interfaces	7
3.3.1	Kollisioner	7
3.3.2	Rutesøgning	9
3.4	Controller	9
3.5	Grafiske Brugergrænseflade	9
4	Kvalitetssikring	11
5	Proces	12
6	Diskussion	13
7	Litteraturliste	17

8 Bilag	18
8.1 Første UML-diagram	18
8.2 Endelige UML-diagram	19

1 Introduktion

I dette projekt har opgaven bestået i at designe og kreere et simpelt spil, som er inspireret af det klassiske og velkendte arkadespil Pac-Man. Ved at benytte Java og JavaFX som værktøjer, har gruppen været i stand til at benytte essentielle designprincipper for objektorienteret programmering til at kreere spillet. I denne rapport vil der blive gennemgået design processen samt udviklingen af det faktiske produkt.

2 Design

Allerede som det første i projektet, diskuterede gruppen om hvilke designprincipper, -strategier og mønstre der skulle tages i brug i projektet. Der er mange at vælge mellem og alle kan have væsentlig betydning både for arbejdsprocessen, men også den endelige kildekode. Derfor var det imperativt for gruppen at evaluere, hvilke var til fordel for processen og hvilke der ikke havde nogen positiv indvirkning, når gruppens fremgangsmåde tages i mente. For at kunne evaluere designet, valgte gruppen at modellere kildekodens nuværende og fremtidige bestanddele i adskillige UML-diagrammer. Mængden af diagrammer stammer fra forskellige tidspunkter fra forløbet og afspejler gruppens iterative arbejdsproces. Det første UML-diagram kan ses på bilag 6, hvor opdelingen af ansvarsområder mellem klasserne allerede kan ses. På det første diagram havde gruppen allerede konkretiseret idéen om en centraliseret overklasse, ved navn **Spillet**, som holder overblik over Pac-Man spillet, blandt andet ved at tælle scoren og antallet af liv. Udover det er der et hierarkisk struktur mellem klasserne. **Spillet**-klassen består og afhænger af underklassen **Board**, som selv består af underklasserne **Tile** og **BoardLayout**. I helhed programmeres store komponenter ved at sammensætte og kombinere mindre komponenter. Der er få sammenhænge, hvor strukturen er nødt til at være delvist cirkulæret. Spillet skal være interaktivt og dynamisk, som resultere i at de fleste klasser skal reagere enten direkte eller indirekte på input fra brugerne. Pac-Mans bevægelse skal direkte reagere på brugerens tastetryk, mens scoren forhøjes i takt med at Pac-Man indsamler piller, som en indirekte konsekvens af tastetrykkene. Mange af elementerne fra det op-

rindelige UML-diagram beholdes i de senere iterationer og endda i det sidste. Det sidste UML-diagram som afbilde den endelige kildekode kan ses på bilag 7. Ligesom det første, har det endelige diagram en centraliseret klasse, der ikke længere hedder **Spillet** men **GameManager** i stedet, dog har de den samme rolle. Hovedforskellen mellem de første og sidste diagram er simpelthen omfanget. Det sidste UML-diagrammet indeholder naturligvis flere klasser, da gruppen ikke kunne forudse den endelige struktur af kildekoden. Mange af klasserne blev først konkretiseret senere og er blevet trinvis integreret i det samlet design. For eksempel konstrueres store klasser, for eksempel **GhostController** stadig af mindre klasser, der hjælper med at implementere den store klasses funktionaliteter og skabe overblik. De fleste klasser genbruges i flere stedet i det samlet design. For eksempel afhænger mange komponenter af **PacMan**-klassen, da den er roden og omdrejningspunktet for meget af spillets dynamik.

En af de mest væsentlige designprincipper som gruppen har gjort brug af er SOLID. SOLID er et objektorienteret udviklingsprincip der favner en masse principper, her under (Single responsibility, Open-closed, Liskovs substitution, Interface segregation og Dependency inversion). Der har været stor fokus på SRP (Single Responsibility Principle) som sikre at en enkelt klasse ikke har ansvar for mere end den selv. Dette gør programmet ekstremt modulært og tillader udskiftning af implementering igennem hele programstrukturen. Gruppen har også gjort brug af OCP (Open-Closed Principle), dette er kommet til udtryk ved brugen af private og beskyttet variable og klasser, som sikrer at det kun er klassen der skal benytte dem der kender til dem og kan rette i dem. Det minimerer risikoen for uventede interaktioner mellem klasserne i programmet, og har gjort fejlsøgning væsentligt nemmere.

Det designmønstre som gruppen i meget høj grad gjorde brug af er MVC (Model View Controller). Dette er et designmønster der opdeler ansvarsområder i et program. Model varetager datastrukturen, view er den del der varetager GUI delen af programmet og Controller håndterer logikken til programmet. Disse tre ting interagerer med hinanden i en cyklistisk formation. Dette er tydeligt at se i den måde programarkitekturen er kreebet på. Det er opdelt i view klasser, controller klasser og model klasser. Et eksempel er den måde **GUI** klassen står for at danne den faktiske grafiske del, hvorimod **BoardLayout** håndtere datastrukturen der danner grundlag for at **MazeView** klassen kan tegne et board. Hertil er det en **Board** klassen der står for Controller delen,

ved at varetage logikken bag boardet. Selve logikken bag at drive spillet med start, slut, liv og score sammes i `GameManager` klassen, som er den der binder det hele sammen, ved at kreere de nødvendige objekter, og benytte de forskellige objekters controller til at ændre spillets stadie. Alt dette arbejder sammen i en cyklisk formation, hvor ansvarsområder er opdelt, men med en hoved klasse til at binde det sammen.

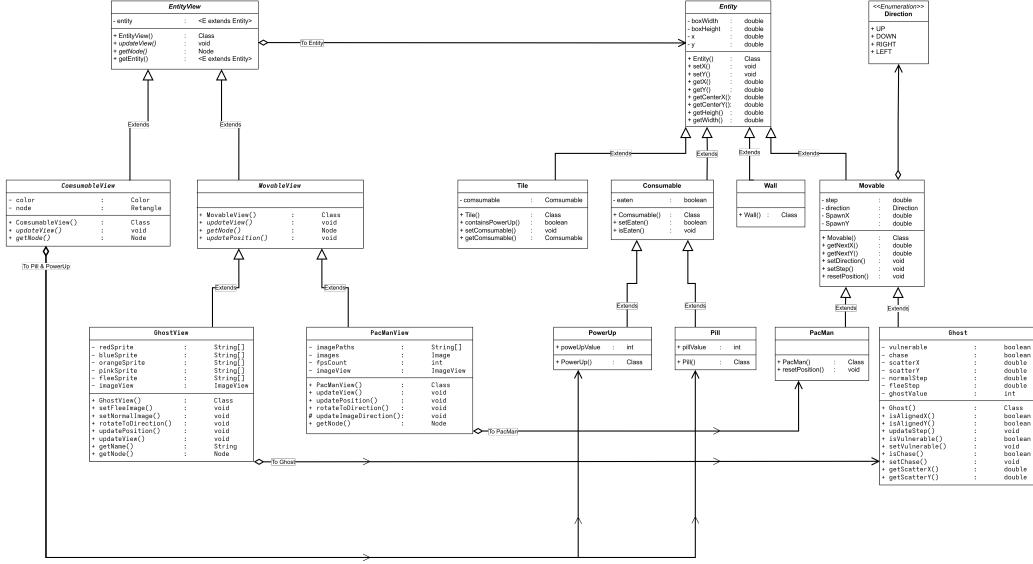
I udviklingen af projektet har gruppen benyttet sig af projektbeskrivelsen for at sikre sig at minimumskravene opfyldes først. Ved at sikre dette, tillod det gruppen at tilføje animationer og AI til spillet. I det originale Pac-Man er der flere komponenter som ikke står i kravspecifikationen, men som gruppen har tilføjet. Dertil er der også andre aspekter som kunne adderes, eller ændres. Dette snakkes der mere om i diskussions sektionen af denne rapport.

3 Implementation

Gruppen har benyttet sig af OpenJDK 24, som er den nyeste stabile version. OpenJDK er et open-source projekt der tillader gruppen at udvikle spillet, med en transparent version af JRE (Java Runtime Environment). For at kompilere koden og kører spillet, skal man benytte Maven til at kompilere koden og som minimum have JDK 11 installeret, ved at placere sig i rod mappen, som indeholder en `pom.xml` fil, samt `src` og `target` mapperne, kan man kører kommandoen i terminalen `mvn clean javafx:run`. Denne kommando vil læse xml filen og hente de nødvendige JavaFX afhængigheder for at kører programmet.

3.1 Entity stamtræ

I ovenstående diagram ses en oversigt over programmets `Entity` struktur. Det bygger på en abstrakt klasse `Entity` som beskriver en skabelon til hvad en `Entity` er. Dette er ens for den abstrakte klasse `EntityView` som definere den visuelle del af en `Entity`. Ved at benytte en abstrakt klasse struktur kan der defineres abstrakte metoder som underklasser skal implementere når



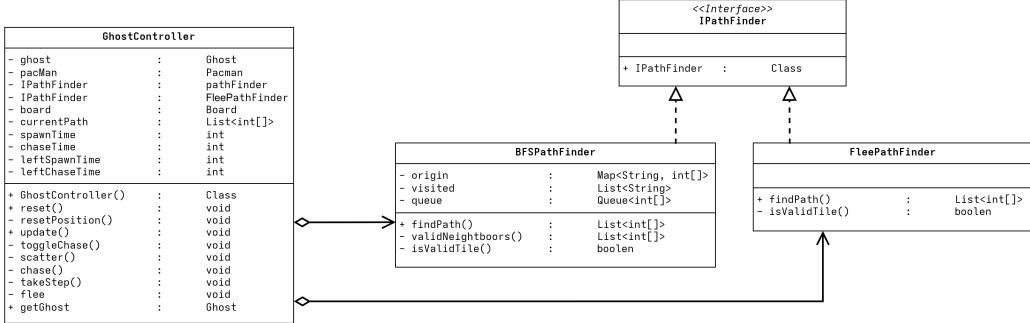
Figur 1: Oversigt over Entity stamtræ

de udvider den abstrakte klasse. Ved at bruge abstrakte klasser fremfor interfaces, kan der benyttes en **constructor** som tillader en klasse at kreere en instans af f.eks. **EntityView**. Designet i sig selv tillader nedarvning af de nødvendige parameter til et visuelt eller fysisk objekt er ens. Derved overholderes DRY (Don't Repeat Yourself) princippet, ved at mindske unødvendige gentagelser.

Denne struktur viser også tydeligt programmets brug af MVC principippet. Ved at opdele ansvarsområderne og benytte abstrakte klasser til at sikre nedarvning. Dette gør også at LSP (Liskov Substitution Principle) overholderes, i det at implementeringen af underklasserne kan udskiftes modulært. Derudover benytter gruppen få gange **Override** flaget, til at overskrive metodernes funktionalitet.

3.2 AI (Spøgelser)

Gruppen valgte at implementere en AI til spøgelserne, for at sikre forskellig adfærd. Den overordnet struktur ses på nedenstående diagram.



Figur 2: Oversigt over AI opbygning

Spøgelser i spillet har forskellige adfærd alt efter tilstanden af spillet. I normal tilstand skifter de mellem at sprede sig ud og har et designet udspredningspunkt de skal nå før de begynder at jage PacMan. Hvor lang tid de bruger på at jage PacMan er tidsbaseret før de igen spreder sig ud. Ruten til PacMan og udspredningspunktet laves af en Bredde-Først-Søgning (BFS) og genererer den korteste vej til begge punkter. Når et spøgelse er sårbart vil dens adfærd ændre sig til at flygte. Ruten bliver her lavet som en grådig søgning der vælger det gyldige felt, der er længst væk fra PacMan.

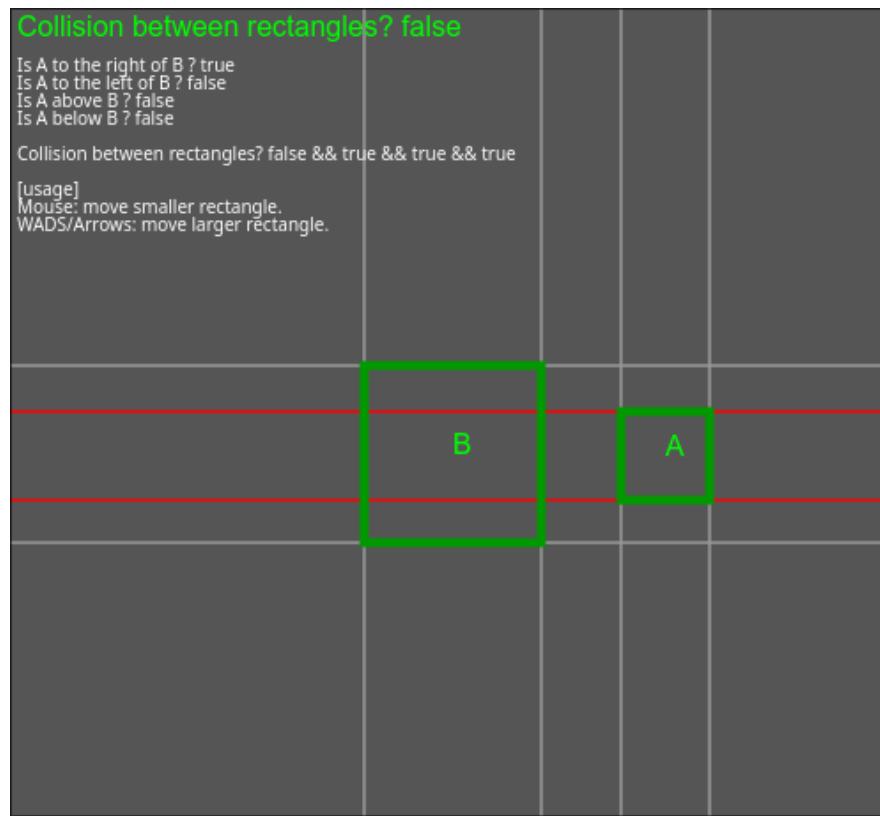
3.3 Interfaces

3.3.1 Kollisioner

For at tjekke kollisioner er der lavet et `interface`. Dette er gjort så man kan lave konkrete implementationer for forskellige `entities`. Nogle implementeringer modtager nemlig flere `entities` adgangen. Når man tjekker for kollisioner bruger man metoden `hasCollision` med en `entity` som argument og den `entity` sammenlignes med alle `entities` i kollisionstjekkeren. `wallCollisionChecker` er en konkret implementation og bruges for at vurdere om PacMan kan bevæge sig ind i et nyt felt uden at kolidere med en væg.

For at bedømme om et objekt har kollideret med et andet objekt som f.eks.

PacMan med en pille. Har gruppen gjort brug af AABB (Axis-Aligned Bounding Box) kollision. Dette en kollisions metode der baseres på at lave en hitbox som er en firkant rundt om objektet, hvor der er streger der bevæger sig længere end objektets firkant. Disse streger tegnes for hver kant på den firkantet boks, der omfavner objektet. Ved at tjekke overlap mellem disse linjer, kan man på en beregningsmæssigt let måde bestemme ved hjælp af nogle booleske udtryk om der er kollision mellem objekterne [Kis15]. En visuel repræsentation kan findes nedenstående.



Figur 3: Axis-Aligned Bounding Box (AABB) repræsentation.

3.3.2 Rutesøgning

Der er lavet et **interface** til rutesøgning. Spøgelserne bruger f.eks. en konkret implementation af rutesøgning til at jage PacMan. Man kan derfor implementerer nye metoder til at justere adfærd under jagning.

3.4 Controller

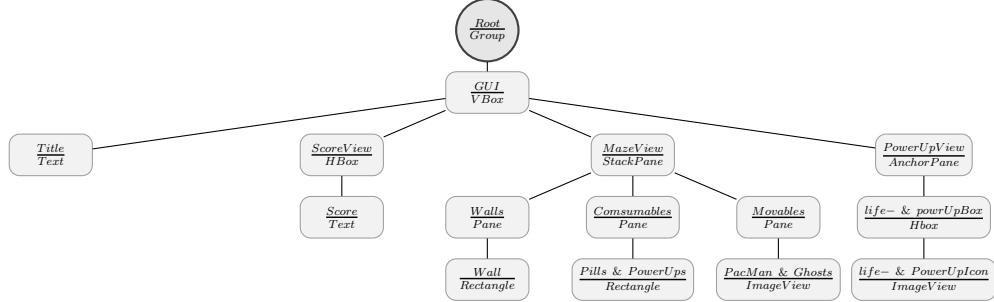
Der er lavet **controllers** for alle **entities** der ændrer adfærd under spillet. Piller og Power-Ups kan blive spist. Spøgelser kan jage, sprede sig ud og flygte. PacMan kan flytte sig og skifte retning baseret på spillerens input. Under et **updateTick** i **GameManager** vil deres adfærd blive udført.

3.5 Grafiske Brugergrænseflade

Til projektet benyttede gruppen Java biblioteket JavaFX til at programmere den grafiske brugergrænseflade [Ope25b]. I kildekoden har JavaFX til ansvar at fremvise og præsentere selve computerspillet. Biblioteket muliggøre at gruppen i højere grad kan fokusere på kodearkitekturen, så det er i overensstemmelse med OOP-principperne, og ikke funktionaliteter der er irrelevante for projektet. f.eks. understøtter JavaFX registrering af tastetryk og andre inputmetoder, så gruppen ikke behøver at implementere det fra bunden af.

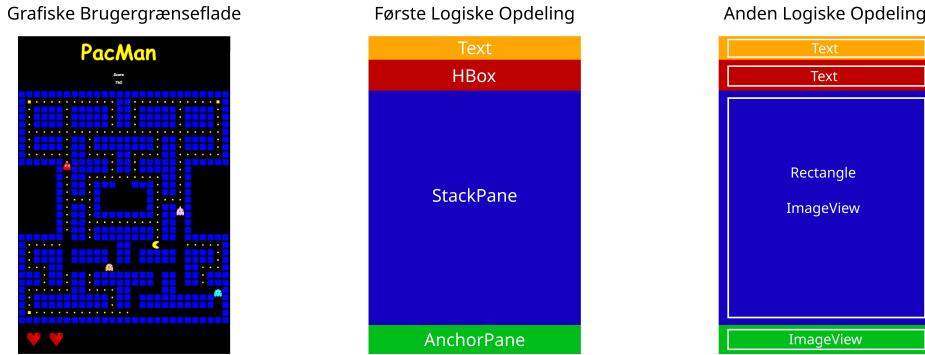
I selve implementeringen benyttede gruppen i meget høj grad den abstrakte klasse **Node** fra JavaFX [Ora25] og dens underklasser. **Node** overklassen tillader gruppen at gruppere og organisere de grafiske komponenter i en træliggende struktur med rødder og blade. Træstrukturen hjælper ikke kun med organiseringen, men komplementerer den objektorienteret arkitektur. Knuderne i træet implementeres som objekter, hvor deres interne funktionaliteter indkapsles og gøres delvis uafhængige af knudens rod. Da programmet er et dynamisk computerspil, kan komponenter ikke gøres komplet uafhængige, da elementer skal ændres eller opdateres på betingelse af andre. De adskillige

knuder i træstrukturen samt deres primære grafiske komponent er afbilledet på nedenstående figur.



Figur 4: Træ over klasser og grafiske elementer

Rent visuelt er brugergrænsefladen inspireret af arkadeudgaven af Pac-Man. Der findes overraskende mange forskellige udgaver af spilbanen, men gruppen tog udgangspunkt i et officielt design fra Bandai Namco [Nam25], som er indehaverne af Pac-Man IP'en. Gruppen genskabte det design og opdelte det i mindre sektioner for både at skabe overblik, men også for at beslutte, hvilke af de grafiske komponenter, som JavaFX tilbyder, der er bedst egnet til hver sektion. Beslutningen om, hvilken komponent der passede bedst til hvilken sektion, var ikke åbenlys i starten af processen. Dermed itererede gruppen over den grafiske brugergrænseflade adskillige gange for at opnå et kompromis mellem æstetik, funktionalitet og simplicitet. f.eks. benyttede gruppen i høj grad af **Canvas**-klassen, der tillader at tegne arbitrære figurer på den, til at tegne labyrinten fra spilbanen. Gruppen fik dog senere udfordringer med **Canvas**-klassen, da den ikke spillede godt sammen med **ImageView**-klassen, som blev brugt til at tegne Pac-Man og spøgelserne. Gruppen genevaluerede brugen af **Canvas** og valgte at benytte **Pane** og dens underklasser i stedet, da de blandt andet også er underklasser af **Node** ligesom **ImageView**. Den endelige sammensætning af grafiske komponenter fra JavaFX og den endelige grafiske brugergrænseflade kan ses på følgende figur.



Figur 5: Den grafiske brugergrænseflade og opdeling af komponenter

4 Kvalitetssikring

Gruppen har benyttet sig af manuelle test ved at kører spillet af flere omgange, og manuelt krydstjekke med kravspecifikationen løbende. Dertil har gruppen benyttet sig af Javas egen funktion til at printe til terminalen for at kunne sikre at logikken kørte som forventet.

Under testning blev gruppen bekendt med en fejl i AI implementationen. Da spøgelserne i spillet har en `scatter()`-funktion hvor de søger mod en specifik position inden de skifter til `chase mode`, er det vigtigt at koordinaterne til hvert spøgelses mål under `scatter()` er sat til en valid position. Hvis ikke denne `scatter` koordinat sættes rigtig, så vil spillet crasha da den kører BFS for evigt, indtil der ikke var mere RAM tilgængeligt. Under udvikling af spillet blev der arbejdet på forskellige `branches`, og inden samling af disse, stod gruppen med to adskilte, men færdige spil. Et af dem var et færdig spil med AI men ikke de rigtige grafiske elementer, og den anden af dem var den færdige grafiske del, men uden en rigtig AI. Da dette blev samlet opstod der en masse `merge` konflikter, som manuelt blev løst. Under løsning af disse konflikter blev der ved en fejl sat de givne `scatter` koordinater til $(0, 0)$ under instansieringen af hvert spøgelse. Dette blev først opdagede under manuel test af spillet, da det som før nævnt løb tør for RAM og crashede. Dette er en af de problemer der er svære at løse, da der ikke kommer en direkte fejlbeskæft som beskriver hvor fejlen er.

For at løse dette blev der indsat `print statements` flere steder i logikken af programmet, og manuelt udkommenteret instansireringen af de forskellige objekter, for at isolere problemet. Det blev reduceret til kun at kunne skyldes AI, da alt andet kørte fint. Til sidst blev fejlen opdagede og rettet, og spillet virkede som forventet.

En anden løsning til problemet ville være brugen af et `if-statement` der tjekkede om koordinatet var valid før der regnes på en rute. Ved at bruge funktionen `isValidTile` som en boolske sætning, ville fejle kunne fanges før, ved at bede `findPath` funktionen om at returnere en `IllegalArgumentException` fejl i tilfældet af at `isValidTile` er falsk når `findPath` kaldes.

En anden mulighed for tests ville være `unit testing`, hvor man mere slavisk og struktureret kan få bekræftet om logikken fungere, dog har gruppen ikke set behov for brug af `unit testing` da logikken i sig selv er utrolig simpel, og meget af projektet har omhandlet den grafiske implementering, samt de væsentlige designprincipper for objekt orienteret programmering.

5 Proces

Gruppen arbejde med en `subject-verb` analyse som det første i projektet. Ved at bruge projekt beskrivelsen kunne gruppen danne et overblik over de nødvendige klasser og metoder der som minimum skulle kreeres. Det første gruppen over i en brainstorm med første udkast af et UML-diagram, som i fællesskab blev tegnet på et whiteboard, jf. bilag 6. Efterfølgende har gruppen delt hver problematik op i mindre dele, og taklet dem en efter en. Ved brug af `Git`, kunne hvert gruppemedlem arbejde på hver sin ting uden komplikationer. Gruppen har gjort meget brug af `Git` til versionsstyring og samarbejde. Ved at benytte `GitLab` har hvert gruppemedlem kunne rette og tilføje til hver deres ansvarsområde. Dette gjorde det muligt at arbejde på samme program, uden at der skete konflikter. Det fungerer dertil også som en dokumentations form for gruppen selv. Ved at udeleligere arbejdet på denne måde, sikrede gruppen også lige deltagelse.

Gruppen gjorde brug af en blanding mellem fysiske og online møder, for at

balancere sparring med koncentreret arbejde. Ved de fysiske møder var der stor fokus på sparring mellem gruppemedlemmer til at komme med løsninger til implementationen af forskellige klasser og metoder i spillet. Ved at bruge de fysiske møder som sparring og arbejdsfordeling har gruppen kunne arbejde selvstændigt hjemmefra, men med mulighed for kontakt online igennem Discord.

Gruppen har i løbet af projektet benyttet AI som støtte til udviklingen. Ved at benytte funktionalitet som **deep research**, har gruppen kunne give hele projektet som kontekst til ChatGPT [Ope25a], ved at gøre dette, sikre gruppen at givne **prompts** til ChatGPT vil blive besvaret med respekt for konteksten. Dertil har gruppen været ekstremt kritiske overfor alle svar der er kommet, og ved hjælp af den kritiske tænkning er eventuelle problemer løst ved hjælp af AI.

En af de faldgruber der opstod i gruppearbejdet, udsprang af brugen af **Git** til versionstyring. Da **Git** tillader gruppemedlemmer at arbejde selvstændigt, skete der en opdeling af arbejdet. Dette var intentionen for gruppen, men der opstod en faldgrubbe som kom til kende i fejlen som blev opdaget og udbedret gennem manuel testing. Da opdeling af gruppen gjorde at det enkle gruppemedlem ikke havde en dybdegående forståelse for den andens arbejde, derfor blev der, som nævnt, sat et koordinatsæt til $(0, 0)$, som udmundede i mange timer med debugging, som kunne være undgået, hvis gruppen havde haft en bedre kommunikation i den sidste del af projektet.

6 Diskussion

Projektet blev udviklet ved at starte fra bunden med de basale objekter, som **board** og **entities**. Det var relativt sent at brugergrænsefladen blev arbejdet på. Hvis projektet skulle laves igen ville det have været intuitivt at starte fra begge ender for at sikre sig at objekterne havde den korrekte information til at kunne blive repræsenteret grafisk.

Den måde der tjekkes for kollisioner kunne også optimeres. I stedet for at tjekke i lineær tid gennem en liste af **entities** om der opstår kollisioner kunne

man finde de dimensionelle grænser for en `entity` og kun tjekke ved brug af en `map/dictionary` struktur de objekter der bliver overlappet i stedet. `PacMan` kan højst overlappe to felter adgangen, så man kunne nemt begrænse søgeområdet for vægge og reducere køretiden fra linær tid til konstant tid.

Der er lavet en `controller` for de fleste `entities`. Dette kunne også have været et `interface`. Så kunne man i `GameManager` have kaldt en `update` metode per `controller` og været ligeglads med hvilken `controller` det var og ladet det være helt op til `controlleren` selv at bestemme hvad og hvis der skulle opdateres.

Spøgelsernes adfærd kunne også blive gjort mere divers ved implementering af nye rutesøgningsalgoritmer. Man kunne have lavet en randomiseret Dybde-Først-Søgning (DFS), altså hvor den forsøgte retning per knude er tilfældig. Der kunne også have været udvidet på flygteruten og gjort den mere kompleks end en ets-felt bedste først søgning.

I nedenstående tabel ses forskellen mellem minimumskravene, gruppens implementering og det originale arkade spil Pac-Man. Der ses at gruppens implementering ligger imellem minimumskravene og det reelle arkade spil. Derudover var der ikke noget i projektbeskrivelsen som ikke blev nået af gruppen, da alle mindste krav er opfyldt. Hvis gruppen havde mere tid ville der være sat mere fokus på at få de sidste dele implementeret, og som ekstra, forskellige søgealgoritmer til spøgelserne og andre power ups.

Element	Kravspecifikation	Egen implementering	Pac-Man
Labyrint	✓	✓	✓
Piller	✓	✓	✓
PowerUps	✓	✓	✓
Spøgelser	✓	✓	✓
Normal tilstand	✓	✓	✓
Power tilstand	✓	✓	✓
Slut tilstand	✓	✓	✓
Score	✓	✓	✓
Liv	✓	✓	✓
AI ¹	✗	✓	✓
Animationer	✗	✓	✓
Ekstra grafik ²	✗	✓	✓
Lyd	✗	✗	✓
Bær	✗	✗	✓
HighScore	✗	✗	✓

¹ Herunder individuel opførsel for hvert spøgelse.

² Ekstra grafik skal forstås som grafikkens der får det til at ligne et arkade spil, herunder titel og pixel hjerter.

Tabel 1: Oversigt over forskellen i implementering

Figurer

1	Oversigt over Entity stamtræ	6
2	Oversigt over AI opbygning	7
3	Axis-Aligned Bounding Box (AABB) repræsentation.	8
4	Træ over klasser og grafiske elementer	10
5	Den grafiske brugergrænseflade og opdeling af komponenter . .	11
6	Det første UML-diagram	18
7	Endelige UML-diagram	19

Tabeller

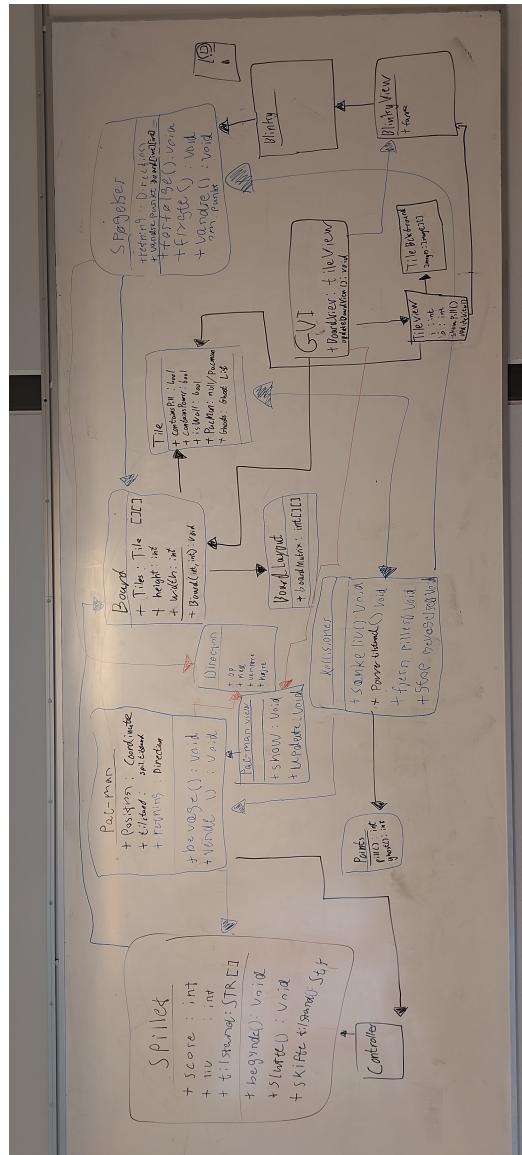
1	Oversigt over forskellen i implementering	15
---	---	----

7 Litteraturliste

- [Kis15] Kishimoto Studios. *AABB-AABB 2D Collision Detection*. Accessed: May 29, 2025. 2015. URL: https://kishimotostudios.com/articles/aabb_collision/.
- [Nam25] Bandai Namco. *Pac Man Arcade Game Carpet*. Accessed: May 27, 2025. 2025. URL: <https://www.pinterest.com/pin/pinterest--1016617315862571551/>.
- [Ope25a] OpenAI. *ChatGPT (GPT-4)*. Used as a sparring partner and for repetitive task assistance in this assignment. Accessed: May 26, 2025. 2025. URL: <https://chat.openai.com>.
- [Ope25b] OpenJFX. *JavaFX*. Accessed: May 27, 2025. 2025. URL: <https://openjfx.io/>.
- [Ora25] Oracle. *JavaFX 8 Documentation: Node*. Accessed: May 27, 2025. 2025. URL: <https://docs.oracle.com/javase/8//javafx/api/javafx/scene/Node.html>.

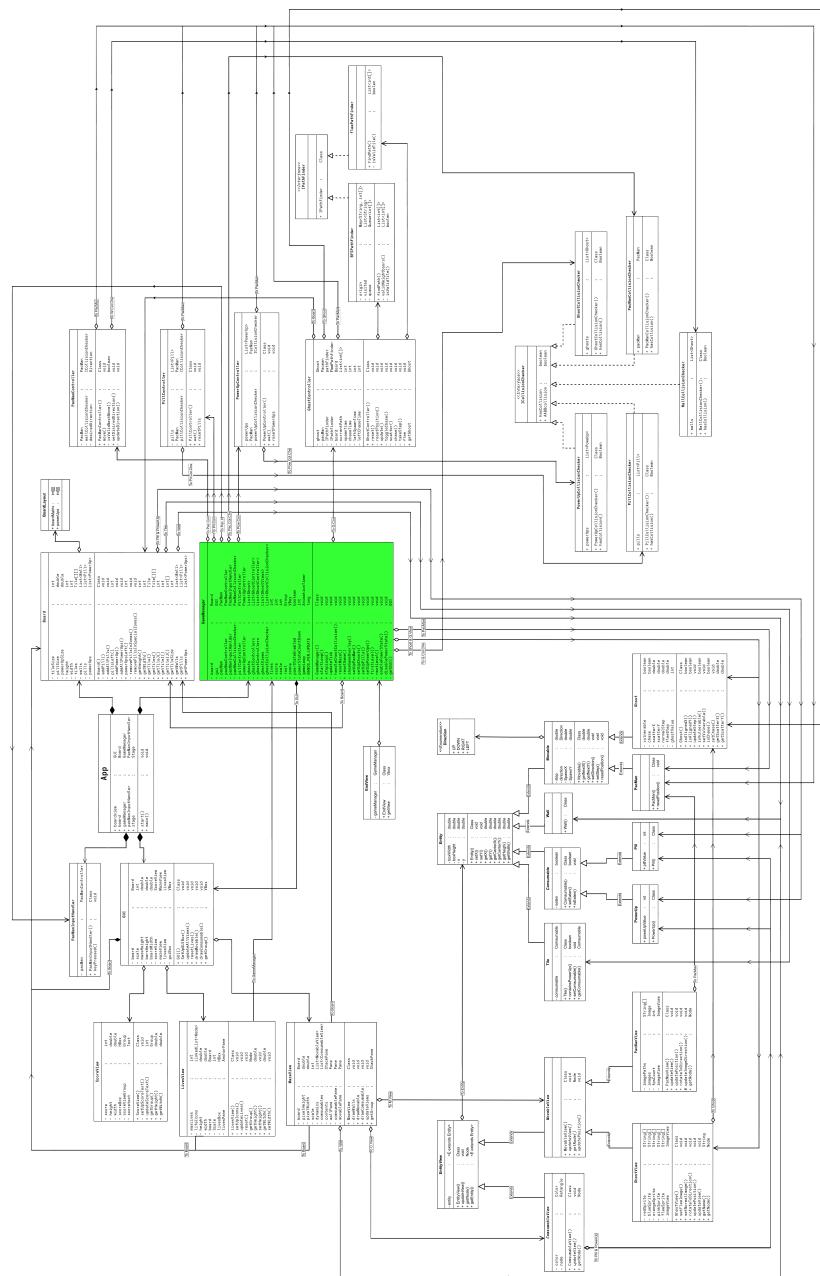
8 Bilag

8.1 Første UML-diagram



Figur 6: Det første UML-diagram

8.2 Endelige UML-diagram



Figur 7: Endelige UML-diagram