# Indholdsfortegnelse

## Liste over tabeller



## Liste over figurer

# 1 Indledning

Vi har valgt at beskæftige os med spilbalance, idet vi syntes at emnet med tiden er blevet et uhyre vigtigt aspekt af alle spil, samtidigt med at det kan afgøre et spils success meget hurtigt. De færreste vil spille noget der er ubalanceret, men der skal samtidigt være forskellige måder at spille på, og forskellige klasser/faktioner eller andet skal helst ikke have for meget til fælles. Samtidigt har den stigende online-verden gjort det muligt for spillerne at diskutere og gå i dybden med detaljerne, således at det at balancere er blevet endnu sværere for udviklerene, men at de samtidigt har en ekstrem god mulighed for at få feedback fra de mange spillere. Det oplagte eksempel her er *World of Warcraft* med dets, i skrivende stund, omkring 12 *millioner* spillere.

Vi har derfor valgt at stille spørgsmålet:

- *I hvor høj grad er det muligt at afgøre om et assymetrisk spil er balanceret?*

Begrundelsen for at vælge dette, er først og fremmest at uanset hvilken afskygning et spil har, er det nødvendigt at det har balance mellem spillerene, for at være spilbart. Dette gælder alle typer spil, om det så er bræt-, kort- eller computerspil, eller en blanding mellem de tre.

Valget på brætspilstypen er baseret på, at forskningen indenfor spilbalance har eksisteret i langt tid, og er godt gennemarbejdet. Desuden mener vi, at et brætspil af den type vi har lavet, vil kunne afspejle generel spilbalance rimeligt præcist, idet vi vil forsøge at få mange af de generelle grundlæggende balance-elementer med, og så vidt muligt undgå tilfældige faktorer.

## 1.1 Definition af balance

Før der kan skrives yderligere om balance, er det dog nødvendigt at give en formel definition. En af de mere kendte definitioner, der har været udgangspunktet til vores definition, lyder:

> *"A state of equilibrium or parity characterized by cancellation of all forces by equal opposing forces."* [Your Dictionary, 2010]

I større spil er det nærmest urimeligt indviklet, at redegøre for balance, og det sker ofte at spillerne (og herved strategierne) gør at et ellers balanceret aspekt kan blive ubalanceret, samt at et andet ubalanceret aspekt kan udnyttes til at gøre et helt tredje aspekt balanceret. Derudover kan en spiller også udnytte et specielt aspekt af et spil, som udviklerne ikke havde tænkt på, dengang det blev lavet. Dette kan ofte være en næsten uendelig cyklus, da det at løse et problem således, blot skaber et nyt et andet sted.

Af den grund, samt for at gøre det overskueligt for os selv, var det nødvendigt, at finde nogle underkategorier af spilbalance, at fokusere på, og her har vi primært valgt se på to typer, der kan benyttes til at påvirke balancen.

Den første skabes ud fra afvekslende parametre hos de forskellige spillere, som - hvis de bliver ændret - skubber balancen. Disse kan eksempelvis være helbredspoint, mængden af skade en spiller kan gøre, eller hvor mange felter der kan rykkes per tur[1].

Den anden er, hvordan en kunstig intelligens tager beslutninger og reagerer, alt efter hvilken "opførsel" de er sat til. Denne kan eksempelvis være offensiv, defensiv eller neutral, og er lavet således at der stræbes efter at tage optimale valg. Den præcise metode benyttet til disse Kunstige Intelligenser, er beskrevet i afsnit 4, side 17.

Ud fra alt dette, er vores opfattelse og definition af balance i et spil det følgende:

> *Spilbalance opfatter vi som værende den tilstand hvor alle parter af et spil som udgangspunkt har lige stor mulighed for at sejre, på trods af at deres udgangsmæssige parametre er forskellige.*
>
> *Således skal mængden af vundne spil være ligeligt fordelt mellem spillerne, hvor et uafgjort spil ikke tæller som værende vundet for nogen af parterne.*

## 1.2 Terminologi

Gennem rapporten, vil de følgende termer - udover dem der bliver introduceret i løbet af rapporten - optræde:

- **KI**: Kunstig Intelligens.
- **AP**: Adfærds parametre
- **PyGame**: Et `Python 3.1` framework designet til spiludvikling, som vi har brugt til at lave spillet.
- **Wrap-around**: En type af simulering af "en rund planet". Dvs. hvis en spiller rykker ud af brættet i højre side, vil han komme ind på brættet i venstre side.

Desuden er funktions-, fil- og parameternavne konsistent skrevet med `maskinskrift`, så det er let at identificere, og ikke er for svært at identificere i teksten.

[1] Vores valg af disse parametre er beskrevet mere detaljeret i afsnit 3.1.2, side 13.

## 1.3 Metode

For at undersøge spilbalance, har vi benyttet vores eget konstruerede brætspil, hvor vi ved hjælp af simuleringer får forskellige KIer til at spille mod hinanden, for derved at afgøre om spillet er balanceret.

Disse kunstige intelligenser er konstruerede ud fra en grad af genetisk programmering, hvor forskellige adfærds parametre tilpasses efter udfaldet mod en anden kunstig intelligens, i stil med "Survival of the Fittest"-scenariet. Den bedste strategi overlever. Hele processen og de valgte parametre er yderligere beskrevet i afsnit 4.4, side 19.

I og med at to kunstige intelligenser har samme præmisser for sejr og begge umiddelbart laver ikke-trivielle træk, er det eneste, der kan skabe ubalance mellem dem en ændring af de parametre, der beskriver en spillers styrker og/eller svagheder. Det er således disse, som danner udgangspunkt for balanceringen.

## 1.4 Afgrænsning

Da brætspil - så vel som kunstige intelligenser - i sig selv er et stort emne, har det af gode grunde været nødvendigt at begrænse os, hvilket vi har gjort indenfor områderne "Antal spillere", "Specifik type af brætspil" samt "Kunstige Intelligenser".

**Antal spillere:**
Vi har valgt at gennemføre spillet med to spillere, hvilket primært er gjort for at undgå, at arbejdet på KIerne ville blive signifikant højere og mere detaljeret, da der ville være meget mere at skulle holde styr på. Desuden ville der opstå en mulighed for en ubalance ved at nogle spillere kunne indgå en alliance mod den sidste, hvilket essentielt ville blive en tilfældighedsparameter, noget vi så vidt muligt har forsøgt at undgå.

Det skal dog nævnes at spillet sagtens ville kunne understøtte flere spillere uden de store problemer, og det ville langt fra være umuligt at implementere, da spillets regler først og fremmest lægger op til at kunne gøre det, samt at spillepladen sagtens kunne laves større. Kamp ville dog potentielt blive et problem, i og med at det bliver nødvendigt at vælge en modstander at angribe, hvis der er flere i rækkevidde.

**Specifik type af brætspil:**
Valget på en specifik type af brætspil var en meget naturlig begrænsning, for at gøre emnet mere præciseret, men også fordi det ville være umuligt at dække bare et par stykker. Typen af brætspil vi valgte er løst baseret på brætspillet "Diplomacy"[2], hvor forskellige spillere bevæger sig rundt på verdenskortet og kæmper for at opnå et bestemt antal resurser først. Vores

[2] En online version af spillet kan findes på [Diplomacy Online, 2010].

version er modificeret og stærkt simplificeret herefter. Dette betyder således at vores konklusioner kun er baseret på resultaterne af simulationer i vores eget spil.

**Kunstige Intelligenser:**
KIerne er begrænset til ikke at være alt for avancerede, da det primære fokus ligger på spil-balancen, og at KIerne kun er et middel til kunne simulere en masse spil i træk. Hvis KIerne har samme begrænsninger, må de nødvendigvis potentielt også være lige gode. Af den grund ser de kun "frem i tiden", og vurderer deres bedste træk ud fra den nuværende situation.

# 2 Spilteori og værktøjer

I dette afsnit vil vi kort udskitsere de forskellige redskaber vi har benyttet generelt, for at opnå det endelige resultat. Den primære del omhandler "Nash's equilibria", mens der også kort bliver beskrevet de to primære redskaber vi har brugt - "Genetisk programmering" og "Minimax" - til at få vores KIer til at bestemme deres valg. Hver af disse, og deres præcise brug, bliver desuden beskrevet i kontekst ved de passende punkter i løbet af rapporten.

## 2.1 Nash's equilibria

En af de mest udbredte teoremer indenfor spilbalance er "Nash's equilibria". Den lyder, i korte træk, på at ingen spiller på noget tidspunkt vil ønske at fortryde deres valg og ingen spiller på noget tidspunkt kan bruge en strategi som er bedre end hvad den anden spiller kan [Kolokoltsov and Malafeyev, 2010, Afsnit 1.7]. Arbejdet med "Nash's equilibria" gav desuden John F. Nash nobelprisen for økonomi i 1994[3]. En af fordelene ved "Nash's equilibria" er desuden også at den kan bruges i mange forskellige sammenhæng, fx. "Prisoner's Dilemma" [Kolokoltsov and Malafeyev, 2010, Afsnit 1.7] og "Best-Response Dynamics" [Kleinberg and Tardos, 2006, Afsnit 12.7].

Måden vi har benyttet den på er ikke direkte, men mere taget i brug som en mentalitet. Dette skal forstås således, at der på intet tidspunkt i spillet, hvis begge spillere spiller deres bedste, skal kunne laves et træk af den ene spiller, således at den anden spiller kan lave et der er mindst lige så godt for ham selv. Desuden skal der ikke være et givent træk der, i langt størstedelen af de mulige situationer, altid er bedre end et andet.

[3]Prisen fik han sammen med John C. Harsanyi og Reinhard Selden [Nobel Prize, 1994].

## 2.2 Redskaber brugt

Til at løse vores problemstilling, har vi gjort brug af en række forskellige udbredte programmerings-metoder. "Minimax" er brugt til at få vores KIer til at vælge deres næste træk med omhu, mens der er brugt "Genetisk Programmering" til at bestemme optimale sæt af parametre.

### 2.2.1 Genetisk Programmering

Genetisk programming (GP) er en programmerings-metode, hvor man designer et program, til at finde en løsning på et problem eller algoritme. Problemets natur kan være af vidt forskelligt karakter afhængigt af programmet. Et problem kunne eksempelvis være at finde løsningen til en matematisk formel, hvor et andet kunne være at få en programmeret bille til at bevæge sig rundt i et rum, hvor det gælder om at bruge mindst mulige skridt, for at få dækket rummets areal.

GP vil blive mere uddybet i afsnit 4.3, side 19, hvor det bliver beskrevet hvordan vi benytter det i sammenhæng med vores KIer. Til dette er der benyttet begreberne "Crossover" og "Mutation" beskrevet i [Affenzeiler et al., 2009, Afsnit 1.4.2,Afsnit 1.4.3].

### 2.2.2 Minimax

Minimax er en rekursiv algortime, som undersøger de forskellige muligheder i en given situation, og så kalder sig selv igen for hvert af disse. Når den i forvejen givne rekursionsdybde er nået, bliver positionen som minimax er nået til vurderet og returneret således at første iteration af muligheder bliver bedømt udfra deres "sub"-muligheder. Figur 1 skitserer et eksempel på Minimax, hvor der i hver iteration er 3 forskellige muligheder at følge.

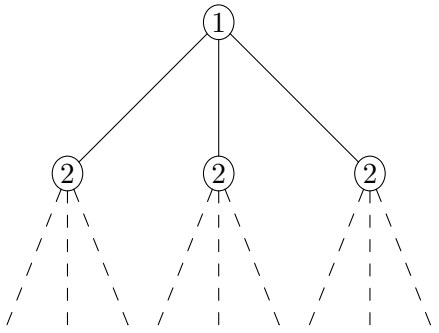


Figure 1: Eksempel på Minimax.

Det videre brug af Minimax bliver beskrevet mere detaljeret i afsnit 4.2, side 17.

# 3 Beskrivelse af spillet

Vores spil har to spillere, der kæmper om enten at dræbe modstanderen, eller opnå et fastlagt antal resurser. Spillerne kan i en given tur enten vælge at stå stille (dvs. defensivt træk), bevæge sig eller angribe den anden spiller (såfremt han er inden for rækkevidde). Trækkene bliver udført simultant, ved at spillerne hver især vælger deres træk, hvorefter spillet beslutter om dette er muligt, og derefter afgør udfaldet.

Angreb foregår efter princippet at angriberen skal have en fordel, ved at den forsvarende spiller, udover at tage skade, også modtager en nedsat mængde af resurser eller liv, alt afhængigt af hvilken type træk han laver.

Resurser fås efter hver tur, og bestemmes ud fra antallet af felter man ejer. Begge spillere vil enten få et fuldt antal resurser, eller en mindre mængde hvis de er under angreb eller har taget en modstanders felt. Et givent felt kan desuden kun ejes af den ene af spillerne, og alle felter er, som udgangspunkt, ejerløse (undtagen felterne spillerne i den givne tur er placeret på). En spiller ejer således altid mindst ét felt, hvilket vil resultere i at spillet ikke kan vare for evigt, da der i en given tur altid til være en eller af spillerne der får resurser og/eller tager skade.

Mere detaljeret, kan de forskellige kombinationer af udfald beskrives som i den nedenstående liste, ud fra synspunktet af hvordan trækket påvirker Spiller 1.

- Spiller 1 *bevæger* sig, Spiller 2...:

  ...*bevæger sig:* Fuld mængde resurser hvis feltet ikke ejes af nogen, halvt hvis modstanders samt evt. et felt mere.

  ...*angriber:* Ingen resurser, fuld skade og evt. et felt mere.

  ...*forsvarer sig:* Samme som ved bevægelse.

- Spiller 1 *angriber* Spiller 2...:

  ...*bevæger sig:* Der gives fuld skade, og fuld resurser.

  ...*angriber:* Begge spillere får halv skade, og ingen resurser.

  ...*forsvarer sig:* Der gives halv skade.

- Spiller 1 *forsvarer sig*, Spiller 2...:

  ...*bevæger sig:* Fuld helbredelse og halv resurser.

  ...*angriber:* Halvt helbredelse og ingen resurser.

  ...*forsvarer sig:* Samme som ved bevægelse.

I standard-tilfældet betyder dette således at angreb og forsvar går ud med hinanden skadesmæssigt, og angriberen har fordel ved at få resurser. Dette kan dog påvirkes i begge retninger, ud fra en given spillers assymetriske parametre, som senere beskrives i afsnit 3.1.2, side 13.

Ser man på dette i henhold til *Nash's equilibrium*, er dette tilfældet da begge spillere altid vil kunne de samme træk[4], og at ingen ubetinget vil have et bedre udfald efter en tur, lige meget hvad hver af spillerne trækker. Det antages naturligvis at hver spiller forsøger at tage et optimalt valg, fx. ville det være sub-optimalt at stå stille og helbrede, hvis man har fuldt helbred og modstanderen ikke er i nærheden.

## 3.1 Opbygning

I hvert af disse afsnit, vil de forskellige elementære gennemgående dele af spillet blive beskrevet, samt hvordan og hvorfor de er konstruerede som de er.

### 3.1.1 Spilbrættet

Spillet udføres på et bræt (I programmet kaldet `Board.py`) med heksagon-felter (kaldet `Fields.py`), som det er nødvendigt at gå i dybden med, da det er her grundlaget for at KIerne kan opfatte spillet, og se brættet på samme måde som os, bliver lavet. Figur 2 viser spillebrættet med de to spillere.

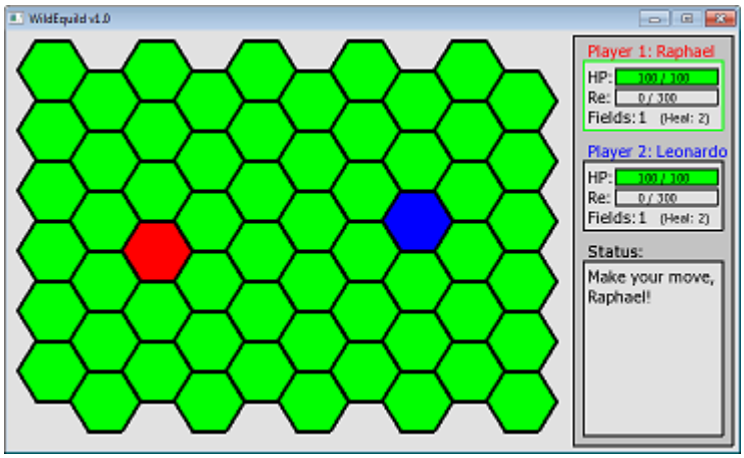


Figure 2: Eksempel på spillets udseende.

Det primære indhold i `Board` er værdier for højde og bredde af brættet, samt en liste af lister af `Fields`, hvor placeringen heri svarer til deres placering i

[4]Dette kan dog ændres, ved brug af en af de assymmetriske parametre `attackRange`, beskrevet dybere i afsnit 3.1.2, side 13.

spillebrættets koordinatsystem. Et `Field` er således den "fysiske" repræsentation af et felt på brættet[5]. Hvert af disse indeholder det følgende data:

- Om feltet er "optaget" (dvs. om der står en spiller i forvejen eller ej).
- Om feltet er "ejet" (er som udgangspunkt **None**, indtil en spiller rør det).
- To lister med punkter for de 6 hjørner af den heksagon som repræsenterer feltet, genereret ud fra feltets koordinat-sæt. Disse bruges til at optegne feltet, og bruges udelukkende i den grafiske del af spillet.

Eftersom heksagoner ikke naturligt passer ind i et klassisk rektangulært koordinat-system, samt at brættet skulle understøtte *wrap-around*, var det nødvendigt at lave de følgende regler for hvordan brættet skulle opfattes:

- Brættets koordinatsystem starter i $(0, 0)$ i ØVERSTE venstre hjørne. Dette på grund af *PyGame*s tegne-funktioner, der bruger dette format.
- Et felts koordinat-sæt er altid begge enten lige eller ulige. Dette er gjort for at gøre det konsistent og overskueligt.
- Brættets dimensioner (højde og bredde) er altid begge lige. Hvis dette ikke var tilfældet, ville felterne ikke passe sammen iht. *wrap-around*, og der ville være felter, der overlappede.
- En heksagons nabofelter ligger som følger ud fra udgangsfeltet i koordinatsystemet: ovenfor $(0, -2)$, nedenfor $(0, +2)$, oppe til højre $(+1, -1)$, oppe til venstre $(-1, -1)$, nede til højre $(+1, +1)$, nede til venstre $(-1, +1)$.

Dette betyder desuden også at der ikke er `Fields` på alle koordinater - f.eks. er $(1, 2)$ markeret som et **None** objekt i listen. Et godt eksempel på hvordan sammenhængen mellem felterne er, kan ses på figur 3, side 12.
Som et resultat af disse regler, var det nødvendigt at lave en række funktioner der kunne udregne de nødvendige relationer, således at brættet kunne opføre sig som det så ud, samt at de kunstige intelligenser ville have en mulighed for kunne "se" brættet som vi kan, og derved få de nødvendige midler til at lave meningsfulde beslutninger. Der blev således lavet følgende funktioner[6], der alle tager højde for brættets *Wrap-around*:

- `distance`:
  Finder afstanden mellem to punkter på brættet, i antal felter. Wrap-around bliver behandlet ved at tage vælge de mindste mulige x og y-mæssige distancer, således at det bliver sikret at den endeligt udregnede distance er den korteste mulige.

[5] Logikken for navnet, ligger i betydningen *"en mark"*, eller *"et område"*.
[6] Der kan findes i spillets `Board.py`-modul.

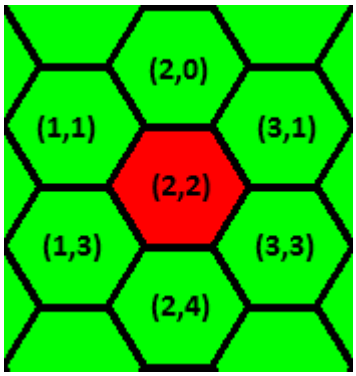


Figure 3: Felternes koordinater på brættet.

- `getBorderingPoints`:
  Gør som navnet siger, og finder de 6 felter der grænser op til feltet på det give punkt. Disse bliver returneret sammen med en parameter der svarer til de forskellige retninger spilleren kan vælge at rykke i, som følger beskrevet i tabel 1, side 12. Er punktet eksempelvis $(2, 2)$, vil indeks **UL** fra resultatet give punktet $(1, 1)$, som vist på figur 3, side 12.

| Kort navn: | Retning: | Koordinatændring: |
|---|---|---|
| **UL** | Oppe til venstre (**U**p **L**eft) | $(-1, -1)$ |
| **U** | Lige ovenfor (**U**p) | $( 0, -2)$ |
| **UR** | Oppe til højre (**U**p **R**ight) | $(+1, -1)$ |
| **DL** | Nede til venstre (**D**own **L**eft) | $(-1, +1)$ |
| **D** | Lige nedenfor (**D**own) | $( 0, +2)$ |
| **DR** | Nede til højre (**D**own **R**ight) | $(+1, +1)$ |

Table 1: Retninger der kan bevæges på spillebrættet.

- `getPointsWithDistance`:
  En mere generel version af `getBorderingPoints`, der returnerer en punktliste med en given distance fra udgangspunktet. Således vil `getPointsWithDistance`$(1)$ returnere de samme punkter, men disse vil blot ikke være indekseret efter retninger.

- `isPlayerInRange`:
  Denne funktion benytter `distance` sammen med den nuværende spillers angrebsvidde, til at bedømme om den anden spiller kan nås ved et angreb.

### 3.1.2 Spillere

En given spiller (`Player.py`-modulet) bliver bestemt ud fra en placering på brættet, et identifikationstal (0 og 1 for hhv. spiller 1 og 2), et navn, en farve (til brug i den grafiske repræsentation) samt et sæt af asymmetriske parametre, der benyttes til at bestille spillerens "klasse", beskrevet i næste afsnit.

Der bliver der derefter lavet en liste til at holde styr på hvilke felter spilleren ejer, samt variable og funktioner til spillerens resurser og liv.

**Assymmetriske Parametre:**
Parametrene der bruges til at bestemme de forskellige klasser, er beskrevet i tabel 2, side 13.

| **Parameter:** | **Beskrivelse:** | **Standard værdi:** |
|---|---|---|
| `maxLife` | Maksimalt antal livspoint | 100 |
| `attackRange` | Angrebsrækkevidde i felter | 1 |
| `healingRate` | Effektiviteten af helbredelse | 2 |
| `attackMultiplier` | Styrken af angreb | 2 |

Table 2: Beskrivelser af en spillers assymetriske parametre.

Disse påvirker således ikke spillerens stil direkte, men giver mulighed for forskellige "figurklasser". Grunden til at standardværdierne for `healingRate` og `attackMultiplier` er 2, er for at denne både kan halveres uden at skulle ud i halve værdier, og at der således stadig kan bruges heltal.

**Overordnede spillertyper:**
Der er desuden to forskellige overordnede typer: `AIPlayer` og `HumanPlayer`, værende henholdsvis en KI og en menneskestyret spiller. Den eneste rigtige forskel mellem de to ligger i metoden `determineMove`, hvori deres træk bestemmes:

- Den menneskestyrede afventer et korrekt taste-tryk svarende til trækket.
- Den kunstige intelligens udregner/bestemmer hvilket træk der udføres, ud fra et variabelsæt indholdelse forskellige parametre der bestemmer dens spillestil, og hvor høj betydning de forskellige træk har. Dette beskrives dybere i afsnit 4, side 17.

### 3.1.3 Spilcyklus

Spillets forløber ganske simpelt. Siden turene er simultane, vil spillernes træk blive udført samtidigt, men vil af gode grunde ikke kunne "opsamles" således. Dette betyder dog ikke at den anden spiller har nogen fordel, da

han ikke har mulighed for at se eller bedømme noget ud fra den anden spillers træk. Der hele sker således på følgende måde:

1. Spiler 1 og Spiller 2 spørges om hvad de vil, i den rækkefølge.
2. Det tjekkes om der opstår konflikt mellem trækkene (om spillerne ender på samme punkt)
3. I tilfælde af step 2, startes der forfra.
4. Udfaldet af spillernes angreb udført, hvis der er nogle.
5. Bevægelser udføres for de spillere, der ikke har angrebet.
6. Helbredelse udføres for de spillere, der ikke har angrebet eller bevæget sig.

Mellem hver cyklus tjekkes følgende for at afgøre om spillet er afgjort, i denne rækkefølge:

1. Om begge spillere er døde: *Uafgjort*.
2. Om en af spillerne er døde: *En spiller vinder*.
3. Om begge er nået det maksimale antal resurser: *Uafgjort* (hvem der "har mest" gør ingen forskel).
4. Om en af spillerne har nået det maksimale antal resurser: *En spiller vinder*.

Kort fortalt; en spiller, der har opnået det maksimale antal resurser, kan ikke vinde, hvis han er død.

## 3.2 Undermoduler

Ud over de basale elementer i spillet, var der desuden også udviklet en række undermoduler, der udvider den basale førnævnte model til forskellige tilstande. Først og fremmest kan spillet vises grafisk, så menneskelige spillere kan spille mod hinanden eller en KI. Dernæst, er der et sæt moduler der gør at spil bliver gemt i logfiler, således at et givent spil altid kan genses. Desuden kan en større mængde spil mellem KIere simuleres uden det grafiske aspekt.

### 3.2.1 Grafisk repræsentation

Når spillets grafiske repræsentation startes via `WildEquild.py`, bliver der sat en udvidet spilcyklus[7] igang i `Game.py`, hvor tegnemodulet `Draw.py` og statuspanelet `Statspane.py` bliver initialiseret. `Draw` indeholder samtlige funktioner til at grafisk repræsentation af spillebrættet, mens `Statspane` viser mængden af liv og resurser for spilerene, samt hvilken spiller der skal lave sit træk.

Den grafiske repræsentation af spillet er desuden et "krav" for at benytte menneskestyrede spillere, da det ellers ville være grænsende til det umulige at danne et overblik over informationsmængden.

### 3.2.2 Simulation af spil

Den grafiske repæsentation er rigtig god til at danne sig et overblik over hvad der foregår, men den er langt fra optimal når det kommer til at håndtere simulerede spil imellem to KIer, og slet ikke flere hundrede KIer på samme tid, som er et krav, for at udviklingen af vores KIer skal gå bare nogenlunde hurtigt.

Til at håndtere dette stykke arbejde, er der blevet lavet 3 forskellige klasser. `Simulation.py` har samme funktion som `Game.py`, blot uden de grafiske dele inkluderet. Derudover, så er der også en *nolog*-version, som bruges i Minimax-delen, så de træk der bliver regnet på, ikke bliver skrevet til loggen.

`Genetic.py` er blevet beskrevet tidligere, og det er den fil som har med udviklingen og udvælgelsen af KIerne at gøre. `Play.py` er i teorien bare `WildEquild.py` uden det grafiske og menneskelige input. Derudover, så har den også den ekstra egenskab, at den startes som en *thread*, hvilket vil sige at man køre et stort antal spil parallelt med hinanden. Dette er nøjagtigt hvad der er brug for i forbindelse med `Genetic.py`.

### 3.2.3 Logning og genspilning

Spillet indeholder en metode til at logge et helt spil, og gøres ved at modulet `Logger.py` initialiseres når et spil startes. Spiller- og brætobjekterne gemmes direkte ved brug af en metode kaldet *pickling*, der essentielt betyder at der gemmes en tro kopi af objekterne fra da spillet blev spillet. Herefter bliver hvert af trækkene skrevet ned i logfilen, i rækkefølgen de blev udført.

Modulet `Repickler.py` gør således det at den henter objekt-kopierne fra en valgt logfil, og skaber en grafisk repræsentation af spillet. Dette gøres ved alle trækkene deles efter hvilken spiller der lavede dem, og de parvist udføres. Spilteknisk set bliver spillet spillet som normalt, hvor alle trækkene

[7] Den simple version er beskrevet i afsnit 3.1.3, side 13.

blot er bestemt på forhånd. Siden der ikke er nogle udfald der er tilfældige, vil udfaldet af spillet således være præcis det samme som da det original blev spillet.

Logikken bag denne udvidelse, var at et spil mellem to KIer kunne undersøges under kontrollerede forhold, og i den hastighed som vi selv ønskede. Dette er specielt brugbart når der laves en simulation, da disse som standard foregår uden grafisk repræsentation. Man kan derefter se et udvalgt spil i dybere detaljer, sammen med de valgte spilleres parametre, som demonstreret på figur 4, side 16. Parametrene er skrevet i spillernes rækkefølge.

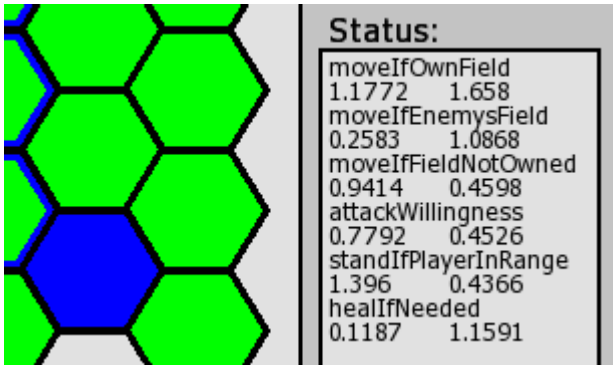


Figure 4: Statusområdet i `Repickle`.

# 4 Kunstige Intelligenser

For at kunne simulere spillet ordenligt, var det nødvendigt at lave en konsistent KI, som vi vidste ville opføre sig på en kontrolleret måde, og lave gode velovervejede træk. Hele essensen af, hvordan KIerne fungerer, findes i funktionen `determineMove`, der, som navnet antyder, vurderer hvilket træk, der skal tages. Dette er faktisk også den eneste egentlige forskel mellem `HumanPlayer` og `AIPlayer`, der er henholdsvis den menneske- og KIstyrede subklasse til `Player`.

Måden det bedste nuværende træk findes, bestemmes ud fra *Minimax*-funktionen (`Minimax.py`-modulet), som bliver kaldt i `determineMove`. Begge funktioner bliver beskrevet mere nøje i et senere punkt.

## 4.1 Adfærds-parametre

Vurderingen af et givent træk sker ud fra forskellige AP, alt afhængigt af hvilket træk der er tale om. Disse har en baseværdi på 1, hvilket vil sige, at KIen ikke bliver påvirket af den givne AP. Vurderingerne kan variere baseværdien med $\pm 1$, og den endelige vurdering af et felt er alle disse værdier ganget sammen, for hvert af de mulige træk. I tabel 3 beskrives navnet, hvilken type af træk (hhv. *bevægelse*, *angreb* eller *stå*) den bruges til, samt overvejelsen der vurderes for de forskellige værdier.

| **Parameter:** | **Type:** | **Overvejelse:** |
|---|---|---|
| `moveIfOwnField` | bevægelse | Er feltet mit eget? |
| `moveIfEnemysField` | bevægelse | Er feltet ejet af modst.? |
| `moveIfFieldNotOwned` | bevægelse | Er feltet ikke ejet? |
| `attackWillingness` | angreb | Er modst. i rækkevidde? |
| `standIfPlayerInRange` | stå | Er jeg indenfor modst. rækkevidde? |
| `healIfNeeded` | stå | Har jeg brug for helbredelse? |

Table 3: Beskrivelser af en Kunstig Intelligens parametre.

## 4.2 Minimax

Den måde Minimax er implementeret på, er at den søger ud på brættet fra spillerens nuværende position, og får en vurdering af de omkringliggende felter ved hjælp af funktionen `evalPosition`. Søgningen sker ved at der bliver lavet et træ for samtlige mulige træk i den givne runde. Hvert træ får grene som består af samtlige mulige træk, præcis som hvis man havde lavet det træk som først startede træet.

Der bliver vurderet 3 træk ud i "fremtiden". Dog bliver dybden på de træer som Minimax laver ikke større end 2. Grunden til dette, er at når man

når rekursionsdybden, bliver `evalPosition` kaldt, der vurderer de omkringliggende felter, hvilket giver en samlet vurderingsdybde på 3.

Dybden af Minimax kan godt være større end 2, men på grund af størrelsen af spillebrættet, vurderede vi 2 til at være passende. Derudover betyder dette, at vi sparer en masse vurderinger, hvilket forbedrer køretiden markant, uden at gøre nogen videre forskel i de endelige vurderingers udfald. På nuværende tidspunkt bliver der lavet imellem $6^2$ og $8^2$ vurderinger per træk, hvor en forøgelse af dybden med 1 ville føre til mellem $6^3$ og $8^3$ vurderinger.

Grunden til dette, er at der er 8 forskellige træk, fordelt på bevægelse i seks forskellige retninger, at stå stille, samt angreb. Da det at stå stille og angribe er afhængige af faktorer, som at være inden for modstanderens rækkevidde og om modstanderen er inden for ens egen rækkevidde, strækker antallet af vurderbare træk sig fra 6 til 8.

Minimax bliver kaldt fra funktionen `determineMove`, som får værdierne fra de omkringliggende felter, og så afgører hvilket træk KIen skal lave. Værdierne bliver rundet af til 2 decimaler, så hvis der er mere end en retning som har værdier tæt nok på hinanden, bliver de sat til samme værdi. Sidst vælges en tilfældig af de højeste værdier.

Dette gøres for at det ikke skal blive alt for forudsigeligt hvordan KIen bevæger sig. Især i starten af spillet, vil mange af felterne returnere samme værdi, da der kun er tomme felter, samt modstanderen at tage højde for.

### 4.2.1 Evaluering af placeringer

`evalPosition` bliver kaldt når rekursions-dybden er blevet nået. Funktionen vurderer feltets omgivelser, som man er kommet til og returnerer den værdi feltet er blevet vurderet til, tilbage til Minimax.

Vurderingen bliver lavet ud fra KIens AP. Hvis modstanderen er inden for rækkevide til et angreb, bliver der taget højde for de statistikker som har med angreb at gøre, ellers ikke. Ligeså bliver der kun taget højde for en AI's lyst til at helbrede sig (ved hjælp af et defensivt træk), når den ikke har fuldt helbred. Hvis KIen er inden for modstanderens angrebsrækkevide, så er det dog også muligt at lave et defensivt træk, da den så vil tage halv skade, hvis den bliver angrebet.

Udover den offensive og defensive adfærd, er der også bevægelsesadfærden. Denne bliver afgjort af de felter som der rundt omkring feltet man er nået til, samt det felt man står på. Værdien feltet får, bliver så afgjort ud fra AP og de forskellige type felter der findes i spillet, ens egne, modstanderens eller ingens.

I sidste ende bliver det så vurderet hvilken adfærdstype der er størst, og det er den værdi der bliver returneret.

## 4.3 Genetisk programmering

I vores tilfælde bliver genetisk programmering brugt til at spille vores spil på en ordentlig måde. Ved det skal der forstås, at man godt kan spille vores spil ved at lave helt tilfældige træk, men at det bedre at have en ide om hvad man laver, og hvilke træk man foretager sig.

Dette er vigtigt, da man ved hjælp af velovervejede træk undgår de faldgruber der er ved tilfældige træk, som at komme til og gå frem og tilbage på de to samme felter konstant. Dette sørger for at man kun samler 2 resurser ind lineært hver runde, hvor at man helst skulle prøve at opnå en kumulativ stigning i resurser, for at kunne vinde den vej.

Vi har benyttet, og udviklet, nogle værktøjer til at hjælpe KIerne med at spille spillet, og senere hen udvikle dem, så der efter hvert iteration af spillede spil, skulle være færre og færre KIer som var dårlige til spillet.

## 4.4 Udvikling af de kunstige intelligenser

Til at udvikle KIerne gøres der brug af nogle forskellige teknikker. Måden teknikkerne implementeres på, er ikke fastlagt på forhånd, og det har derfor været op til os at få dem implementeret på en måde, som vi synes passer til vores problem og program.

### 4.4.1 Fitness

*Fitness* er den værdi som afgør hvor godt en KI har klaret sig. De faktorer der spiller ind i vores udregning, er hvor mange resurser og hvor meget liv KIen har, når spillet slutter, i forhold til det fastsatte maksimum.

En anden faktor er hvor mange ture KIen har brugt på et spil. Denne værdi sættes op imod en ideal værdi, som til at starte med er på 25 ture, men som har muligheden til at ændre sig efter nogle iterationer af spillet.

Grunden til at ideal-værdien er dynamisk, er at KIerne gerne skulle blive bedre til spillet, hvilket gerne skulle give færre ture, men eftersom at KIens modstander også bliver bedre, så kan antallet af ture variere fra cirka 7 til 150 ture, alt afhængigt af om de går direkte mod hinanden og angriber eller om begge spillere bare bevæger sig frem og tilbage på de samme to felter.

Den sidste faktor der spiller ind, er hvilken placering KIen har fået efter spillet, om den har tabt, vundet, eller om spillet endte uafgjort. Da det er vigtigt at vinde, har vi valgt at øge vindernes fitness-værdi med 20%, og ligeledes har dem der spillede uafgjort, fået deres fitness-værdi øget med 10%.

Dette er gjort, da forskellen på to KIers fitness værdi, fra samme spil, ville være af næsten ingen betydning, hvis den ene KI havde vundet med f.eks et point forskel i resurser. Selv om de to KIer har klaret sig næsten lige

godt, så handler spillet om at vinde, og derfor er det vigtigt at fremhæve vinderne fremfor taberne.

Det som fitness-værdien bruges til, er at udvælge de KIer, som skal have en chance for at udvikle sig til nogle nye, og potentielt bedre, KIer. Måden udvælgelsen sker på, er ved at fitness-værdien bliver udregnet for alle KIerne som har deltaget i spillene, og derefter bliver alle KIer som har en fitness-værdi over gennemsnittet, så udvalgt til at udvikle sig.

Af de KIer som er blevet udvalgt, bliver der valgt ca. 60% fra til `crossover`, og yderligere 20% bliver valgt fra til `mutation`. De resterende KIer kommer med i næste iteration af spil uden nogle modifikationer.

### 4.4.2 Crossover

*Crossover* fungerer ved at der bliver lavet to `child` KIer ud fra to `parent` KIer. `Child` KIerne består, ligesom ved mennesker, af forældrenes statistikker, men det er tilfældigt, for hver gang der bliver lavet *crossover*, hvilke AP der bliver arvet af `parent` KIerne.

Dette skulle gerne føre til at de KIer, som har arvet svage AP bliver udfaset efter nogle iterationer af spil, og dem med stærkere AP forbliver i spillet. Stærk og svag AP skal forstås ved at vores spil kan være lavet på en sådan måde, at det farvoriserer angrebs-trækket over andre træk, og derfor ville en angrebs-lysten KI vinde over en mindre angrebs-lysten KI.

På forhånd er det ikke givet hvilke AP der er stærke, og hvilke der er svage, og da der findes forskellige typer af spillerklasser, så kan nogle AP være stærke hos en klasse, og svage hos en anden.

### 4.4.3 Mutation

*Mutation* er en ændring i værdi på en tilfældig AP hos en KI. Umiddelbart har *mutation* ikke den store indvirkning på en KIs adfærd, men da det er muligt at ramme en stærk AP, er det muligt at ændre en KIs adfærd drastisk.

*Mutation* kunne godt have ændret flere AP end en, men ikke alle AP'ene, da det ville svare til at lave en hel ny KI istedet. Grunden til at det kun er en AP der ændrer sig, er som nævnt tidligere, at ændringen strækker sig fra ingenting til en drastisk ændring, og jo flere AP der ændres på en gang, jo større chance er der for at lave en drastisk ændring. En drastisk ændring i sig selv er ikke noget slemt, men for mange drastiske ændringer gør brugen af mutation nytteløs, da der, i sidste ende, vil være for lidt af den oprindelige KI tilbage, til at kunne genkende den.

# 5 Tests og Simulationer

Måden applikationen er blevet skrevet og konstrueret, har gjort at traditionelle unittests iht. applikationens funktionalitet har været udeladt. Dette betyder dog ikke at det ikke er testet igennem, da alle implementerede funktioner blev tjekket da de blev skrevet, således at de virkede som forventet. Ellers er fejl og andet rettet undervejs i processen, og under testkørsler af simulations-modulet.

Det kan tilnærmelsestvist siges at tests af applikationen har været "vandfalds"-processen, ved at vi først løbende har udført egne tests under implementeringen, dernæst brugertests for at se på udefrakommendes meninger, og sidst simuleringerne med KIerne.

## 5.1 Brugertests

Der blev taget overraskende mange designmæssige beslutninger på baggrund af brugertests. Siden det aldrig var intentionen at spillet skulle være balanceret mod mennesker, et emne der også bliver diskuteret senere i afsnit 6.2, blev disse primært brugt til at prøve spillet af mod en person der ikke kendte spillet i dybden, og derfor kunne give et neutralt og friskt input, samt se ting som vi ellers havde set os blinde på.

Alt i alt var disse utroligt givende, og var passende at få udført på et nogenlunde tidligt stadie af spillets udvikling, hvor KIerne endnu ikke var færdigtudviklede.

## 5.2 Simulation af Kunstige Intelligenser

Den vitale del af testsne var simulationerne, der var lavet i forskellige stadier. Først en række tests uden assymetriske parametre, for at undersøge om KIernes adfærdsparametre får lavet forskellige typer, og at fordelingen mellem vundne kampe af begge spillere er ligelig. Den interessante og fundamentale del af opgaven ligger i dog i de tests hvor de assymetriske parametre er taget i brug, og vi vil her bruge en stor mængde forskellige kombinationer, som beskrives nærmere i de respektive afsnit.

Simulationerne er delt op i tre primære forskellige dele, henholdsvis med og uden spillerparametre, samt hvor en af spillerne gives en gradvis større fordel. Til hver af disse test-afsnit er der kørt 10 x 100 spil, hvilket således giver 200 forskellige KIer bare i den første af de 10 iteration. De detaljerede resultater af testne vil blive vist i grafform, under den beskrevne simulering, mens de mere detaljerede resultater der er brugt til at lave grafer kan findes i appendix A.1.

I hvert af underafsnittene bliver der kigget kort på resultaterne, som så munder ud i en længere analyse i afsnit 5.2.4.

### 5.2.1 Uden assymetriske parametre

Denne serie tests er primært lavet for at se om der, i tilfældet hvor spillerne er lige stillede, vil være balance. Dette skulle helst være tilfældet for at der kan snakkes om betydningen ved en afvigelse. Resultaterne kan ses på figur 5, side 22.

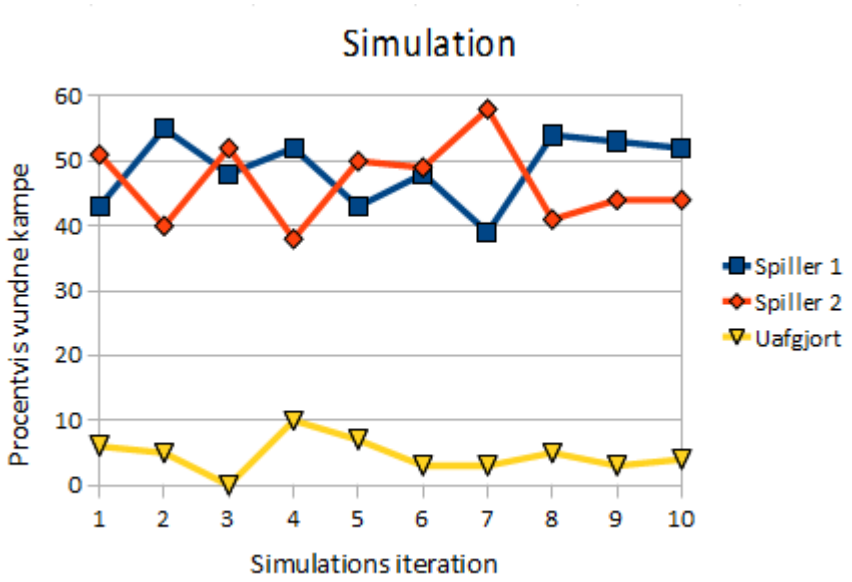


Figure 5: Simulation: Normal vs. Normal

Som det ses, ligger resultaterne meget tæt, og ved nærmere gennemgang af resultaterne ses det at forskellen mellem vundne kampe afviger med 2%, fordelt på 48.7% og 46.7%, med 4.6% uafgjorte.

### 5.2.2 Med assymetriske parametre

Ved brug af de assymetriske parametre, konstruerede vi 3 nye klasser, som kan ses afbilledet i tabel 4, hvor "Normal" er standardværdierne fra hvor de forskellige parametre blev beskrevet i tabel 2, side 13.

| **Klasse:** | `maxLife` | `attackRange` | `healingRate` | `attackMultiplier` |
|---|---|---|---|---|
| **Normal** | 100 | 1 | 2 | 2 |
| **Mage** | 80 | 2 | 2 | 2 |
| **Healer** | 80 | 2 | 3 | 1 |
| **Fighter** | 120 | 1 | 1 | 3 |

Table 4: Spillerklassernes assymetriske parametre.

Som i forrige afsnit, er hver af disse hver især testet mod hinanden, således

at hver af spillerne konsistent er en specifik klasse. Sammenhængen mellem de forskellige klasser iht. simulationerne kan ses i figur 6, hvilket viser at der er 6 forskellige kombinationer.

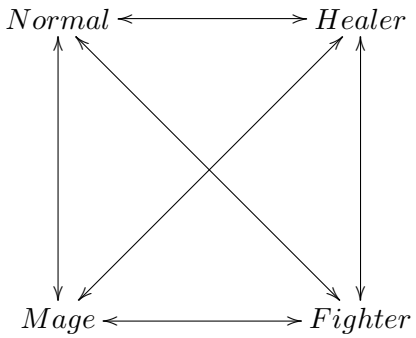


Figure 6: Simulations relationer.

Der er bevist ikke lavet tests mellem to ens klasser, da dette ville give samme udfald som uden de assymetriske parametre. Resultaterne af disse tests kan ses i figur 7-12, side 24-26.

Som det ses på graferne, er der få tilfælde hvor forskellen mellem de to spillere afviger mere end 5%. Værste tilfælde er "Mage vs. Fighter" hvor forskellen er 6,6%, mens bedste tilfælde er "Healer vs. Fighter" hvor forskellen kun er 1,6%. Generelt set er disse således ganske balancerede, med **Mage** værende den generelt bedste af disse, ved at altid at have vundet mellem 2,6% og 6,6% flere kampe.

### 5.2.3 Gradvis åbenlys ubalance

For at yderligere vise at udfaldene af simulationerne ikke er tilfældie, satte vi den normale spillerklasse op mod "Goliath" - en figurklasse der havde stigende bedre parametre, så det kunne vises hvor stor en forskel de forskellige ændringer i parametrene kunne gøres. De 4 forskellige grader er beskrevet i tabel 5.

| **Klasse:** | `maxLife` | `attackRange` | `healingRate` | `attackMultiplier` |
|---|---|---|---|---|
| **Normal** | 100 | 1 | 2 | 2 |
| **Goliath 1** | 200 | 2 | 2 | 2 |
| **Goliath 2** | 300 | 3 | 3 | 3 |
| **Goliath 3** | 400 | 4 | 4 | 4 |
| **Goliath 4** | 500 | 5 | 5 | 5 |

Table 5: Den gradvist stigende spillerklasse Goliath.

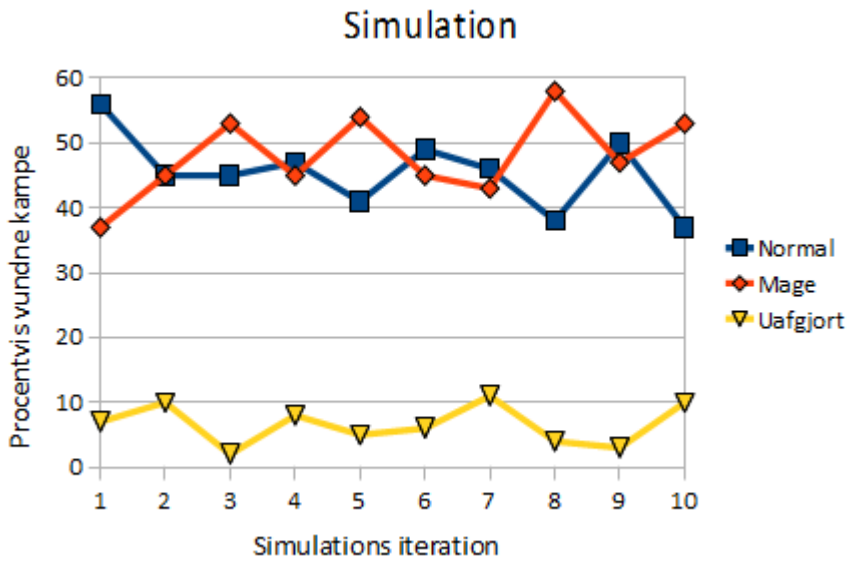


Figure 7: Simulation: Normal vs. Mage

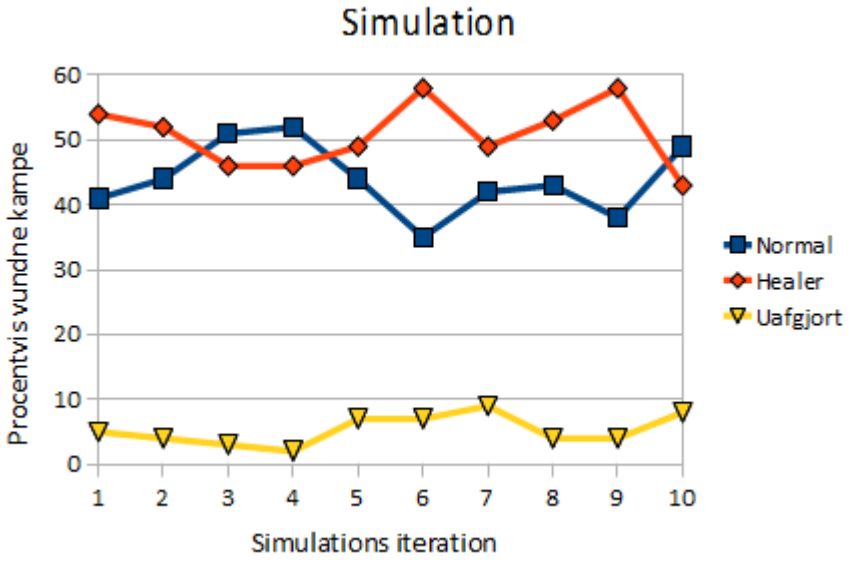


Figure 8: Simulation: Normal vs. Healer

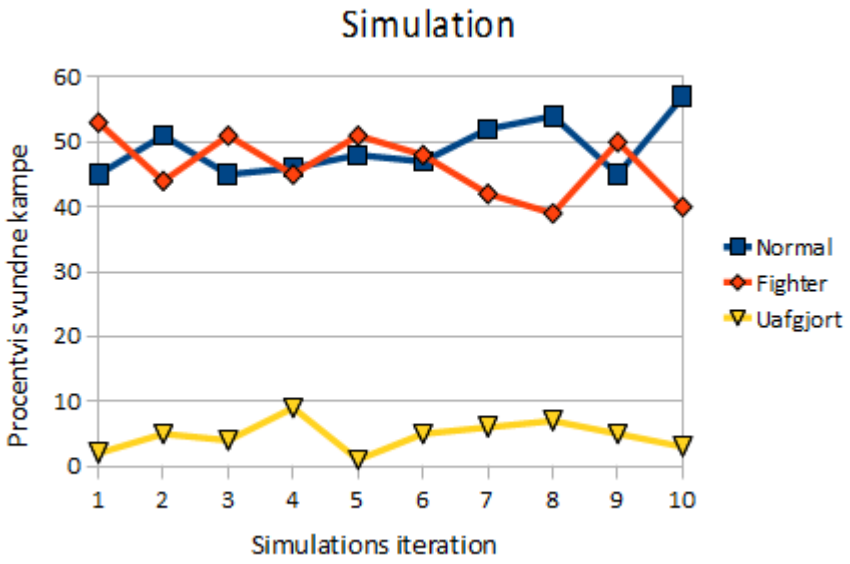


Figure 9: Simulation: Normal vs. Fighter

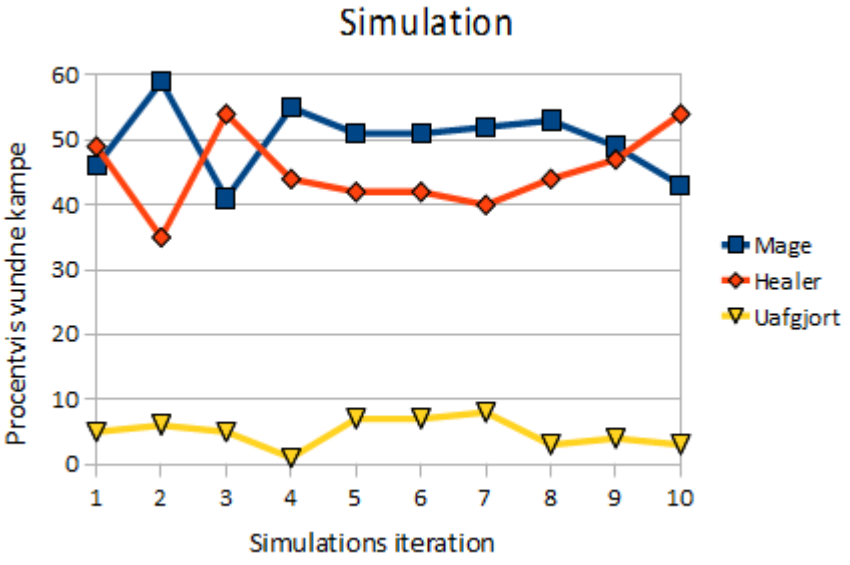


Figure 10: Simulation: Mage vs. Healer

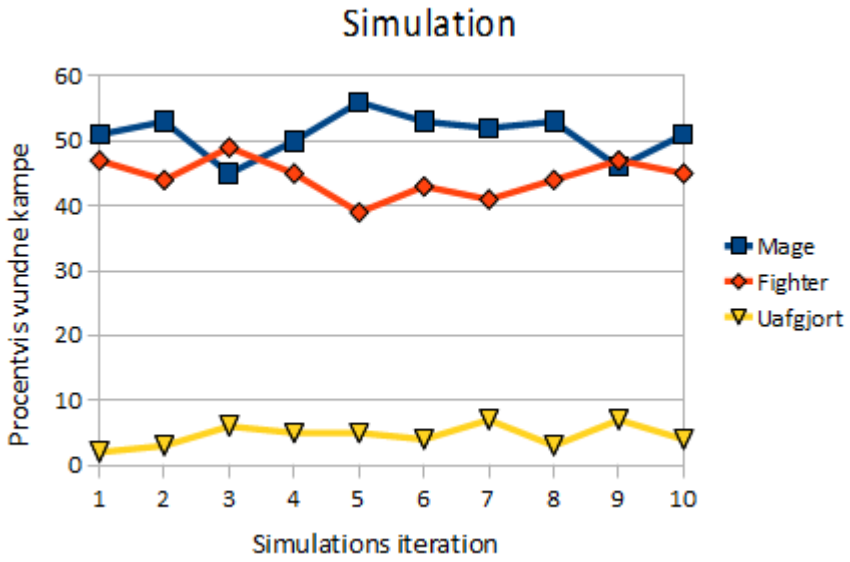


Figure 11: Simulation: Mage vs. Fighter

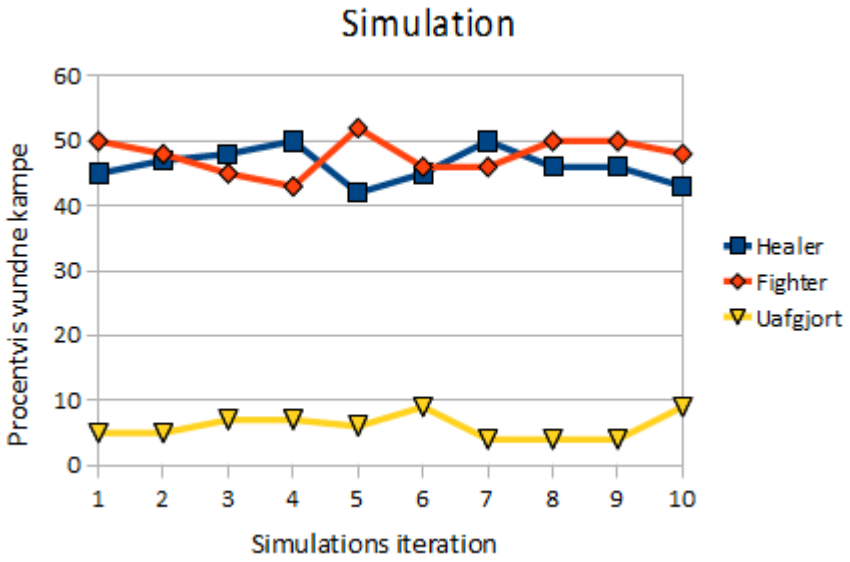


Figure 12: Simulation: Healer vs. Fighter

Resultaterne af simulationerne kan ses i figur 13-16, side 28-29. Allerede ved niveau 1 ses en forskel på hele 10,8%, som derefter bevæger sig op til henholdvis 27,5%, 72,4% og 94,2% for henholdsvis niveau 2, 3 og 4.

### 5.2.4 Sammenfatning af assymetriske simulationer

Ud fra graferne kan man læse, at de forskellige klassetyper er nogenlunde lige balanceret, og dette er på trods af at implementering af Minimax, og KIerne, ikke fungerer efter hensigten. Det er ikke umiddelbart til at se ud fra graferne med standardklasserne, men hvis vi ser på "David og Goliath"-graferne, er det tydeligt, at det sagtens kan lade sig gøre, og skabe en decideret uretfærdig og ubalanceret klassetype.

Det fremgår også tydeligt fra graferne, at der er nogle statistikker, som er mere følsomme, og sværere at balancere end andre. Den statistik som har størst betydning for spillet, er hvor langt en klasse kan skyde. Grunden til dette, er at selvom det og slå hårdt kan være effektivt, så nytter det ikke noget, hvis modstanderen holder sig ude fra ens rækkevidde. Derimod, så hvis man næsten ingen skade giver, men kan skyde rigtig langt, har man en klar fordel, hvis man bevæger sig ordentligt på brættet, har muligheden for at dræbe ens modstander, uden den kommer i nærheden af en.

Det er dog igen et spørgsmål om balance, for hvis man har en klasse, som fokuserede på helbredelse, ville den kunne ignorere den langtrækkende klasse, og bare samle felter ind, imens ens modstander fokuserede på at angribe en. Når man så kom tilpas langt ned i liv, ville man kunne stå stille en runde eller to, og have regenereret langt mere liv, end modstanderen gav i skade. Dette ville føre til en klar sejr via resurser, hvis altså ens modstander kun fokuserede på at dræbe en. Hvis modstanderen derimod varierede sin rytme ved at samle resurser, og kun angribe engang imellem, så ville den i sidste ende vinde, da et angreb halverer modstanderens resurseindkomst i en runde.

Der er også nogle statistikker som passer bedre sammen end andre, og man skal derfor passe på når man laver en ny klasse, ikke at komme til og skabe en ubalance, ved at øge de statistikker for meget. Angrebstyrke og rækkevidde er to statistikker, som passer rigtig godt sammen, da det og kunne slå hårdt og langt, gør en overlegen i forbindelse med kampe. Derfor er det også oplagt, at hvis man laver en klasse, hvor man forstørrer rækkevidden, så mindsker man angrebsstyrken, og omvendt.

De to statistikker, helbred og helbredelse, passer også godt sammen, og har potentialet til at nullificere skaden fra ethvert angreb, ved at fokusere på resurser og helbred. En mærkbart længere rækkevidde hos modstanderen, vil dog altid kunne vinde en kamp imod en sådan klasse, hvis den spiller korrekt. Ud fra de statiskikker spillerne har på nuværende tidspunkt, er det ikke muligt at slå en klasse med markant længere rækkevidde, men ved tilføjelsen af andre statistikker, ville klasserne få en helt anden dimension

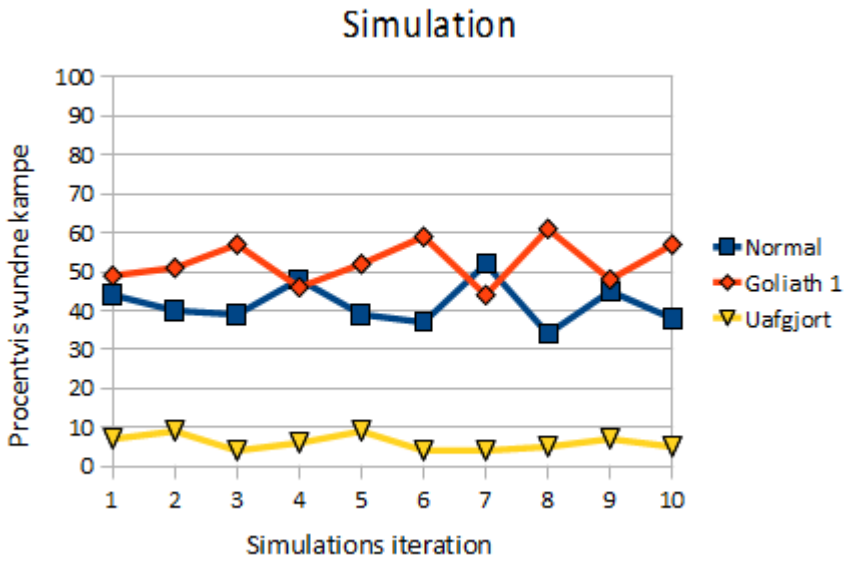


Figure 13: Simulation: Normal vs. Goliath 1

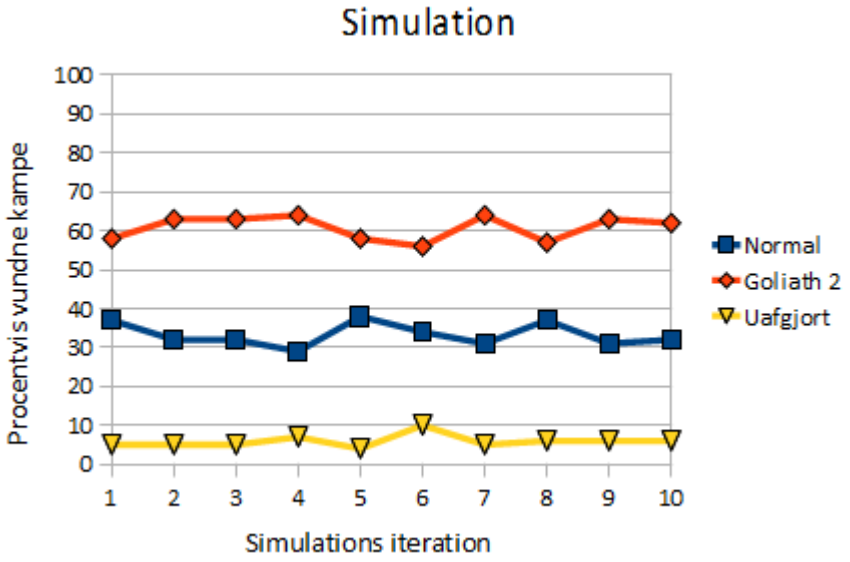


Figure 14: Simulation: Normal vs. Goliath 2

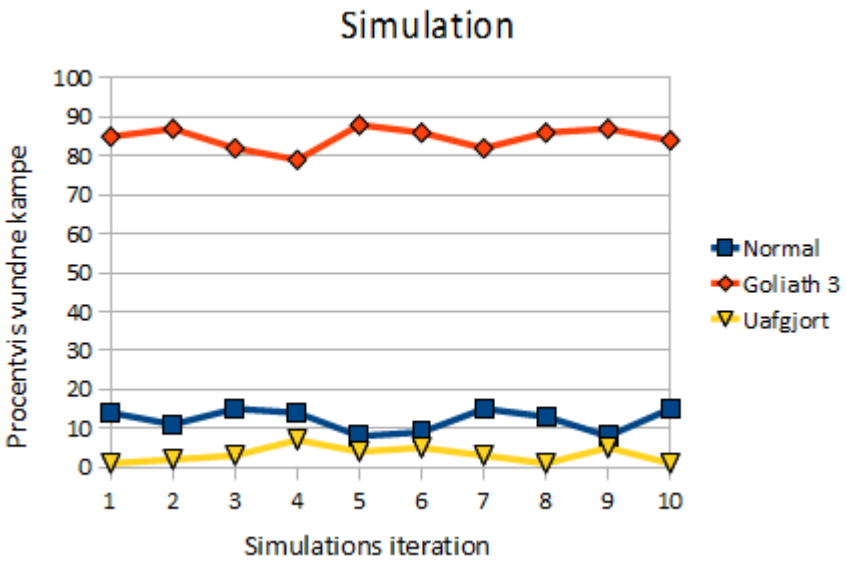


Figure 15: Simulation: Normal vs. Goliath 3

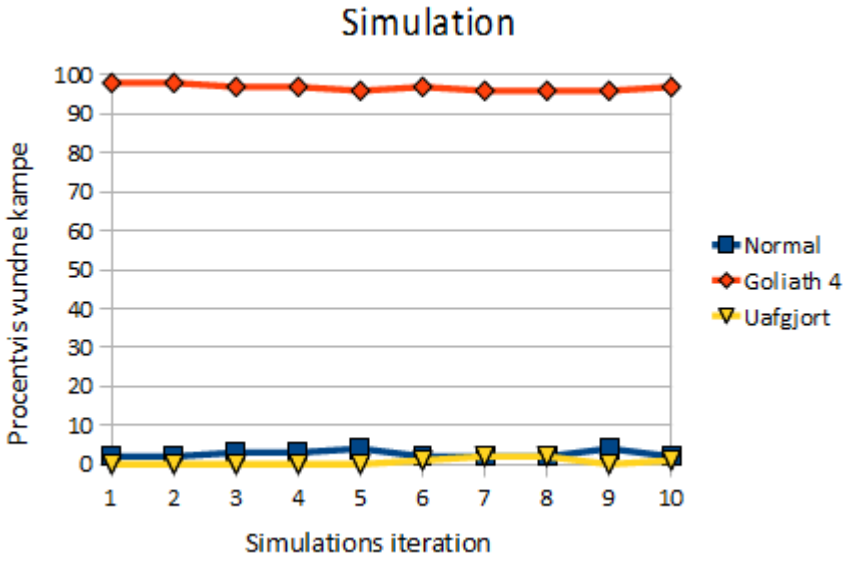


Figure 16: Simulation: Normal vs. Goliath 4

og dynamik.

Et eksempel på en sådan statistik, er hvor mange felter en klasse kan gå per tur. Allerede når klassen kan gå to felter på en tur, er det umuligt for den langtrækkende klasse at vinde, da man langt hurtigere får opbygget sig en mængde felter, som så udligner angrebets effekt på ens indkomst. Derudover, så kan man overtage to af modstanderens felter, og stadig kun miste halvdelen af ens indkomst den tur.

Grunden til at en sådan statistik ikke er blevet implementeret, er at den tydeligvis uretfærdig, uden en anden statestik til at balancere den. Derudover, så komplicerer flere statistikker også den opgave, at skabe en balance imellem klasserne. Flere statistikker gør dog opgaven, at lave flere forskellige klasser, nemmere, da der inden for hvert statistik findes en arketypeklasse. Det er klasser som "Fighter" og "Mage", som hver især har nogle høje værdier, inden for en statestik, men lave i andre.

Med andre ord, så giver mange statistikker en større variation og adspredelse i spillet, men det gør også spillet sværere at balancere og teste. I sidste ende handler om hvad man vil have ud af spillet, og vores spil er ikke beregnet til at skulle være sjovt at spille, eller have en lang levetid, men derimod er det lavet, så det er nemt at se balancen i spillet og imellem de forskellige klasser.

# 6 Diskussion

I dette afsnit vil diverse emner som "Designovervejelser", "Kunstige Intelligenser mod mennesker" samt "Bedre brug af Genetisk Programmering" blive diskuteret, og skal tænkes på som værende stof til eftertanke og som supplerende information og begrundelser for forskellige beslutninger der er taget i udviklingsforløbet.

## 6.1 Designovervejelser

For at skabe et godt spil, der kunne bruges til emnet, blev der undervejs taget mange designmæssige beslutninger for både at gøre det nemt at arbejde med, ikke for advanceret for os selv, og ikke have for mange forskellige ting som KIerne skulle overveje eller overveje. Dette fordelte sig primært inden for 4 overordnede emner: "Resurser", "Kamp", 'Defensivt træk" og "Assymetriske Parametre og Felttyper".

**Resurser:**
Måden resurser skulle optjenes var en af de ting der blev ændret mest, fordelt på tre overordnede beslutninger: hvad deres brug var, resurse-tyveri, og iht. modstanderens felter.

Først var spillet udformet således, at resurserne blev brugt på en måde der gav mere mening iht. navnet. spillerne startede med en mængde af dem, og brugte en variende mængde bestemt udfra om de ville angribe, bevæge sig, eller stå stille. Sidstnævnte var måden spilleren kunne optjene dem, ud fra hvor mange felter han ejede. Aspektet med antallet af ejede felter var dog det eneste der forblev som det var, da resurserne blev lavet til en værdi der kun kunne stige eller forblive konstant.

Ideen med at resurser kunne tages fra den anden spiller, resulterede desværre i at et spil potentielt kunne vare for evigt, ved at spillet ikke garanterede at der ville blive en vinder. Logikken i den nuværende version, er at hvis der ikke bliver givet resurser, bliver der mistet liv. For at helbrede liv bliver der givet resurser til en anden af spillerne. Før eller siden vil resultatet således blive at en af spillerne dør eller når maksimalt antal resurser. Resurse-tyveri havde også en risiko at gøre angreb for stærkt.

Mængden af resurse-indkomst blev også ændret, da det hurtigt blev tydeligt at det var alt for effektivt at tage modstanderens felter, fremfor at tage et der ikke var ejet af nogen. På tidspunktet hvor resurser også kunne bruges, kostede det således at tage en modstanders felt. Da dette blev afskaffet, blev det istedet gjort sådan at mængden af indkommende resurser blev halveret hvis man stjal et felt fra den anden spiller.

**Kamp:**
I starten var hensigten at kamp mellem de to spillere skulle være udformet som "Prisoner's Dilemma", hvor "Confess" og "Not Confess" svarede til to forskellig typer angreb. Dette var desuden en af grundene til at spillet blev lavet som simultantræk. Det viste sig dog at dette dels satte meget større krav til den kunstige intelligens, ved at spillet i teorien kunne afgøres udfra hvordan beslutningen heri blev taget, og ikke afhang af den egentlige strategi. Selvom "Prisoner's Dilemma" er balanceret [Kleinberg and Tardos, 2006], fungerede det bare ikke rigtig på den måde det blev benyttet. Således blev der blot lavet "Angreb". En anden fordel, er at der derved også er mindre træk der skal tages højde for.

Måden vi derefter valgte at balance det, var udfra tanken om at det at angribe skulle give en fordel til angriberen, udover "bare" at give skade, da spilleren i den tur således ikke ville have mulighed for at udvidde sit territorium med et felt mere. På den måde får en angrebet spiller enten ingen eller et lavere antal resurser i den tur.

**Defensivt træk:**
I og med kamp blev fuldstændig ændret, ønskede vi også at der skulle være en mulighed for at forsvare sig. Det defensive træk - at stå stille - skulle helbrede liv, hvis man havde mistet noget. Dette betød dog også at dette træk kun kunne udføres, hvis man enten var skadet, eller hvis modstanderen var ingen for rækkevidde, da der ellers ikke var noget, som ikke kunne gøres bedre ved et andet træk. Da spilleren ville bruge tid på at helbrede sigselv, vil resurse-indkomsten også være halveret, præcis som når man ville tage en modstanders felt.

**Assymetriske Parametre og Felttyper:**
Mængden af assymetriske parametre, blev også overvejet udvidet, specielt med blik på to: `resourcePerField` samt `movedPerTurn`. Fælles for disse var dog at de, på trods af at tilføje en masse sjovt til spillet, højst sandsynligvis også nemt ville kunne gøre det ubalanceret.

På samme måde var der også muligheder i at lave forskellige felttyper og give felter egenskaber derefter. Fx. kunne et felt være vand, der ville tage ekstra tid at krydse/overtage, men alternativt give flere resurser per tur.

Spillet ville sagtens kunne understøtte begge typer af tiltag uden for store tilføjelser, men KISS[8]-mentaliteten gjorde dog i sidste ende at de ikke kom med.

[8] Keep It Simple, Stupid.

## 6.2 Kunstig Intelligens mod Mennesker

En anden ting der er værd at overveje, når der arbejdes med kunstige intelligenser, i henhold til spilbalance, er det at balance dem med en menneskelig spiller. Af gode grunde har vi fokuseret på at balancere disse med hinanden, istedet for at være god nok til potentielt at slå en menneskelig spiller. To spillere, der har samme udgangspunkt og opfattelsesevner i et balanceret spil, må af gode grunde være lige gode.

Problemet med at en KI som regel ikke er så smart som et rigtigt menneske, er dog en ting der i mange spil oftest bliver "løst" ved brugen af det "David og Goliath"-princip, som blev benyttet i vores test-afsnit. Man sætter i dette tilfælde en bedre strategisk spiller mod en der *potentielt* er stærkere, men er svagere strategisk, hvorved den menneskelige spiller stadig bliver udfordret. Denne styrke ses, i computerspil, i mange forskellige afskygninger, som fx. hurtigere produktion af enheder eller større slagkraft.

# 7 Konklusion

Målet med opgaven var at få besvaret spørgsmålet "I hvor høj grad er det muligt at afgøre om et assymetrisk spil er balanceret?". Dette skulle gøres ved hjælp af et spil, kunstige intelligenser, begge lavet helt fra bunden.

Spillet blev færdiggjort, men delene omkring de kunstige intelligenser blev ikke gjort så godt som vi havde håbet at de kunne, da vi havde undervurderet præcis hvor avanceret en kunstig intelligens egentlig er, selv til et simpelt spil. I synopsen blev der under sektionen Risikovurdering skrevet:

> *"Om programmering af spillet bliver for overvældende, og om der er en god platform".*

Det er nu tydeligt, at der var for meget at programmere, i forhold til egne evner. Vi mener dog at meget at dette kunne forbedres hvis den Genetiske Programmering blev forbedret, og Minimax udvidet til at inkludere modstanderens træk i sine overvejelser.

På trods af dette, fik vi lavet et helt spil fra bunden, som understøtter mange forskellige spilklasser, metoder at simulere spil ved brug af kunstige intelligenser, gense spil under kontrolerede forhold, og stadig have store muligheder for udvidelse af basespillet. **Python** viste sig også at være en god platform at lave dette med, dels på grund af sprogets fleksibilitet, og dels på grund af den store mængde brugbare biblioteker der kunne hjælpe os i det vi prøvede at opnå.

Derudover understøttede testresultaterne vores udkast til balance mellem assymetriske spillerklasser.

Ud fra alt dette kan vi konkludere, at det i høj grad er muligt at vurdere om et spil med assymetriske parametre er balanceret, men at dette bliver gradvist sværere, jo flere parametre der kommer i spil.

# 8 Litteratur- og Kildeliste

Det følgende er en liste over materialer der er blevet refereret til undervejs i rapporten.

## Referencer

[Affenzeiler et al., 2009] Affenzeiler, M., Wagner, S., Winkler, S., and Beham, A. (2009). *Genetic algorithms and Genetic programming: Modern concepts and practical applications*. Chapman & Hall/CRC, numerical insights: 6 edition. ISBN-13: 9781584886297.

[Diplomacy Online, 2010] Diplomacy Online (2010). Online udgave af Diplomacy. `http://www.playdiplomacy.com/help.php`.

[Kleinberg and Tardos, 2006] Kleinberg, J. and Tardos, E. (2006). *Algorithm Design*. Addison-Wesley, Pearson International edition.

[Kolokoltsov and Malafeyev, 2010] Kolokoltsov, V. N. and Malafeyev, O. A. (2010). *Understanding Game Theory: Introduction to the Analysis of Many Agent Systems with Competition and Cooperation*. Singapore: World Scientific. ISBN-13: 9789814291712.

[Nobel Prize, 1994] Nobel Prize (1994). The Sveriges Riksbank Prize in Economic Sciences 1994. `http://nobelprize.org/nobel_prizes/economics/laureates/1994/`.

[Your Dictionary, 2010] Your Dictionary (2010). Definition af “balance”. `http://www.yourdictionary.com/balance`.

# A Appendix

## A.1 Simulations resultater

I den følgende tabel kan de gennemsnitlige resultater af samtlige simulationer, med 1000 spil hver.

Yderligere detaljer kan findes i vedlagte `testresultater.xls`-fil, hvor grafernes samt deres data er inskrevet. Denne danner således grundlag for de gennemsnitsværdier der er at finde i tabellen.

| Kamp og antal | Spiller 1 | Spiller 2 | Uafgjorte |
|---|---|---|---|
| **Normal vs. Normal** | $48,7\%$ | $46,7\%$ | $4,6\%$ |
| **Normal vs. Mage** | $45,4\%$ | $48,0\%$ | $6,6\%$ |
| **Normal vs. Healer** | $43,9\%$ | $50,8\%$ | $5,3\%$ |
| **Normal vs. Fighter** | $49,0\%$ | $46,3\%$ | $4,7\%$ |
| **Mage vs. Healer** | $50,0\%$ | $45,1\%$ | $4,9\%$ |
| **Mage vs. Fighter** | $51,0\%$ | $44,4\%$ | $4,6\%$ |
| **Healer vs. Fighter** | $46,2\%$ | $47,8\%$ | $6,0\%$ |
| **Normal vs. Goliath 1** | $41,6\%$ | $52,4\%$ | $6,0\%$ |
| **Normal vs. Goliath 2** | $33,3\%$ | $60,8\%$ | $5,9\%$ |
| **Normal vs. Goliath 3** | $12,2\%$ | $84,6\%$ | $3,2\%$ |
| **Normal vs. Goliath 4** | $2,6\%$ | $96,8\%$ | $0,6\%$ |

Table 6: Gennemsnitlige udfald af simulationerne.

## A.2 Bilag

Bilagene inkluderer det følgende, i nævnte rækkefølge:

- Liste af filer i projektet - 1 side
- Synopsis - 6 sider

## Liste af filer i projektet

### Rapport

- `rapport.pdf` - Denne fil
- `synopsis.pdf` - Synopsis
- `testresultater.xls` - Testresultaterne mere detaljeret

### Applikationsfiler

- `AIstats.py` - KI Statistikker
- `Board.py` - Spillebrættet
- `Draw.py` - Tegne-modulet
- `Field.py` - Felterne på spillebrættet
- `Game.py` - Selve spil-metoderne der samler spillet
- `Genetic.py` - Genetisk progammerings-modulet
- `Logger.py` - Logger-modulet
- `Minimax.py` - Minimax-modulet
- `Play.py` - Eksekveringsfilen for simulationsdelen af spillet
- `Player.py` - Spiller-modulet
- `Repickle.py` - Modulet til at gense spil ud fra logfiler
- `Simulation.py` - Svarende til `Game.py` for simulationer
- `Statspane.py` - Statistik-panelet i den grafiske del af spillet
- `WildEquild.py` - Svarende til `Play.py` for den grafiske del

Emil Erik Hansen &
Michael Madsen

# Balancering af Brætspil

## 1 Problemformulering

Er det muligt at afgøre om et assymetrisk spil er balanceret?

## 2 Begrundelse

Uanset hvilken afskygning et spil har, så er det nødvendigt for det at have en form for balance, for at være spilbart. Det kan være et brætspil, kortspil, computerspil, en blanding mellem de tre eller noget helt fjerde.

Der er mange måder at opnå en balance i computerspil på, værende forskellige parametre, som, hvis de bliver ændret, skubbe balancen, eller en kunstig intelligens(KI fremover), der reagerer på forskellige måder, alt efter hvad sværhedsgrad den er blevet sat til. Derfor er det nødvendigt at definere begrebet "balance". Dette opfatter vi som værende den tilstand hvor begge (eller alle parter) af et spil som udgangspunkt har lige stor mulighed for at sejre, selvom deres udgangsmæssige parametre og/eller strategier er forskellige.

Vi har valgt projektet idet vi syntes at spilbalance med tiden er blevet et uhyre vigtigt aspekt af alle spil, samtidigt med at det kan afgøre et spils success meget hurtigt. Ingen vil spille noget der er ubalanceret, men det skal samtidigt være forskellige måder og afskyninger at spille på, samt meget andet. Samtidigt har den stigende online-verden gjort det muligt for spillerene at diskutere og gå i dybden med detaljerne, således at det at balancere er blevet endnu sværere.

Desuden mener vi at et spil af denne type vil kunne afspejle generel spilbalance rimeligt præcist, idet vi vil forsøge at få mange af de generelle grundlæggende balance-elementer med, og undgå diverse tilfældige faktorer.

## 3 Evaluering

For at bestemme om vores problemformulering kan løses, har vi valgt at opbygge et spil, hvor forskellige kunstige intelligenser prøver forskellige strategier af mod hinanden, for at påvise at der vil være balance - lige meget hvilken strategi, eller udgangspunkt de er i. Dette gøres ved brug af intensive tests og simuleringer.

# 4 Arbejdsopgaver

Disse skal selvfølgelig uddybes i fællesskab - men kort sagt er det jo det følgende:

1. Opstille en præcis og omfattende definition for hvad balance er.
2. Definere regler og parametre for et spil som vil kunne benyttes til at vise balance.
3. Udarbejde spillet (hvor K.I'erne sættes mod hinanden).
4. Opbygge forskellige K.I'er og strategier disse kan følge.
5. Intensive tests og modificeringer/analyse af strategier.
6. Skrive og gennemrette rapport.

# 5 Metoder

Til opgave 1, 2 og 3 vil vi tage udgangspunkt i spillet "Diplomacy" til vores design, da det har nogle basale elementer vi kan udnytte, samt litteratur om emnet.
Til opgave 3 har vi tænkt os at gøre brug at Python, da det er en nemt sprog at gå igang med- Vi lægger ud med at lave en hurtig prototype, så vi kan se om vi er på rette vej.
Til opgave 5 vil vi blandt andet bruge genetisk programmering til at teste strategierne.

# 6 Information og Informationskilder

Bøger vi bruger som inspiration til projektet:

- Understanding game theory : introduction to the analysis of many agent systems with competition and cooperation:
  - Forfatter: Vassili N Kolokoltsov, Oleg A Malafeyev
  - ISBN-13: 9789814291712
- Genetic algorithms and genetic programming: Modern concepts and practical applications:
  - Forfatter: Michael Affenzeiler
  - ISBN-13: 9781584886297

Vi vil blandt andet bruge disse internetsider som inspiration til projektet:

- Forelæsning af John F. Nash om spilteori:
  `http://www.math.princeton.edu/jfnj/texts_and_graphics/Main.Content/Presentation_on_Game_Theory/`
- Game Theory:
  `http://gametheory.net/`

# 7 Tidsplan

Arbejdsopgaver skrevet i parantes skal forståes som "så småt påbegyndt", "overvejende" eller "i tilfælde af mere arbejde".

| **Måned** | Sep. | Okt. | Nov. | Dec. | Jan. |
|---|---|---|---|---|---|
| **Arbejdsopgave** | $1, 2$ | $2, 3, (4), (6)$ | $3, 4, (5), 6$ | $5, 6$ | $(5), 6$ |

# 8 Afgrænsning

Vores vurdering af spilbalance-begrebet bliver vurderet ud fra en speciel type spil, som vores applikation også bliver en type af.

Vi vil desuden gøre brug af, og fokusere meget på, John F. Nashs forskning indenfor spilteori, specielt "Nash's Equilibrium". Vi vil dog ikke redegøre eller gå videre i dybden med disse.

Resultaterne af vores konklusioner er udelukkende baserede på vores tests og simuleringer fra spillet, ved brug af kunstige intelligenser og generisk programmering. Af denne grund vil vi heller ikke fokusere yderligere på kunstige intelligenser, men udelukkende spilbalance. Dette er gjort for samtidigt at gøre emnet mere håndgribeligt og ikke alt for omfattende.

# 9 Risikovurdering

Der er tre primære risici i henhold til udarbejdelsen af projektet:

1. Om der kan findes tilpasse strategier, og om disse fungerer mod hinanden.
2. Om programmeringen af spillet bliver for overvældende, og om der er valgt en god platform.
3. At tests og simuleringer ikke lykkes eller går som planlagt.

# 10 Disposition af Rapporten

Rapporten kommer til at have de følgende afsnit.

1. Indledning og Problemformulering
2. Afgrænsning
3. Definition af Balance
4. Et kig på forskningen / Spilbalance generelt
   (a) Spilbalancens milepæle
   (b) Nash's Equilibria
5. Beskrivelse af Spillet
   (a) Regler og Opbygning
   (b) Strategi-beskrivelser
6. Tests
   (a) Applikation
   (b) Strategier og Kunstig Intelligens
7. Diskussion
8. Konklussion