A A R H U S U N I V E R S I T E T

Eksamensopgave
Bacheloruddannelse i Informationsvidenskab (2006-studieordning)

Eksaminator: **Afleveringsdato:**

Censor (udfyldes af sekretariatet)**:**

Intro til programmering, intern best./ikke best.	☐	(sæt kryds)
Programmering og systemudvikling, intern 13-skala	☐	(sæt kryds)
Kommunikation-1, intern best./ikke best.	☐	(sæt kryds)
Kommunikation-2, ekstern 13-skala	☐	(sæt kryds)
Digital æstetik, intern 13-skala	☐	(sæt kryds)
Teknologihistorie , ekstern 13-skala	☐	(sæt kryds)
Organisationsanalyse, ekstern 13-skala	☐	(sæt kryds)
Studium generale, intern 13-skala	☐	(sæt kryds)
Informationsvidenskabelig metodet, intern best./ikke best.	☐	(sæt kryds)
Valgfrit projekt, intern 13-skala	☐	(sæt kryds)
Bachelorprojekt, ekstern 13-skala	☐	(sæt kryds)

Opgavens anslag: _____

Afleveret af:

Årskort: **Navn:**

Årskort: **Navn:**

Årskort: **Navn:**

Årskort: **Navn:**

Må eksaminators eksemplar af eksamensopgaven gøres til genstand for udlån? JA ☐ NEJ ☐

Studienævnet for Informations- og Medievidenskab, Helsingforsgade 14, 8200 Århus N

2007

28 Seconds Later

Mavinigruppen
Martin Grove
Villars Gimm
Nicki T. Hansen
05-06-2007

Indholdsfortegnelse

Vi har fordelt ansvaret på følgende måde: Martin (5-15), Villars (15-25), Nicki (25-35), og resten er fælles.

Antal tegn: 94.019

Indledning

Denne rapport omhandler vores eksamensprojekt i kurset Programmering og Systemudvikling foråret 2007. Rapporten er bygget op omkring den systemudviklingsproces, vi har haft i forbindelse med udviklingen af en simulation, der afspejler spredningen af en smitsom virus, der forvandler mennesker til zombier. Vi har lavet en visuel grafisk repræsentation af en by med bygninger, hvor zombier, soldater og mennesker interagerer med hinanden. Brugeren har i den grafiske brugergrænseflade mulighed for at ændre på en række parametre, som influerer på udfaldet i simulationen.

Vi valgte zombieprojektet af de tre udbudte projekter (hvor de to andre var hhv. et computerspil og en projektstyring), da vi i perioden op til projektets start netop havde udviklet et mindre computerspil, hvilket dermed udelukkede dette projekt, da vi følte at det ville være trivielt at lave det samme en gang til. Valget faldt herefter på simulationen, da vi i en kort brainstorm hurtigt fik en masse sjove idéer i forhold til bl.a. konstrueringen af kunstig intelligens, hvilket tiltalte os. Ud fra denne brainstorm vurderede vi, at der ville være omfattende muligheder for modellering og analyse, hvilket er hensigtsmæssigt i forhold til at opnå en afvekslende og meningsfuld iterativ proces.

Problemformulering

Vi vil i denne rapport redegøre for vores projektforløb af en fremstilling af en fungerende simulation af sygdomsspredning, samt reflektere over vores projektforløb, baseret på Craig Larmans systemudviklings-teorier.

Metode

Vi vil i denne rapport beskrive et iterativt udviklingsforløb af vores softwareprojekt, *28 Seconds Later*, som er en simulation af sygdomssmitte. Vi har gennem forløbet brugt bogen *Applying UML and Patterns*, som er skrevet af Craig Larman og beskriver en Agile Unified Process, hvor projektforløbet deles op i adskillige iterationer, hvorunder små ukomplette delsystemer udvikles. Vi har ved hjælp af disse iterationer fået belyst højrisikoområder igennem projektforløbet. Endvidere har vi gennem forløbet brugt og diskuteret Larmans forskellige artefakter, som har været en hjælp i arbejdsprocessen. For at nå frem til dette endelige produkt har vi gennemgået en kronologisk proces, som vi vil forklare i denne rapport med en redegørelse for de forskellige iterationer vi har delt forløbet op i samt en forklaring af de artefakter, vi har valgt at benytte. Vi har benyttet Larmans forskellige principper og mønster med henblik på at udvikle et godt objektorienteret design, hvor vi har tildelt de forskellige softwareobjekter det hensigtsmæssige ansvar. I de forskellige iterationer har vi forsøgt at veksle mellem henholdsvis analyse, hvor vi har foretaget de nødvendige undersøgelser af de forskellige dele af systemet, og design, hvor vi har forsøgt at finde konceptuelle løsninger på de forskellige problemstillinger. Derudover har vi løbene skrevet fungerende kode og testet denne i de forskellige iterationer.

Metoden indeholder mere konkret følgende punkter:

1. Vi vil først redegøre for, og præsentere, det endelige program, således at det er tydeliggjort, hvilket mål vores proces har først os til. Denne gennemgang bliver dog kun overfladisk, da de grundige detaljer bliver forklaret senere i rapporten.

2. Vi vil derefter lave en beskrivelse af forløbet, hvor vi redegør for de forskellige iterationer vi har været igennem samt beskriver de forskellige overvejelser vi har haft i arbejdsprocessen.
3. Dernæst vil vi beskrive de forskellige artefakter vi har anvendt, og hvordan de har hjulpet os, samt sammenhængen imellem disse.
4. Følgende vil vi forklare de mest centrale valg og optimeringer, vi har foretaget i henhold til implementeringen af kode. I denne forbindelse vil vi belyse, hvordan vi har anvendt forskellige designmønstre og hvordan vi har løst diverse ydelsesmæssige problemer.
5. Vi vil ydermere lave en refleksion over projektforløbet, hvor vi vil overvejer nærmere, hvordan vi har grebet opgaven an, og hvilke fordele og ulemper iterationsstrukturen har medført.
6. Vi vil endvidere reflektere over de forskellige artefakter, vi har benyttet, og hvilke spekulationer vi har haft i forbindelse med dette.
7. Afslutningsvis vil vi lave en opsamling og konklusion på hele projektet.

Beskrivelse af simulationen

Strukturen

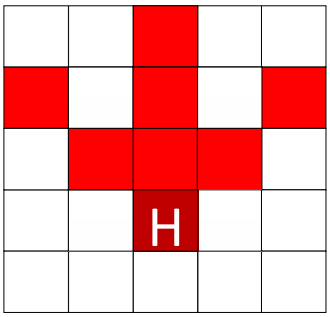
Projektet er todelt i en opsætningsdel og selve simulationen. Opsætningsdelen tager informationer fra brugeren, såsom antal mennesker, bygninger osv. og starter derefter simulationen i et andet vindue. Simulationen er grafisk repræsenteret som en kvadratisk bane, som ses ovenfra, hvorved det størst mulige overblik, og nemmest mulige implementering af visuel repræsentation, opnås. Bygningerne kan forekomme i forskellige størrelser, og det er ikke muligt for aktørerne at gå disse områder af simulationen. Vi opererer med tre forskellige overordnede typer af væsener – en *zombie*, en *soldat* og et *menneske* – og overordnet kaldes disse *aktører*. Alle disse aktører er illustreret i simulationen ved brugen af følgende farver af de enkelte pixels.

Zombie: Grøn
Død zombie: Mørkegrøn
Menneske: Rød
Menneske i panik: Gul
Soldat: Blå

Aktører

Menneske

Mennesker vælger en tilfældig retning og går tyve pixels i denne, hvorved deres mål er defineret. Menneskers synsfelt er defineret ved, at de kan se i de tre fremadrettede retninger, som illustreret i figuren ved siden af, som har sin primære retning mod nord, og derfor kan se nord, nordøst og nordvest. En menneske kan se i denne retning i 10 felter frem, med mindre der står noget i vejen, hvilket vil sige et objekt der ikke er et *empty* objekt.



Menneskers mål annulleres, hvis de ser et felt direkte i bevægelsesretningen, hvis tilstand er alt andet end *empty*. Derudover

ændrer mennesker tilstand, hvis de ser en zombie, hvilket medfører, at de bliver til et nyt objekt af typen *humanPanic* – et menneske i panik. Menneskene reagerer ikke på døde zombier, da disse ikke smitter længere.

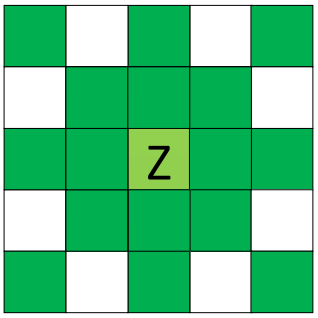
Menneske i panik

Et menneske i panik har en bevægelseshastighed på tre gange så meget som et almindeligt menneske, og bevæger sig direkte væk fra den zombie som forsagede panikken, i 50 ryk. De kan derudover kun se et enkelt felt frem, hvilket simulerer, at menneskene mister deres evne til at orientere sig. Hvis der dog står noget i vejen for disse, forsøger objektet at bevæge sig i den nærmeste retning som ikke er den nuværende. Dette betyder at et menneske i panik kan afbøje sin retning i uforudsete, og for den selv, uheldige retninger (ind i zombier). De 50 ture i panik repræsenterer en paniktilstand, hvorefter mennesket falder tilbage i sin almindelige tilstand (human), og bevæger sig imod et nyt mål i en tilfældig retning.

Et menneske i panik overfører ikke panik til andre mennesker, da vi testede med dette, og det medførte at alle mennesker var i panik konstant. Implementering af dette er dog relativ simpel, og er repræsenteret som *Paranoia* i vores opsætningsdel. Pga. tidsmæssige deadlines er denne feature dog ikke aktiveret, og derfor gemt til en eventuel udvidelsespakke.

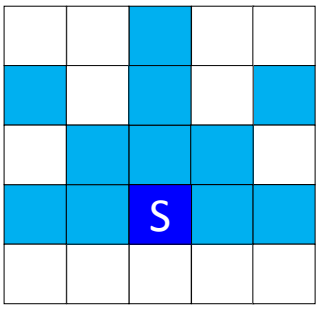
Zombie

Zombier vælger hele tiden en tilfældig retning at bevæge sig i, men kigger kontant fem felter omkring sig i alle retninger. Hvis de ser et objekt der kan smittes, bevæger de sig i den retning. Når en zombie støder ind i et statisk objekt eller en *edge*, skifter disse retning til en vilkårlig retning. Hvis de ser et menneske eller en soldat i deres synsfelt, som alle otte retninger i fem felters rækkevidde, som illustreret til højre, bevæger de sig imod denne aktør. Zombier smitter *alle* soldater og mennesker, som befinder sig i tilstødende felter til det specifikke felt, som den givne zombie befinder sig i, hvilket vil sige, at de potentielt set kan smitte otte aktører pr. tur. Zombier har en prædefineret levetid på 190 ture, og når de smitter får de tildelt deres oprindelige levetid igen, hvilket simulerer, at de får liv af at spise lidt af vedkommende de smitter. Zombier kan ydermere ændre tilstand til et *deadZombie* objekt, hvilket simulerer dets død, og de er herefter blot en forhindring for aktørerne i simulationen. Dette forekommer, når de bliver skudt af en soldat eller ikke har smittet et menneske i det prædefinerede antal ture.



Soldat

Soldater vælger en tilfældig retning og går ligesom mennesker efter et mål, men de kan se noget længere, nemlig femten felter, hvilket simulerer, at de er mere opmærksomme end mennesker. Deres mål defineres ved det felt, som befinder sig tredve felter væk i soldatens tilfældige valgte bevægelsesretning. Soldater har tilmed et større synsfelt end mennesker, da to yderligere retninger er tilføjet, hvilket er illustreret til højre. Soldater går ligesom zombier helt hen til et givent statisk objekt eller en *edge,* før de vælger en ny retning, da de ikke er bange for at blive trængt op i en krog. Soldater bevæger sig, ved opdagelsen af en zombie, mod denne og forsøger derved at komme ind på skudafstand af zombien. Når soldaterne kommer indenfor denne afstand stopper de op for at holde afstand til zombien, og de prøver derved at



undgå at blive smittet. Soldater kan kun skyde én zombie pr. tur, hvilket udligner det forspring, som skudrækkevidden medfører. Soldater forsøger ikke at flygte fra zombier, men bliver stående så længe, der er zombier indenfor skudrækkevidde.

Tur-baseret

Simulation er tur-baseret, hvilket vil sige, at en tur består af en bevægelse og en handling for de respektive objekter. Alle objekter bevæger sig asynkront, hvilket betyder, at hvert objekt finder sin retning og bevæger sig én efter én. Billedet opdateres efter hver tur således det ser ud som om det er realtid, hvilket bevirker, at simulationen virker mere realistisk. Denne opdeling i ture bruger vi ydermere i forbindelse med definitionen af hastighed for de forskellige væsener, da vi ikke tillader alle aktører at foretage en bevægelse hver gang systemet opdateres.

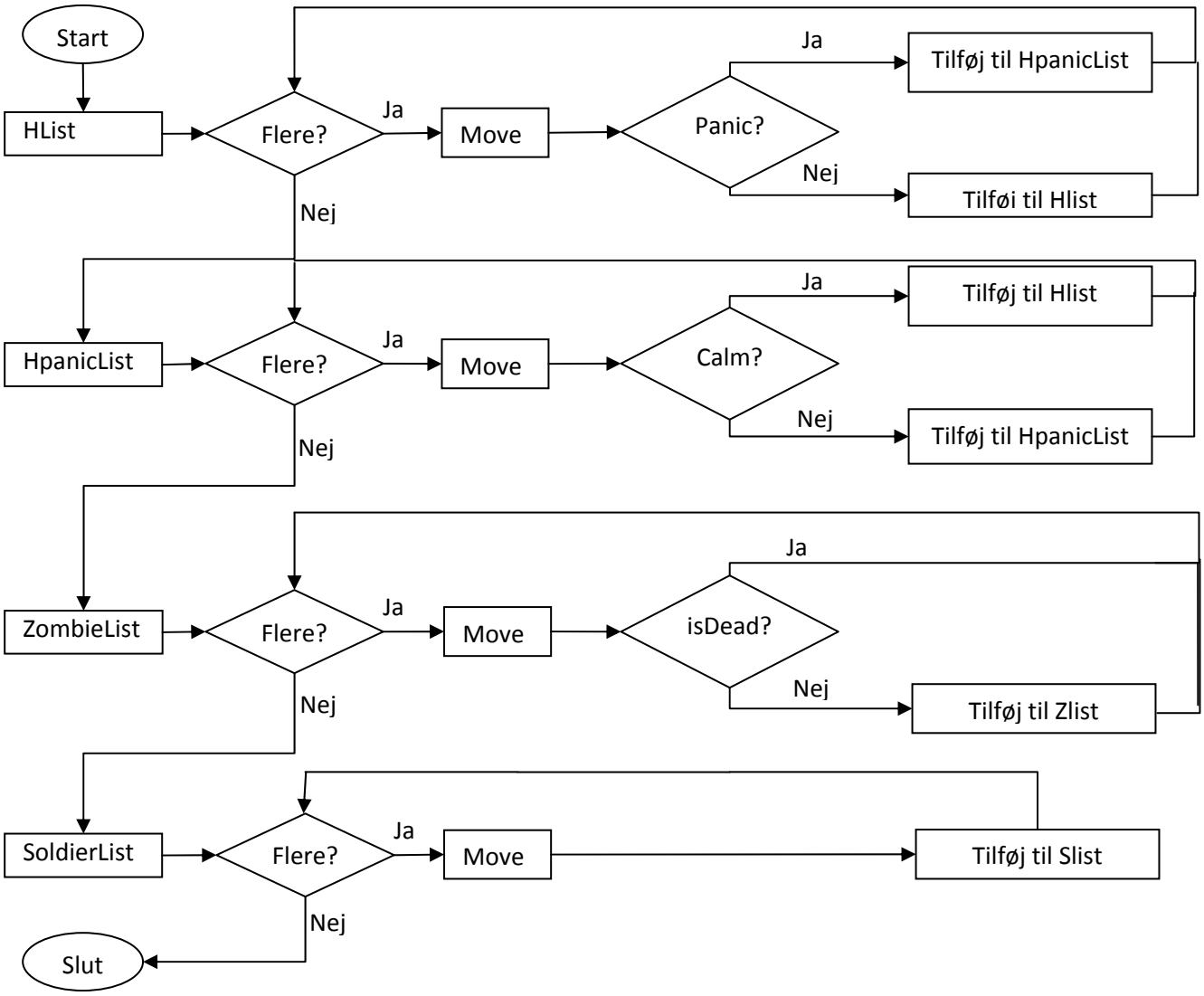
Bevægelse

Systemet indeholder fire lister med de forskellige typer bevægelige objekter (human, humanPanic, zombie, soldier). For hver aktør i hver liste, henter systemet detaljer såsom synslængde og synsretninger, ved hvilke systemet udregner hvad det givne objekt kan ”se” ud fra de begrænsninger vi hat tildelt det. Dette bliver sendt tilbage til objektet, som så ved brug af dets interne kunstige intelligens udregner den bedst mulig retning at gå i. Denne retning bliver så returneret til systemet, som bevæger objektet hen på sin nye position i simulationen. Bevægelserne bliver taget i tur af de forskellige typer i følgende rækkefølge.

1. Mennesker (human)
2. Mennesker i panik (humanPanic)
3. Zombier (zombie)
4. Soldater (soldier)

De tre øverste har tilstande, der kan ændres efter de har rykket, hvilket bliver efterset på det pågældende objekt, der har bevæget sig. Disse tilstandsændringer er følgende:

- Mennesker kan gå i panik (human → humanPanic).
- Mennesker i panik kan slappe af igen (humanPanic → human).
- Zombier kan dø pga. deres begrænsede levetid (zombie → deadZombie).



Som allerede nævnt er der en prioriteret rækkefølge i forbindelse med en bevægelse, som ovenstående *flowchart* illustrerer. Vi benytter fire forskellige *ArrayList* til at holde styr på henholdsvis mennesker, mennesker i panik, zombier og soldater. Disse lister gennemløbes igennem asynkront indtil, der ikke er flere aktører i den pågældende liste, hvorefter næste liste i vores prioritering aktiveres. *HumanList*, som indeholder alle menneskeobjekter i simulationen, gennemløbes som den første, og der undersøges om det enkelte menneske har set en zombie, hvilket resulterer i en paniktilstand og objektet tilføjes til listen med paniske mennesker - *HumanPanicList*. Hvis mennesket ikke har set en zombie tilføjes det til *Humanlisten.* Der foretages en tilsvarende undersøgelse for mennesker, som allerede befinder sig i denne paniske tilstand, når de skal bevæge sig. Herefter gennemløbes listen over zombier, og der undersøges om zombierne er døde efter deres bevægelse. I tilfælde af dette glemmes zombier, hvilket vil sige, at modellen ikke længere læser disse objekter, og listen, som indeholder levende zombier, bliver derved reduceret, hvilket resulterer i, at simulationens hastighed forøges efterhånden som zombierne dør. Efterfølgende gennemløbes listen med soldater, som alle tilføjes til deres respektive liste, da soldater kun har en tilstand.

Hastighed

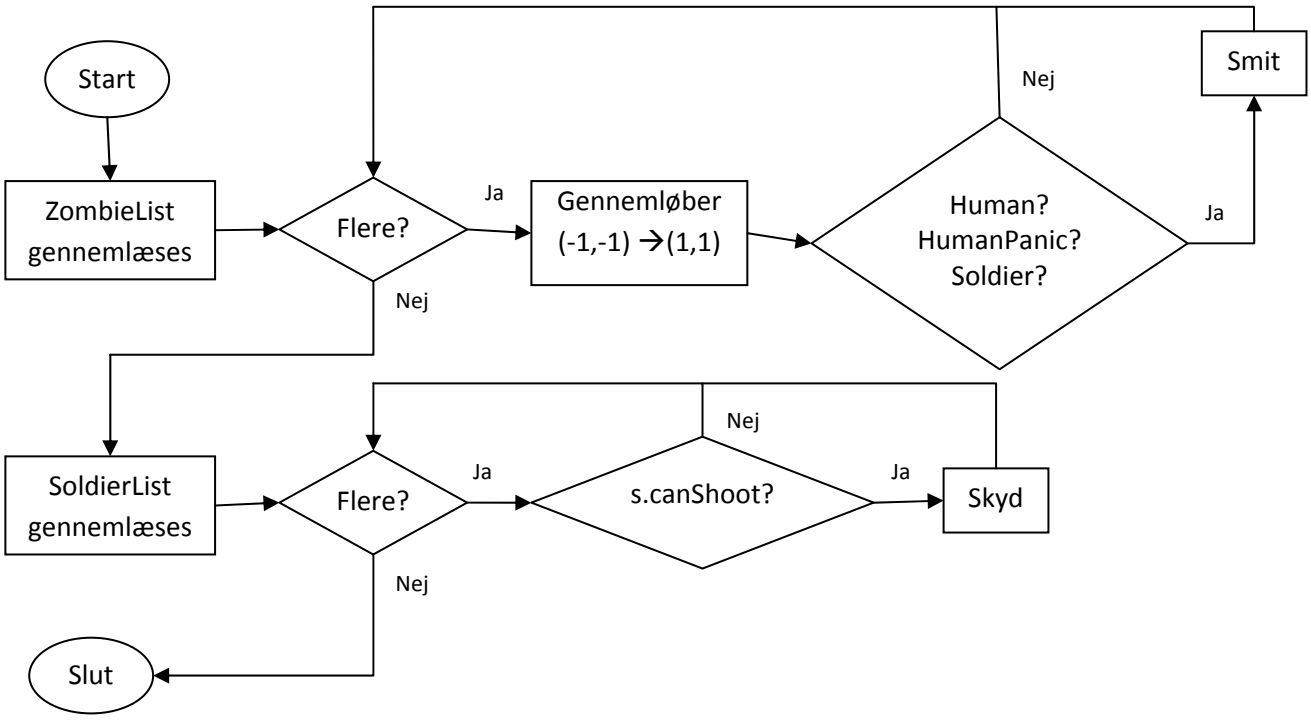
Vi har defineret hastighed ved, at de forskellige aktører laver en bevægelse på forskellige ture. Det vil sige, at simulationen er defineret til at skulle opdatere så hurtigt som det hurtigste objekt på banen.

I vores opgaveformulering er denne hastighed relativt defineret ved, at zombier skal bevæge sig med halv hastighed af mennesker, og mennesker i panik skal bevæge sig med tre gange hastighed af mennesker. Dette kan oversættes til følgende hastigheder: zombie = 1, human=2, humanPanic=6. Vi har valgt, at lade simulationen opdatere sådan at det hurtigste objekt skal bevæge sig hver tur, og langsommere objekter skal bevæge sig på ture der svarer til deres hastighed. Det vil sige at vi definerer en ventetid i ture, hvorefter en given aktør skal bevæge sig. Dette betyder, at følgende ventetider er gældende: humanPanic=0, human=2, zombie=5.

Denne metode overholder den opgaveformulering vi har fået stillet, men der er dog visse realisme-problemer med dette, da alle zombier vil bevæge sig i samme tur, og alle mennesker på samme måde. Dette har vi dog løst ved, at definere den *første* ventetid i simulationen til et vilkårligt tal imellem 0 og deres interne ventetid. De forskellige aktører foretager disse bevægelser asynkront, hvorved en mere dynamisk og realistisk effekt af real-tid simuleres.[1]

Handling

En soldat og en zombie har begge mulighed for at lave en handling. Dette er illustreret af nedenstående flowchart.



Vi har implementeret en liste for henholdsvis zombier og soldater, som løbes igennem i viste rækkefølge hver gang en handling udføres. Først undersøges om, der er flere zombier i listen, hvorefter de otte

[1] Se bilag XXIV

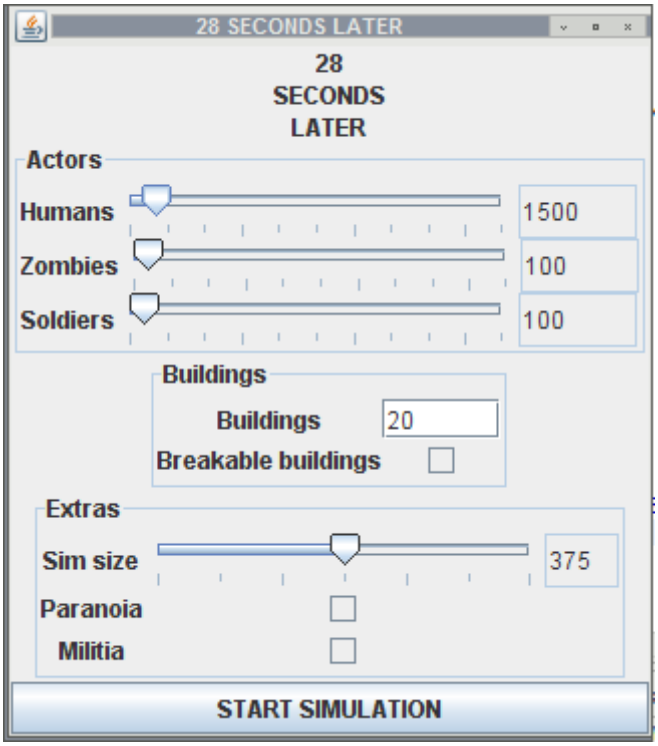
omkringliggende felter, vha. relative feltkoordinater, gennemløbes, i tilfælde af at listen stadig indeholder minimum en zombie. De respektive feltkoordinater undersøges i denne forbindelse for eventuelle soldater, mennesker og mennesker i panik, og zombien smitter alle de aktører, som befinder sig på disse koordinater. Hermed er en handling for en given zombie gennemført, og dette gentages indtil listen er tom. Herefter løbes listen med soldater tilsvarende igennem. Soldaterne har tidligere bemærket om de kan skyde en zombie, og har gemt retning og afstand til denne. Hvis de kan, så skyder de zombien. Når listen med soldater er tom er én handling for alle for alle zombier og soldater i simulationen foretaget.

Bygninger

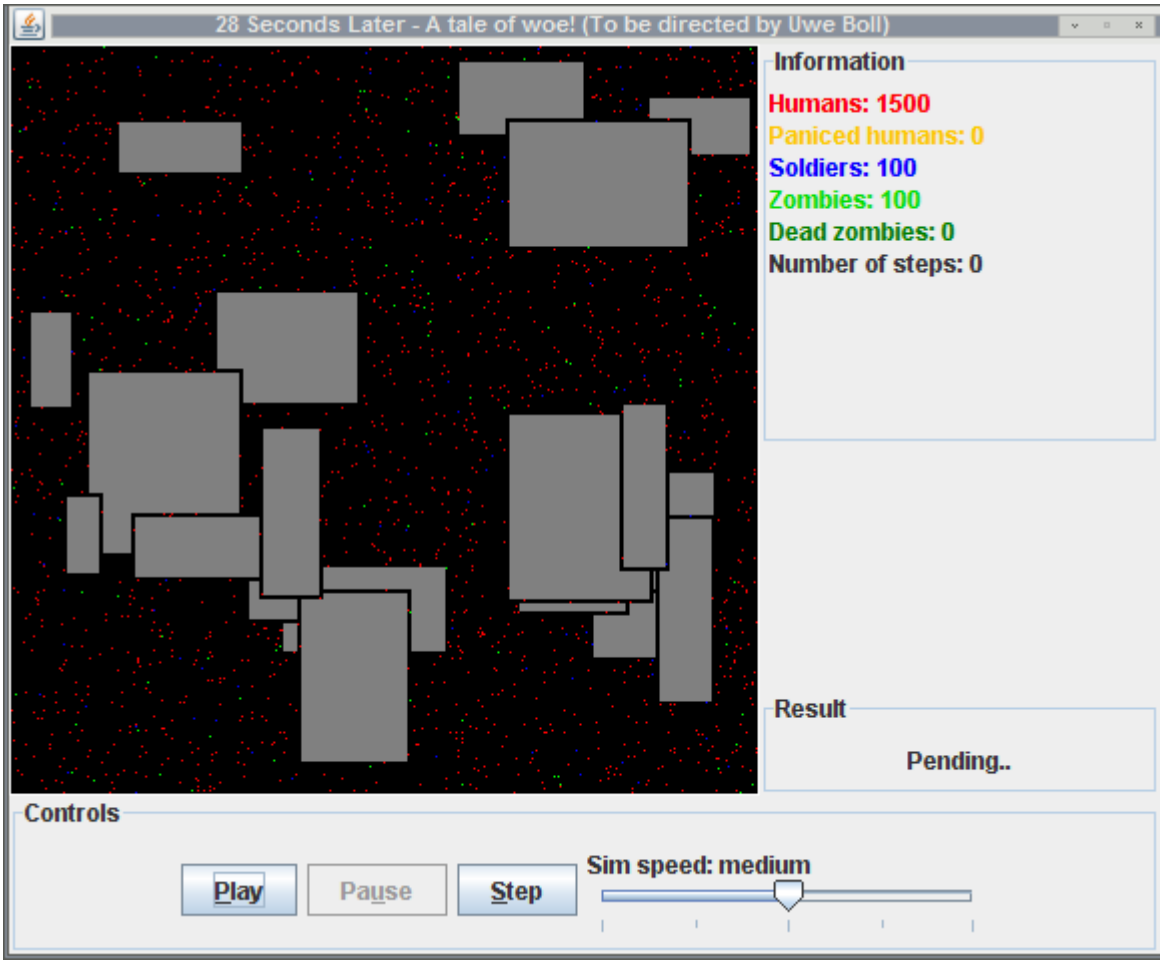
En bygning er defineret som et felt, der ikke kan gennemtrænges. Det vil sige, der kan ikke gås på det felt, og der kan ikke se igennem feltet. Bygninger varierer i størrelse indenfor (20 – 100 x 20 – 100) felter, og bliver placeret tilfældigt. For at undgå problemer med indelukkede aktører, har vi defineret hver bygning til at have en ramme af tomme felter om sig, der er to felter bred. Dette udelukker de fleste tilfælde af indelukkede aktører, hvor det dog stadig kan opstår hvis en lille bygning placere fuldstændigt indenfor en større bygning.

GUI

Vores grafiske brugergrænseflade er, som nævnt, todelt i en opsætningsdel og en simulationsdel. Opsætningsdelen, som er illustreret nedenfor, består af en række forskellige parametre, som det er muligt for brugeren at indstille forud for populeringen af vores simulation. Vi har valgt, at benytte *sliders* i forbindelse med, at en given bruger skal vælge antallet af de forskellige aktører samt angive størrelsen af selve simulationens brugergrænseflade. Derudover er det muligt for brugeren at bestemme, hvor mange bygninger, der skal populeres, hvilket skal indskrives. Ydermere har vi valgt at antyde, hvilke intentioner vi har haft i forhold til ekstra features i form af de potentielle markeringsmuligheder; *Breakable buildings, Paranoia og Militia.* Disse idéer vil blive gennemgået i afsnittet om vores overvejelser. Grunden til, at vi implementeret disse overvejelser i vores GUI er, at vi basalt set har datastrukturerne, som forudsætter funktionaliteten af disse, men vi af tidsmæssige årsager ikke nåede at lave udvidelserne. Dette data vil blive uddybet yderligere i afsnittet om implementeringen af koden.



Simulationsdelen har vi valgt at dele ind i fire forskellige overordnede dele bestående af henholdsvis information, resultater, kontrollerbare indstillinger og selve simulationen. I informationsdelen angives antallet af de forskellige aktører, samt de tilstande disse kan have. Derudover synliggøres det aktuelle antal af ture, som er foretaget. Under denne del printes udfaldet af simulationen ud, hvorfor den benævnes resultatdelen. Til venstre for denne forefindes kontroldelen, hvor brugeren har mulighed for at starte og pause simulationen. Endvidere er det muligt at få vist simulationen én tur af gangen, hvilket øjensynliggør en detaljeret fremvisning af udviklingen. Der er tilmed fem forskellige niveauer af hastigheden, som kan justeres af brugeren. Ovenstående denne del vises selve simulationen, hvor de grå firkanter illustrerer bygninger, og farveforskellene på de forskellige pixels markerer de forskellige aktører.



Beskrivelse af forløbet

I dette afsnit vil vi beskrive forløbet af vores systemudviklingsproces og mere konkret, hvad vi har udviklet i de forskellige faser, som er baseret på teorierne i bogen *Applying UML and Patterns.* Vi har gennemgået en inceptionsfase samt fire iterationer i elaborationsfasen, og vi vil beskrive de mest centrale dele af udviklingen af systemet samt vores arbejdsprocesser og metoder i disse. I dette afsnit vil vi nævne flere Unified Process artefakter, som ikke med det samme bliver forklaret. Disse forklaringer og udspecificeringer gemmes til det nedenstående afsnit *Beskrivelse af Unified Process artefakter*.

Inception

I denne fase foretages en grundlæggende undersøgelse af projektet, hvor målet er at konkludere, hvorvidt det er hensigtsmæssigt at fortsætte med dette projekt. I vores tilfælde kunne vi overordnet vælge imellem henholdsvis en simulation, en projektstyring og et computerspil. Vi lavede indledende en overordnet brainstorm for at få et overblik over opgavens omfang og fastlagde de grundlæggende idéer for zombiesimulationen. Vi blev på forhånd tildelt en *vision* i form af opgaveformuleringen, og vi forsøgte i denne fase at uddybe denne foreløbige *vision*.

Vi fastlagde de basale krav, der var for simulationen og besluttede hvilke aktører og elementer simulationen foreløbigt skulle indeholde. Vi besluttede, der skulle være tre typer væsener – mennesker, soldater og zombier. Disse skulle interagere i en modelleret verden bestående af en række bygninger. Der skulle være mulighed for at pause simulationen og genoptage den senere og antallet af de forskellige væsner skulle kunne angives ved start af simulationen. Endvidere skulle hastigheden af simulationen samt zombiernes levetid kunne angives. Vi opstillede to potentielle slutscenarier – enten vinder soldaterne ellers vinder zombierne. Hvis zombierne vinder indebærer det, at alle soldater og mennesker er blevet smittede. I det andet scenarie er zombierne enten blevet dræbt af soldaterne eller døde pga. af at deres levetid er udløbet.

Vi påbegyndte i denne fase en række artefakter, som videreførtes til de senere iterationer i elaborationsfasen. Vi lavede i fællesskab en skitse af domænemodellen og belyste de mest centrale underfunktioner i programmet vha. nogle få *brugsmønstre*, som er oversat fra det engelske ord, *Use-Case*. Vores udkast til domænemodellen medførte et overblik, som vi kunne bruge i forbindelse forståelsen af sammenhængen i systemet, hvilket var grunden til, at vi allerede i inceptionfasen overfladisk påbegyndte dette artefakt. De forskellige brugsmønstre fungerede som en beskrivelse af de forskellige aktørers handlingerne. Vi valgte, at begrænse antallet af *fully-dressed* brugsmønstre til dem, som var beskrivende for det de mest centrale dele af vores vision. Vi fik ud fra disse brugsmønstre en idé om kompleksiteten samt omfanget af brugsmønstrene for projektet. Udover påbegyndelse af disse artefakter lavede vi tilmed et udkast til et *glossary,* som vi i denne rapport har valgt at oversætte til gloseliste. Denne brugte vi til, at fastsætte de forskellige relevante termer og den specifikke betydning af disse, som vi udtrykte ved at tillægge begreberne forskellige attributter. Herved mindskede vi risikoen for misforståelser og lagde fundamentet for en mere glidende kommunikation. Gloselisteartefaktet er en nødvendighed ved interaktion med kunder, men det skal heller ikke undervurderes i forhold til den interne forståelse mellem systemudviklere, hvorfor det var relevant for os at benytte.

Vi fokuserede næsten udelukkende på at udvikle de grundlæggende idéer til simulationen i inceptions-fasen, og vi påbegyndte derfor ikke noget egentlig kodeskrivning. Vi lavede endvidere ikke korrekt UML notation i denne fase. På denne måde dannede vi et solidt analytisk fundament forud for elaborationsfasen og de senere designdele. Vi planlagde afslutningsvis omfanget af den kommende første iteration i elaborationsfasen, som vi besluttede at afvikle i perioden fra den 28. marts til den 19. april.

Elaboration

I inceptionsfasen havde vi diskuteret og fastlagt de grundlæggende krav for simulationen og dannet grundlaget for at påbegynde elaborationfasen med et udmærket overblik over projektets omfang. Vi uddybede i denne fase, idéerne fra inceptionsfasen og påbegyndte implementeringen af de mest risikofyldte områder. Vi udviklede adskillige domæne- og designmodeller, som vores idéer influerede på, og brugte omvendt disse til inspiration. Derudover lavede vi nogle flere brugsmønstre og udviklede tilmed sekvensdiagrammer ud fra disse. Vi brugte Larmans princip om *timeboxing*, som danner grundlaget for den overordnede struktur af *Unified Processing,* hvormed vi havde et overblik over omfanget af de enkelte iterationer. Derudover bevirkede brugen af dette princip, at det var nemmere at overholde den overordnede deadline for projektet, da disse partielle deadlines er mere overkommelige.

Første iteration

(28. marts – 19. april)

I denne iteration startede vi ud med at fokusere på udviklingen af domænemodellen, som vi lavede med udgangspunkt i de brugsmønstrer, vi havde fra inceptionsfasen og de generelle overvejelser vi havde gjort os omkring systemets fokusområde, og vi analyserede os vha. disse domænemodeller frem til nyttig information om centrale begreber i vores domæne. Derudover lavede vi vores første klassediagram og tillagde os en masse erfaring omkring, hvilke dele af systemet vi kunne definere som højrisikoområder, og vi begyndte at lave den første egentlige kode af disse områder. I staten af kodeprocessen valgte vi at fokusere på, hvordan vores simulation helt basalt skulle populeres, hvilket resulterede i en række overvejelser.

Vores implementering startede ved helt i dybden på simulationen, og i længere tid fungerede den helt konsol-baseret, med hvilket der menes, at GUI oprindeligt var meget lavt prioriteret og kun blev tilføjet som de sidste elementer. Vi startede ud med at lave et todimensionelt array, hvori vi kunne placere zombier. Efterfølgende overvejede vi en række forskellige metoder til at populere bygninger, hvilket gav anledning til en række spændende idéer, som er forklaret i afsnittet om *Overvejelser*.

I forbindelse med dette endte vi lidt i en blindgyde, da vi foretog lidt for meget analyse frem for at fokusere på designdelen og få skrevet noget egentlig kode, vi kunne teste. Vi lavede endvidere vores første udkast til et GUI prototype ud fra vores overvejelser, samt de forskellige brugsmønstrer og systemsekvens-diagrammer vi havde.[2]

Dette udformede vi i fællesskab på et white-board, som var en helt central del af udviklingen af idéer i forhold til systemet. Vi har brugt dette kommunikationsmiddel, som er en del af Larmans idé om Agile Modeling, for at undgå misforståelser og hurtigt få nogle idéer belyst. Vi planlagde afsluttende i denne iteration, den estimerede længde af næste iteration, som skulle afsluttes den 10. maj.

Anden iteration

(19. april – 10. maj)

I denne iteration begyndte vi udviklingen af sekvensdiagrammer, hvor det mest sigende omhandlede bevægelse af de forskellige væsener ud fra vores brugsmønstrer, hvilket vi anså dette for værende et højrisikoområde i systemet.[3] Vi lavede sekvensdiagrammerne ud fra vores forskellige brugsmønstrer og de enkelte handlinger i disse, hvilket gav os et mere systemnært perspektiv. Denne mere systemnære fokusering resulterede i, at vi måtte opgive en række af vores hidtidige idéer, da de var for omfattende. Vi begyndte i denne iteration at fokusere mere på den egentlige implementering af kode, hvilket belyste vores vision af systemet således, at mange elementer måtte revurderes. Sekvensdiagrammer gav os i forhold til brugsmønstrene en mere visuel repræsentation af systemet, som hjalp os i denne proces. Vi begyndte i denne iteration tilmed at fokusere på klassediagrammer og udviklede adskillige, som gav os et foreløbigt overblik over den softwaremæssige struktur. Klassediagrammerne udviklede sig meget i denne iteration, hvilket bevirkede, at vi fik anskuet systemet fra adskillige vinkler. I forbindelse og forlængelse af disse klassediagrammer foretog vi en række overvejelser, som resulterede i et større overblik over strukturen og udmøntede sig i en flere brugsmønstrer, og vi lavede fungerende kode for populationen af simulationen,

[2] Se bilag XXV
[3] Se bilag VI

som vi fortsat anså for værende et højrisikoområde. I afslutningen af denne iteration, foretog vi en række ændringer, som simplificerede vores simulation kraftigt, da mange af vores idéer var for omfattende i forhold til størrelsen af projektet. Vi havde i disse to første iterationer af elaborationsfasen udviklet en række idéer omkring placeringen af bygninger, som blev revideret og ændrede en række elementer i systemet. Vi havde lidt problemer med at fastlægge længden af den kommende tredje iteration, da vi dels netop havde raffineret vores vision kraftigt og projektet derudover ville blive nedprioriteret indtil den 23. maj, hvor vi havde deadline for et andet eksamensprojekt. Dette bevirkede, at det tidsmæssige omfang af denne iteration nødvendigvis måtte forøges i forhold til de tidligere iterationer, og vi valgte derfor at sætte en deadline for tredje iteration den 28. maj.

Tredje iteration

(11. maj – 28. Maj)

I denne iteration måtte vi på baggrund af vores ændringer i anden iteration revurdere en række brugsmønstre og revidere dem ud fra ændringerne. Vi blev i denne del af processen trukket lidt tilbage mod inceptionsfasen, da de store ændringer krævede overvejelser på et *lavere* niveau, end vi foretog i anden iteration, og vi lavede grundlæggende ændringer af vores vision. Vores brugsmønster "*Soldat giver våben til menneske*" blev i denne forbindelse skåret fra, da dette scenarie ikke længere var gældende for simulationen. Vi lavede en kraftig omstrukturering af vores klassediagram og nåede på baggrund af tidligere erfaringer hurtigt et stadie, hvor vi kunne bevæge os imod designdelen af projektet. Vi lavede et nyt sekvensdiagram af en given aktørs bevægelse på grundlag af de brugsmønstrer, vi havde lavet om de forskellige aktørers bevægelser.[4] Dette gav os et overblik over de ændringer, vi foretog i denne iteration. Forud for dette lavede vi en stor ændring i vores klassediagram, da vi på dette tidspunkt bevægede os væk fra vores oprindelige idé om, at de forskellige aktører skulle nedarve fra en *Creature-klasse*. I stedet benyttede vi os af et *statedesignmønster* (forklaret senere), som medførte en større fleksibilitet, når aktørerne skulle ændre tilstand. Vi kunne bruge de tidligere erfaringer fra arbejdet med de forskellige artefakter til kodemæssigt at fokusere på de områder af systemet, som var mest risikable, og vi fik testet mere fungerende kode, såsom at give aktører syn og få aktører til at bevæge sig.

Den iterative proces og vekselen mellem analyse og design medførte i denne sammenhæng, at vi løbende kunne foretage ændringerne og dermed ikke var nødsaget til at starte forfra med designet af softwaren. Vi havde i slutningen af denne iteration en klar forestilling om, hvordan vores system skulle struktureres og udformes, og vi planlagde at fokus for fjerde iteration overvejende skulle indebære designmæssige tiltag og afsluttes den 6. juni.

Fjerde iteration

Efter en tredje iteration, som overvejende opsummerede projektet i forhold til de ændringer, vi havde fortaget, startede vi denne iteration med et øget kodenært fokus. I forbindelse med dette mere kodespecifikke perspektiv lavede vi endnu en revidering af vores sekvensdiagram for en bevægelse, hvilket resulterede i adskillige udspecificerende sekvensdiagrammer for en bevægelse, hvorefter vi testede fungerende kode for både bevægelse.[5] Disse sekvensdiagrammer omfattede tilmed problematikken omkring aktørernes træk, som vi havde lavet brugsmønstrer for i tredje iteration.[6] Vi lavede endvidere

[4] Se Bilag VII
[5] Se bilag VIII, XI og X
[6] Se Bilag I

vores første korrekte UML klassediagram og fik dermed et overblik over, hvilke klasser og metoder vores system konkret skulle indeholde. Vi foretog i denne iteration en stor kodemæssig ændring i forhold til aktørernes bevægelse, da vi oprettede en *enumeratorklasse* til at håndtere dette, hvilket vil blive yderligere uddybet senere i rapporten. Derudover lavede vi et sidste udkast til vores GUI, som på dette tidspunkt ikke krævede de store ressourcer, da vi løbene havde diskuteret dette og tilmed havde en klar forestilling om systemets features. Vi anvendte os selv som brugere og lavede to systemsekvensdiagrammer, som vi brugte til at få en mere konkret forestilling om, hvorledes vores GUI layout skulle hænge sammen og udformede derefter et layout, som vi fandt intuitivt og sigende for simulationen.[7] Vi begyndte i denne iteration at teste højriskoområderne i forbindelse med GUI-delen af vores system, og vi havde i slutningen af denne elaborationsfasen velfungerende kode for disse. Projektet bevægede sig på dette tidspunkt mod konstruktionsfasen, da vi havde testet højrisiko områderne og kun manglede at skrive den resterende kode.

Overvejelser

I dette afsnit vil vi redegøre for de forskellige idéer, vi har haft igennem forløbet, hvilket afspejler den iterative proces vi har gennemgået. Vi har på baggrund af disse overvejelser løbende tillagt os et indsnævrende fokus i forhold til struktureringen og indholdet af simulationen.

Bygninger

Vi fik i første iteration af elaborationfasen en idé om at opdele vores daværende klasse, *Board*, som på dette tidspunkt blev fastsat til størrelse 800*600, i 48 forskellige zoner i forbindelse med placeringen af bygninger. Idéen var, at hver zone skulle indeholde netop én bygning. Til at holde styr på selve placeringen af bygningerne forestillede vi os at bruge en zoneliste. Derudover overvejede vi at lave nogle mere generelle opdelinger i større zoner, hvilket kunne simulere hele bydele som f.eks. soho, chinatown og lignende. Vi ville oprette tre forskellige typer af bygninger, som opererede med et *offSet* for hele zonen og et indenfor zonen. Vi tildelte de forskellige bygninger en form og en størrelse, som opstillet nedenstående og i vedlagte bilag.[8]

- Barak
  - En pentagon
  - 60x60
- Station
  - Illustreret med et M
  - 70x70
- Job (resten)
  - Vilkårlig størrelse (40-60 x 40-60)

I forlængelse af idéen med de forskellige typer af bygninger overvejede vi at simulere en undergrundsbane, hvor besætningen i det enkelte tog skulle illustreres i en separat simulation, som evt. var zoomet tættere på. Denne simulation blev dog hurtigt skrottet til fordel for en metro simulation, hvor en given aktør blot skulle gå ind i en station og lidt efter komme ud af en anden station, hvorved transporten var simuleret. Det skulle være muligt for alle typer af væsener at komme ind i disse toge, hvilket ville bevirke at en zombie i løbet af en togtur kunne smitte en hel besætning og dermed sprede zombier ud i en ny bydel. Dette ville

[7] Se bilag XXVI og XXVII
[8] Se bilag XIIX

medføre en vis spontanitet og adspredelse i simulationen, men denne idé blev desværre droppet, da vi vurderede det ville blive for uoverskueligt.

Der skulle være forskel på placeringshyppigheden af de forskellige typer af bygninger og kun forekomme en station pr. 16 zoner, hvilket vil resultere i tre i alt. I forbindelse med dette overvejede vi sandsynligheden for, at to givne stationer ville komme til at ligge i to zoner, som støder op til hinanden, men vi besluttede, at det ville forekomme så sjældent, at det var acceptabelt. Derudover forestillede vi os, at der skulle placeres 1-2 barakker pr. simulation for at holde genereringen af soldater på et samhørigt niveau med spredningen af zombier. Måden det skulle placeres på var først at lave jobs i alle zoner og derefter placere stationer og barakker ved at overskrive disse jobs. Endvidere overvejede vi en række ekstra features som f.eks., at det skulle være muligt at placere vægge mens simulationen kørte således, at man kunne indkredse dele af simulationen, hvilket ville medføre en øget brugerinteraktion. Tilmed havde vi inden afslutningen af anden iteration en idé om, at vægge og på dette tidpunkt også *gateways* skulle tildeles en form for livspoint, hvor zombierne skulle bruge betydeligt flere ture på at nedbryde en *gateway*. Denne idé blev undladt, da vi først fravalgte idéen om *gateways* og senere hen tilmed valgte at zombier ikke skulle være i stand til at nedbryde bygninger.

Aktører

Udover disse forskellige bygninger skulle vores simulation indeholde nogle forskellige levende væsner; en zombie, et menneske og en soldat. Dette var allerede overfladisk diskuteret i inceptionsfasen, men vi fik en række yderligere idéer i elaborationsfasens første to iterationer.

Mennesker

Mennesker skulle have en bestemt hastighed samt et bestemt mål, som er det mennesket bevæger sig imod. Vi valgte i begyndelsen af projektforløbet at definere målet som værende en bygning, hvilket kunne simulere at de var på vej til arbejde eller lignende og skulle dermed afspejle et mere realistisk bybillede. Vores idé var at oprette en *toDoListe,* som var knyttet til hvert menneskeobjekt således, at de havde en række forudbestemte mål. Denne idé måtte vi dog undlade, da det var for kompliceret at implementere i forhold til tidsrammen for projektet. Aktørernes aktuelle mål skulle annulleres, hvis mennesket så en zombie, hvorefter hastigheden øgedes til det tredobbelte og mennesket flygtede i en tilfældig retning indenfor +- 45 grader direkte væk fra zombien. Hvis mennesket stødte på noget, prøvede det at gå rundt om dette, sådan at det så vidt muligt fulgte sin oprindelige flygteretning. Desuden ville menneskene helt miste deres syn, og de kunne derved ikke tage højde for, at de løb ind i eventuelle zombier. De ville dog stadig spørge systemet om de kunne gå i en retning, sådan at de ikke bare står stille med ansigtet op imod en bygning for at undgå en statisk simulation. Vi udviklede en række idéer om, hvorledes menneskenes synsfelt skulle være og prøvede tidligt at sætte bestemte værdier på dette, hvilket blev ændret gennem den iterative proces. Det er et eksempel på, at vi havde et lidt for stort behov for at specificere de enkelte handlinger i systemet med henblik på at opnå så høj en realisme i simulationen som muligt. Vi valgte, at tildele menneskene et synsfelt, hvor de kunne se i de tre fremadgående retninger, da det er realistisk, at menneskerne kun kan se fremad, da disse ikke har øjne i nakken. Derudover estimerede vi, at det ville være for svært for zombierne at fange menneskerne, hvis de ikke blev pålagt visse begrænsninger, da zombierne kun måtte bevæge sig med halv hastighed af menneskene jf. de formelle regler for programmet. Vi overvejede dog, at menneskene skulle være i stand til at se hele vej rundt om sig selv, da der ellers ville

forekomme scenarier i simulationen, hvor menneskene går direkte ind i en zombie, hvis den placere sig bag ved mennesket.

Soldater

Udover at mennesker kunne blive forvandlet til zombier havde vi også en idé om, at de skulle være i stand til at blive forvandlet til soldater. Vi opererede i både første og anden iteration af elaborationsfasen med to typer soldater, *BarracksSoldier* og *Soldier*. Idéen med den førstnævnte var, at denne type af soldater blev dannet ved at et menneske gik ind i en barak og N ture efter kom ud som en baraksoldat. Dette simulerede en "uddannelse" til soldat, på samme måde som diverse strategispil. Disse baraksoldater var udstyret med en rygsæk, der indeholdte et vist antal våben, som vi på dette tidspunkt satte til fem. Derved kunne disse baraksoldater rekruttere nye soldater ved at give dem et våben. Disse rekrutterede soldater kunne ikke selv rekruttere nye soldater, men var ellers fuldt ud kampdygtige. Ydermere havde vi en idé om, at baraksoldaterne efter at have uddelt alle sine våben vil kunne fylde rygsækken op igen ved at gå ind i en barakbygning. Soldaterne skulle vælge en vilkårlig retning at bevæge sig i, indtil de ramte en væg, hvorefter de skulle vælge en ny vilkårlig retning. Der var to undtagelser til dette mønster, hvor den sidste kun var gældende for baraksoldater – 1) Hvis han ser en zombie 2) Hvis han ser et menneske. Hvis soldater så en zombie skulle de rykke imod denne indtil de var indenfor skyderækkevidde og derefter stå stille og skyde. Derudover havde vi den idé, at en soldat enten kan skyde eller bevæge sig indenfor én tur. Vi forestillede os, at en baraksoldat ville gå imod et menneske og give det et våben, når han så dette. Det ville dog have lavere prioritet end opdagelsen af en zombie. Endvidere fik vi den idé, at flere soldater kunne samles og derved danne en gruppe, hvilket vil medføre nogle fordele i form af et udvidet synsfelt og en større slagkraft. En gruppe skulle skyde asynkront, således de ikke skød den samme zombie. Den simplificering vi valgte at foretage omkring bygningerne resulterede i, at disse idéer omkring baraksoldater måtte tages op til revision og vi endte med at droppe dem. Vi havde også en idé om at soldater skulle have en beroligende effekt på mennesker, som var gået i panik. Formålet med disse idéer var at øge soldaternes indflydelse på udfaldet af simulationen samt skabe en mere dynamisk og underholdende udvikling i simulationen med muligheden for spredning af soldater.

Zombier

Vi overvejede i inceptionsfasen, at zombier hele tiden skulle have et mål at gå efter på samme måde som et menneske, men vi droppede dette, da zombier i vores forstand er simuleret mere realistisk ved at gå helt tilfældigt rundt i en sanseløs *zombiemode.* Derudover havde vi en idé om, at de levende zombier skulle være i stand til at æde de døde zombier og derved få næring, men dette gav dog zombierne for stor en fordel, hvorfor vi droppede det. Vi havde tilmed en idé om, at zombierne kun skulle være i stand til at smitte en soldat pr. tur, men vi vurderede, at dette dels ville gøre spillet for statisk og derudover medføre at styrkeforholdet ville blive skævvredet, da zombierne ikke havde et ligeså langt synsfelt som soldaterne. Denne balancegang i forhold til styrkeforholdet mellem de forskellige aktører fokuserede vi på gennem hele processen. Vi overvejede endvidere at, zombier skulle være i stand til at nedbryde bygninger, hvilket i sidste ende blev droppet, da vi ikke havde den nødvendige tid til at implementere dette.

Aktørhjerner

Vi havde i første iteration af elaborationsfasen en idé om at lave en *enumeratorklasse*, hvor det skulle være muligt at spørge hvert felt, hvilken type aktør den indeholdte og herefter køre et tilhørende "hjerneobjekt", der svarede til denne. Dette ville betyde, at simulationen kunne gemmes som et todimensionelt

heltalsarray. Problemet med dette var dog, at det ikke var muligt gemme tilstande for de forskellige væsner, hvilket var et krav, da f.eks. mennesker skulle være i stand til at huske, at de flygter, og zombier skulle huske, når de havde set et menneske. I stedet besluttede vi at gemme simulationen som et todimensionelt *Piece-array*, hvor hver *Piece* kunne indeholde en af to forskellige typer – *Creature* eller *Other* – den første bestod af dynamiske objekter og den sidste af statiske objekter. Denne model ville medføre en form for *shadow-map*. *Creature* kunne have følgende forskellige tilstande: *Zombie*, *Human*, *Soldier* og *BarracksSoldier. Other* kunne have tilstandene *Empty*, *Edge*, *DeadZombie*, *Wall*, *GateWay*, hvor sidstnævnte skulle være en af de tre specificeringer *Job*, *Station* eller *Barracks*. Disse forskellige typer af bygninger blev som ovennævnt droppet og vi har tilmed undladt at bruge dette *shadow-map til at* skelne imellem dynamiske og statiske objekter, som vi på dette tidspunkt havde forestillet os. Vi droppede vores oprindelige idéer med aktørhjerner i forbindelse med oprettelse af *ContentState* klassen, da de enkelte tilstande nu blev defineret ved hjælp af denne.

Tur, træk og hastighed

Vi har gennem forløbet foretaget en række overvejelser omkring de forskellige *steps* i simulationen, og hvorledes vi ville definere en tur. Vores målsætning var, at simulere hastighed på den mest realistiske måde, og derudover at opnå et fornuftigt styrkeforhold mellem de forskellige aktører, hvilket den hierarkiske fordeling af de forskellige aktørers handlinger influerede på. Vi havde i starten af projektforløbet en forestilling om, at alle objekter skulle foretage et ryk på samme tid, hvor størrelsen af de forskellige objekter skulle være forskellige, hvormed hastighed kunne simuleres. Dette droppede vi, da det ville medføre, at nogle aktører ville stå stille i flere tur, da eksempelvis mennesker i panik, jf. de formelle retningslinjer for simulationen, skulle bevæge sig seks gange så hurtigt som zombier.

Beskrivelse af Unified Process artefakter

Systemudvikling er baseret på de processer, som går forud for udviklingen af artefakter, da personer udenfor udviklingsprocessen ellers ikke ville have nogen forståelse for noter og figurer tegnet af udviklingsholdet. Ved at lave artefakter er det muligt for udviklere at sende deres idéer til programmører og få det udviklet efter deres pakke- og systemdesign. Det kan også muliggøre, at andre programmører kan overtage projektet og udvikle videre.

Brugsmønstre

Brugsmønstre giver en god beskrivelse af projektet i et tidligt stadie i forløbet, da de beskriver funktioner og handlinger i systemet. Udover selve brugsmønsterdiagrammet kan brugsmønstre tilmed indeholde supplerende artefakter. Det er henholdsvis vision, gloseliste og Business Rules, som oversætter til forretningsregler. Sidstnævnte er mere juridisk orienteret, hvilket vil sige, at det vil sige at punktet indeholder restriktioner for patenteret kode. Da dette er en opgave givet af instituttet for information- og medievidenskab, går gruppen ud fra, at der ikke er sådan restriktioner i forhold til de opgaver vi laver. Dette artefakt har vi derfor undladt at beskæftige os med. Vision- og gloselisteartefakterne, som udvikles i inceptionsfasen, er derimod meget anvendelige for projektet, da disse skaber et overblik over indholdet af simulationen. Visionen er defineret ved det, som programmet skal indeholde og danner derved grundlaget for det endelige resultat, mens gloselisten udspecificerer, hvad de forskellige termer og definitioner indeholder. Ved at lave disse artefakter kan man som systemudvikler skabe et hurtigt overblik over, hvordan programmet skal fungerer, og hvilke funktioner en given bruger har rådighed over.

De første tre brugsmønstre er lavet i inceptionsfasen, og da store dele af projektet er blevet ændret siden dengang, er *Soldat giver våben til menneske* ikke længere understøttende for simulationen. Det første brugsmønster *Zombie smitter menneske eller soldat* er lavet med henblik på at belyse smitte problematikken omkring tur baseret styring. Simulationen er baseret på ture, og det var derfor nødvendigt at oprette en prioriteringsliste for at undgå at zombier smitter soldater i samme *action,* som soldater skyder zombier. Det betyder, at der efter en afsluttet bevægelsesfase vil komme en handlingsfase, hvor zombier laver deres handling først, hvorefter soldaterne udfører deres handling.

I elaborationsfasen benyttede vi fortsat brugsmønstre, men vi gjorde ikke så meget ud af dette artefakt, da det gradvise større designmæssige fokus medførte, at brugsmønstrenes betydning blev mindre. Derudover bevirkede den manglende interaktion med en virksomhed, at en række af artefaktets potentiale ikke kunne udnyttes i vores projekt.

Vision

Vi har i vores vision forsøgt at beskrive de forskellige aktører og simulationens overordnede mål. Vi har gennem projektforløbet udviklet adskillige visioner for simulationen. Problemet har været, at vores visioner generelt har været for omfattende til, at de kunne implementeres i tidsrammen for projektforløbet. Vi har som allerede nævnt i afsnittet om overvejelser haft en række forskellige forestillinger, som har været centrale i forhold til vores vision. I opgaveformuleringen fik vi formuleret en grundlæggende vision, som vi har brugt som fundament for projektet. I forlængelse af denne har vi haft en række idéer i forhold til udvidelser, hvor vores inddeling af zoner og de forskellige typer af bygninger i starten af projektforløbet var afgørende i forhold til vores vision. Vi udviklede ud fra disse mange andre idéer, og havde efter anden iteration en klar forestilling om, hvorledes vores vision skulle formuleres.

- Det skal være muligt at pause simulationen.
- Det skal være muligt at justere hastighed.
- Simulationen skal kunne vises *step by step.*
- Antallet af aktører og bygninger skal angives inden stat på simulationen.
- Menneske går efter et bestemt felt i deres synsretning.
- Mennesker i panik skal bevæge sig tre gange hurtigere end mennesker i normalt tilstand.
- Zombier skal gå med halv hastighed af mennesker.
- Soldater skal holde afstand til zombier.
- Soldater bevæger sig imod en zombie, når de ser en.
- Soldater stopper op, når de er indenfor skudafstand af en zombie.
- Bygninger er tilfældig placeret og har tilfældig størrelse.
- Bruger kan fastsætte indstillinger i starten af simulationen.
- Mennesker skal prøve at holde afstand til vægge.
- Der skal være så få aflukkede områder som muligt.

De forskellige aktører skal identificeres med en bestemt farve.

Gloseliste

En gloseliste bliver lavet for at udviklerne har en liste over begreber på indholdet af projektet, og hvordan de forskellige elementer relaterer til hinanden. Der giver et overblik for flere programmører, når de arbejder på samme projekt samtidigt.

Vi lavede vores første gloseliste i inceptionsfasen for at fastligge og forklare de mest grundlæggende termer i simulationen.[9] Vi rettede igennem forløbet listen til i forlængelse af de ændringer vi foretog og tilføjede attributter til gloserne efterhånden, som detaljerne kom på plads. Senere i forløbet begyndte vi at lave længere definitioner af de forskellige termer i stedet for blot at skrive stikord, hvilket medførte en bedre forståelse.

Systemsekvensdiagram

Systemsekvensdiagrammer er udviklet af Craig Larman, men er ikke optaget som en officiel standart UML notation. Men er stadig et godt diagram til at vise, hvad systemet modtager, og hvad det returnerer. Det har også fordelen frem for sekvensdiagram, at det kan arbejde med andre systemer. Som eksempel kan det være et system, der skal hente data fra en database, hvilket ikke kan illustreres i et sekvensdiagram.

Vores simulation tager ikke imod mange kommandoer fra brugeren og har heller ikke interaktion med andre systemer. Derfor har dette artefakt ikke været oplagt for vores projekt, men vi har lavet to systemsekvensdiagrammer, hvor det ene er en beskrivelse af den interaktion brugeren har med systemet ved oprettelsen af en ny simulation.[10] Derudover har vi illustreret, hvorledes en given bruger kan interagere med systemet imens en simulation er i gang.[11] Disse to diagrammer udviklede vi allerede i første iteration af elaborationfasen for at få et illustrativt overblik over, hvilke inputs og outputs systemet havde i forhold til en potentiel bruger. Vi har i forbindelse med systemsekvensdiagrammer snakket om muligheden for at indsamle data fra de gange systemet bliver kørt og samle dataene på en fælles database.

Sekvensdiagram

Sekvensdiagrammer fungerer ligesom interaktionsdiagrammer som kommunikationsdiagrammer for systemspecifikke handlinger. Sekvensdiagrammer og interaktionsdiagrammer bruges til at belyse det samme, men har deres fordele og ulemper. Et sekvensdiagram er let overskueligt, da det følger læseretningen, og det giver et hurtigt overblik over, hvordan den givne systemproces fungerer. Udover det fylder det også relativt lidt, da sekvensdiagrammer størrelse ofte kan tilpasses et A4 ark. Ulempen ved sekvensdiagrammer er, at det besværligt at slette notationer i udviklingsfasen, fordi det metodekaldene meget tæt, hvilket vi har haft lidt problemer med i vores brug af disse. Interaktionsdiagrammer har derimod den fordel at det er lettere at foretage ændringer i diagrammet, hvis man vil fjerne et punkt. Dette opvejer dog ikke den ulempe, at de fylder meget og er ikke specielt intuitive at læse. Læseretningen er bestemt af tal, der står rundt omkring i diagrammet, hvilket kan virke uoverskueligt for personer der ikke har været med i udviklingen af diagrammet. Vi har overvejet disse fordele og ulemper og valgt kun at benytte os af sekvensdiagrammer, da vores rapport er udskrevet på A4 papir og er henvendt til læsere der ikke har set vores tanker eller modeller før programmet er færdigudviklet.

[9] Se bilag II
[10] Se bilag XXI
[11] Se bilag XX

Det første sekvensdiagram vi lavede omhandlede bevægelse af aktører i simulationen. Diagrammet viser, at den daværende klasse *GW*[12] aktiverer metoden *decideMove,* og hvis det er en lovlig bevægelse for objektet, og *pixel* er fri, vil den returnere en *booleanværdi*, der får *board* til at returnere endnu en boolean værdi. Til sidst fortæller *Creature GW* om sin nye position. Det får *Board-klassen* til at bruge metoden *moveCreature*, hvilket betyder at en *pixel* bliver tom og den nye positions *pixel* bliver fyldt. Dette sekvensdiagram er fra anden iteration, og der er sket meget med opsætningen af simulationen siden.[13]

Den opdaterede beskrivelse af en bevægelse er blevet mere avanceret i de nye sekvensdiagrammer, og diagrammerne viser mere præcist, hvad der foregår internt i koden, når et objekt skal bevæge sig, da vi har brugt flere sekvensdiagrammer til at beskrive dette. Vi valgte at dele sekvensdiagrammet op i mindre dele, da beskrivelsen af en bevægelse er for omfattende at samle i et diagram. Et stort diagram, kan sige meget, men under udviklingen er det lettere at overse eller glemme nogle detaljer, og derudover har white-boardet, som vi har anvendt hyppigt i forbindelse med kreeringen af diagrammer, sat begrænsninger i forhold til størrelsen af diagrammerne. Den første del af sekvensdiagrammet omhandler den del af funktionen, hvor aktøren finder ud af, hvor denne befinder sig, og i hvilke retninger, objektet kan bevæge sig. Modellen finder ligeledes ud af nogle attributter for den aktør, der skal foretage en bevægelse. Det gør den ved at finde længden af synet, hvor langt aktøren kan gå og hvor hurtigt aktøren bevæger sig, hvilket er defineret ved hjælp af *step*. Disse informationer bliver returneret i en *Arraylist* og til sidst har *Model-klassen* information om, hvordan de forskellige aktører opfører sig. Dette er *Model-klassen* nød til at gøre hver gang, der skal foretages en bevægelse. Hvis det ikke blev gjort, ville den ikke tage højde for de aktører, der ændrer tilstand, og så ville *deadZombie* ikke have nogen egentlig funktion. Når en aktør ser sig omkring, vil denne køre en række metoder for hver retning denne ser mod.

Vi har brugt vores brugsmønstre som inspiration for genereringen af sekvensdiagrammer og vil derfor forklare sammenhængen mellem disse i forbindelse med beskrivelsen af en bevægelse.

Antallet af disse udførelser er defineret ud fra den givne aktørs tilstand, hvilket konkret vil sige, at zombier vil afvikle denne funktion otte gange, hvilket simulerer, at den ser 360 grader rundt om sig, mens et menneske vil have tre afviklinger. For hver afvikling finder *Model-klassen* navnet på indholdet af den *Piece,* der bliver spurgt til. Hvis den får en *empty* værdi retur, vil den tilføje det til en midlertidig variable, hvorefter hele indholdet tilføjes til en *Arraylist*. I vores brugsmønstre svarer det til at finde eventuelle fejl i bevægelsen, så en aktør ikke går ind i en væg og bliver stående. Sammenhængen mellem brugsmønstret og *optional* i sekvensdiagrammet er, at der hele tiden vil være forskellige scenarier for, hvad de forskellige aktører ser – et menneske skal f.eks. gå i panik og løbe den modsatte vej, hvis en zombie observeres. Soldater forsøger omvendt at komme indenfor skudafstand af zombierne, når disse observeres.

Den næste del af sekvensdiagrammet handler om selve bevægelsen af aktøren. Det vil sige punktet, som omhandler *Main Succes Scenario* i vores brugsmønstre. *Model* klassen eksekverer metoden *decideMove(results)*, hvor den kommer tilbage med et resultat, som kontrollerer om metoden blev afviklet korrekt, eller der kom nogle abstraktioner fra det ønskede resultat. Den sidste interne metode for *Model-*

[12] Forkortelse for *GameWorld*
[13] Se bilag VI

klassen er en opdatering af aktørens placering, hvor det bliver gemt som et midlertidigt objekt og sendt til *Content* klassen, som derefter sender det videre til *ContentState* klassen, med de opdaterede koordinater.[14]

Domænemodel

Domænemodellen indeholder ikke kodespecifikke klasser, men derimod mere konceptuelle klasser i forhold til systemet. I vores simulation er de forskellige klasser for aktører og klassen, som indeholder information omkring bygninger eksempler på sådanne klasser. Domænemodellen har gennem forløbet givet os overblik over projektets designmæssige del.

Vi har flere gange påbegyndt en domænemodel, hvor det er endt ud med at blive et klassediagram. Dette hænger sammen med, at programmet er en simulation, og at det derfor kan være nemmere at have en strukturel softwaremæssig tilgang til diagrammets indhold i stedet for at operere med konceptuelle klasser.

For at finde de konceptuelle klasser angiver Larman, at man med fordel kan lave en kategoriseret liste i forhold til projektet og skrive en lille beskrivelse til hvert punkt. Endvidere kan en liste med navneord ud fra de forskellige brugsmønstre, der er lavet til projektet, være anvendelig. Vi benyttede metoden, hvor en liste af navneord oprettes, for at lave domænemodellen. Vi brugte i denne forbindelse vores brugsmønstre og gik succes kriterierne igennem for navneord, hvorefter vi opstillede dem i en liste. Det var ikke muligt at lave en direkte konversion fra navneord til domænemodel, da der kan forekomme redundant data, og det derudover ikke er alle navneord, der egner sig til at blive defineret som en klasse. I forbindelse med kreeringen af modellen reducerede vi den oprettede liste således, at den blev tilpasset modellen.[15]

Første domænemodel

Vores første domænemodel blev meget avanceret, da vi hurtigt udviklede en række idéer. Derfor blev denne idérige model skåret ned til en mere realistisk model. Den første model indeholdte vores idé med de tre forskellige typer af bygninger, som henholdsvis var *Barrack*, *Train Station* og *Job*. De er blevet fjernet og lavet om til en klasse der hedder *Building* efter idéen med forskellige former for bygninger blev droppet. Det betød, at menneskerne ikke havde noget mål at gå imod, hvilket vi i stedet løste ved at definere målet som et *Piece* i deres bevægelsesretning. *Soldier* klassen indeholder attributterne, *inventory* og *weapon*, som var lavet med henblik på at soldaten kunne give sit våben til mennesker og på den måde generere nye soldater.

Seneste domænemodel

I den seneste domænemodel har vi tilføjet brugere, som mangler på de tidligere modeller. Brugeren er blevet undladt i de tidligere, da vedkommende ikke har den store indflydelse på selve simulationens forløb. Det er ydermere udelukkende i forbindelse med GUI delen af vores program, at brugeren direkte interagere med vores system, hvilket mindsker dennes rolle. Derudover har vi fjernet de tre klasser, som repræsenterede de forskellige typer af bygninger og har nu kun klassen *Building* til håndtering handlinger i forhold til bygninger. Vi har tilmed oprettet klasserne *HumanPanic* og *DeadZombie* i den nyeste domænemodel, da vi gik fra idéen om at nedarve til brugen af en klasse som håndterer de forskellige aktørers tilstande.

[14] Se bilag VIII, IX, X
[15] Se bilag III

Klassediagram

UML Klassediagrammet beskriver softwarestrukturen i et program ved at illustrere sammenhængen imellem de forskellige klasser. Et formelt korrekt klassediagram indeholder både klasser og metoder, og deres returtyper, hvilket er et godt udgangspunkt for at kode et program. Vores udkast til et klassediagram har gennem forløbet ændret sig adskillige gange, da vi i de forskellige iterationer har tillagt os erfaring, som vi har brugt til at optimere disse modeller. Vi har gennem forløbet benyttet os af en meget uformel notation i forhold til udviklingen af klassediagrammer, hvilket har medført en hurtig arbejdsproces.

Første udkast til et klassediagram

Vi lavede det første klassediagram i første iteration af elaborationsfasen, hvorved vi dannede en overordnet struktur over den softwaremæssige del af projektet. Vi havde på dette tidspunkt en idé om, at de forskellige aktører i simulationen skulle nedarve fra en *Creature* klasse. Dette viste sig dog at være problematisk i forhold til at aktørerne skulle ændre tilstand, hvorefter vi måtte revidere denne idé. Det bliver beskrevet yderlige i beskrivelsen af *shadowmap* diagrammet.

Indtil anden iteration af elaborationsfasen arbejdede vi med en *Board* klasse, der har virket som modellen i Model-View-Control mønsteret. I forbindelse med vores avancerede planer med bygninger havde vi længe forestillet os at klassen, *BuildingCreater*, skulle være en klasse for sig, men i starten af anden iteration lavede vi markante nedskæringer, som influerede på vores klassediagram. Kompleksiteten af *BuildingCeator* blev herefter reduceret i en sådan grad at den uden problemer kunne presses ned i en enkelt metode i *Board*. Denne metode er dog vigtig, da den sørger for at bygningerne ikke bliver placeret således, at de skaber indelukkede områder i simulationen.[16]

Som det ses har vi en *Other* klasse og som skrevet tidligere, er de tidlige diagrammer ikke komplette. I det første udkast til et klassediagram har vi ikke beskrevet, hvad klassen *Other* kunne indeholde, men meningen var at den skulle håndterer bygninger og deres givne attributter ved at være nedarvet eller implementeret i den.

Shadowmap

Vi lavede en række forskellige ændringer før vi nåede vores endelige klassediagram, hvor det mest interessante var baseret på et *shadowmap*.[17] I dette diagram benyttede vi stadig klassen *Board*, som var klar *information Expert.* Vi havde tilføjet klassen *Zone,* som skulle inddele vores simulation i forskellige zoner. Denne havde en relation til *BuildingCreator,* som på dette tidspunkt fungerede som en klasse, da idéen var, at bygningerne skulle populeres i deres respektive zoner. Derudover havde vi valgt at dele alle objekter op i henholdsvis statiske og dynamiske.

Shadowmap kommer af at der vil være to kort, som systemet gennemløber. Et kort, hvor alle bygninger er på, det hedder et statisk kort, fordi indholdet ikke vil bevæge sig. Det vil betyde at systemet kun skal læse kortet en gang og derefter vil resten af beregningen være på de bevægende aktører. Det vil betyde at den største beregning vil være under opstart af systemet, fordi systemet skal organisere felterne med bygninger i. I det andet kort, vil alle de bevægende aktører være organiseret, og deres mulige felter. Det betyder, at de ikke vil søge bevægelse på felter der ikke kan bevæge sig ind på. Dette kunne have sat bevægelses-

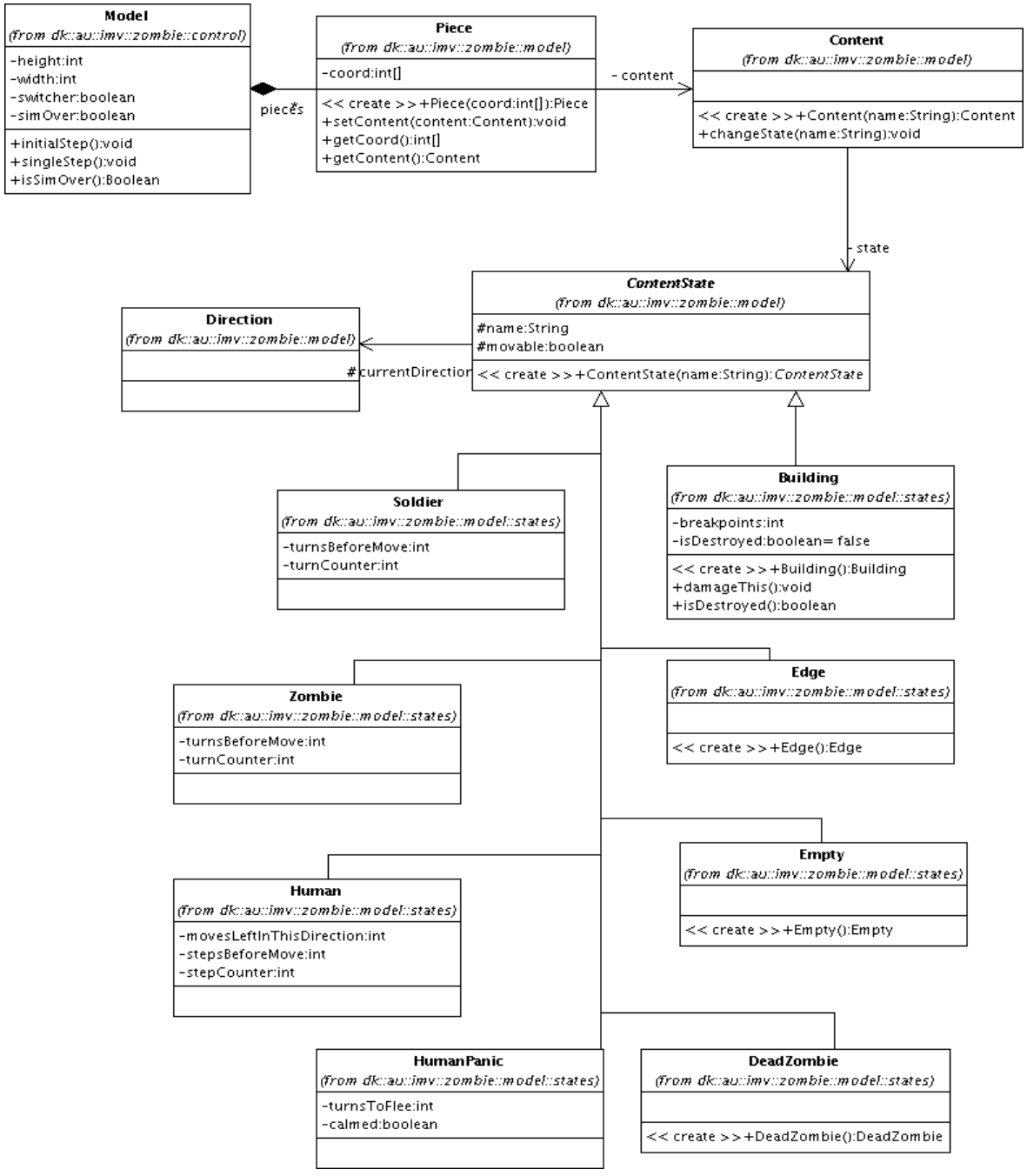
[16] Se bilag XI
[17] Se bilag XII

beregningen ned, og fordi gruppen gerne vil have zombie aktørerne til at nedbryde bygninger, gik vi væk fra den ide igen.

Vi begyndte tilmed at benytte en *stateclass* for at håndtere de forskellige tilstande. Denne idé var mere effektiv end den tidligere ide med nedarvinger, da dette gjorde det muligt at objekterne kunne ændre deres tilstande. Hvis vi var blevet ved vores ide med nedarving, havde det betydet at alle aktører skulle skiftevis fjernes og tilføjes, hver gang en tilstand skulle skiftes, hvilket ville nedsætte hastigheden betydeligt, og den gamle model ville kun virke med relativt få aktører. Derimod er det muligt med tilstande, at simulationen nøjes med at ændre tilstande af dens aktører.

Endelig Diagram

I nedenstående UML klassediagram ses vores endelige version af dette.



Til at holde styr på de forskellige tilstande for aktører oprettede vi klassen *Content*, og vi benyttede os derved af Pure Fabrication og Indirection GRASP mønstre.

I *ContentState* er indholdet baseret på de overordnede tilstande objekterne nedarver, hvilket betyder at vi har lavet nogle *states*, som underklasser, der skal bruge nogle standarter fra *ContentState*. *ContentState* indeholde de forskellige tilstande, der er mulige for aktørerne, det betyder at *ContentState* har information om antallet af aktører og hvilke af de forskellige aktører, der har hvilke tilstande. *Content* kan så interagere med de forskellige tilstande igennem *ContentState*, og da Content har en metode, der ændrer tilstande, *changeState,* betyder det, at den kan skifte tilstand til alle de objekter som arver fra *ContentState*.

Bevægelse for aktører foregår gennem *Model*, den indeholder information om alle *Pieces* og er derfor information expert i bevægelse af aktører i simulationen. Metoderne *initialStep* og *singleStep* er essentielle for bevægelse af aktøren, *initialStep*, opdaterer vores oprindelige tal til GUI'en, og *singleStep* er den enkelte bevægelse. Sammen udgør de bevægelse for simulationen, de bestemmer, hvornår de forskellige bevægelser skal tages i forhold til de forskellige objekter i simulationen.

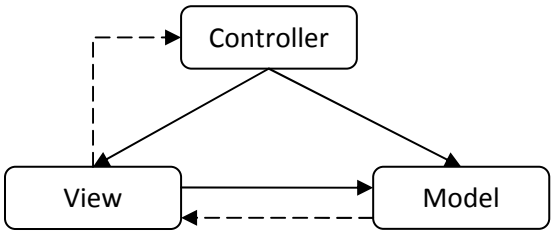
Hovedtanken bag simulationen har været, at der ved slutningen skulle vises et resultat. Gruppen har snakket om mange forskellige måder at vise resultatet på, vi har snakket om graf visninger, store talrækker og pointsystemer. Der ligger stadig nogle påbegyndte ideer til dette, det er gjort fordi vi gerne vil arbejde videre med koden efter projektets fastsatte tidsramme. Det er resulteret i at afslutningsmetoden stopper funktionaliteten for knapperne og skriver en besked om det sejr til menneskeheden eller zombierne vandt.

Struktur på programmet

Hvor de tidligere afsnit definerer de bevæggrunde, som ligger til grund for programmet, vil dette omhandle, hvordan programmet hænger sammen under og på overfladen. Overordnet har vi benyttet en *model-view-controller* (MVC) model, der separerer det grafiske interface fra det underliggende program. Modellen er selve programmet, i vores tilfælde simulationen, og viewdelen er den grafiske visualisering af denne. Kontrolløren fungerer som mellemmand og sørger for, at viewdelen opdateres, når der sker ændringer i modellen. Endvidere sørger den for, at modellen afspejler ændringer fra view, hvilket kan være i form af pause eller andet. Fordelen i denne opdeling er, at simulationen kan afvikles uden at bemærke observeringen af denne, hvilket betyder, at der kan være flere forskellige visuelle repræsentationer af samme simulation.

I forbindelse med MVC, benyttes ofte et andet designmønster, som kaldes *Observer*. Et *observerpattern* fungerer som en *listener* i Java, hvor man i hver klasse, der implementerer Observer, skal definere en *update* metode, som udføres hver gang det objekt man observerer, gør opmærksom på en ændring.

MVC og observer pattern



I ovenstående model er de stiplede linjer en observationsretning, og de fuldt optrukne linjer en direkte associationsretning. Den sidste betyder, at den der peges fra har fuld adgang til den der peges på, som metoderne tillader.

Den overordnede controller, i vores simulation, er *SimFrame* klassen, som er den, der starter hele simulationen. Denne indeholder også viewdelen, som er sammensat af *JPaneler* af forskellig art. *SimFrame* har direkte association til både *Model,* og viewdelen, hvor view delen har adgang igennem observer-mønsteret.

I vores implementering starter vores modeldel, med klassen *Model*, som agerer *Controller*, i GRASP forstand, for selve modellen. Denne starter alle afviklinger og sørger for, at simulationen kan køre. Husk, at dette ikke er det samme som controller i MVC forstand.

Modelklassen er endvidere *creator* for *Piece* og *Content*. Dette er fordi *Model* indeholder *Piece*, og er *information Expert* for Content, da det er denne, der ved, hvor et givent *Content* skal oprettes og gemmes i et *Piece*. Hvis der kigges nærmere på Contentklassen, ses det endvidere, at der her benyttes et s*tatepattern*, som er en form for *indirection pattern*. Dette betyder, at *Content* kun eksisterer for at kunne indeholde en *ContentState*, som definerer alle *Contents* handlinger og bevægelser. *Content* indeholder kun metoder, som sendes videre til *ContentState*.

Et eksempel kan ses her:

```
//Fra Content
private ContentState state;
//snip
public Direction decideMove()
   {
      return state.decideMove();
   }
```

```
//Fra ContentState
public Direction decideMove()
{
   return Direction.NONE;
}
```

Disse to eksempler er meget visende for, hvordan *Content* virker, ved netop at videresende opgaven, og returnere det modtagne svar. Dette er *indirection* mønstret i aktion.

Det andet aspekt – statepattern – ses i funktionen *changeState*(), hvor et givent *Content* kan ændre sin tilstand til noget andet, og derved ændre hele sin opførsel. Statepattern benytter sig af et andet og meget brugt designmønster, som også er propageret som GRASP mønster, nemlig *polymorfi*. Som det kan ses, vil *ContentState* altid returnere `Direction.NONE`. Dette bliver dog ophævet, da der aldrig bliver lavet et *ContentState* objekt, der bliver kun instantieret objekter fra nedarvende klasser, såsom human og zombie. Disse bliver blot gemt i *Content* som en *ContentState* type. Dette betyder, at klasser udefra kun har adgang til metoder, og evt. variable, som bliver synliggjort af *ContentState* klassen. Disse er dog overskrevne i de relevante subklasser, hvilket betyder at de får en anden opførsel. Et eksempel kan ses nedenstående.

```
//Fra humanPanic
public Direction decideMove()
{
   turnsToFlee--;
   if (turnsToFlee == 0)
      calmed = true;
   return currentDirection;
}
```

GUI

Vi har lavet visse sjove interaktioner i vores GUI, som er værd at bemærke. I vores opsætningsdel, har vi gjort maksimum for aktører justerbare således, at størrelsen af simulationen bestemmer, hvor mange aktører, der maksimalt kan være i en given simulation. Dette er implementeret for at minimerer antallet af systemnedbrud, der forekommer, da vi har sat systemet til at lukke (`System.exit(0)`), hvis der er flere aktører end, der er pladser til dem. Dette kunne være omgået på andre måder, men vi mener, at man skal have mulighed for at fylde simulationen med så mange som muligt, uden at være *for* begrænset. Samtidig ville vi ikke have at brugere med vilje kunne nedbryde systemet, ved at indtastet et urealistisk højt tal, hvorfor kun sliderne er aktive.

Datastrukturer

Det er vigtigt at bemærke, hvordan vi lagrer vores data. En given model er defineret til at have et kvadratisk bræt af *pieces*, som skal fungere som spillepladen for vores simulation. Denne gemmer vi som et todimensionelt array, hvilket ud over en god gennemløbshastighed, betyder, at vi kan udregne koordinater på de enkelte *pieces*, ved simpelthen at tage de variable, som vi løber igennem med. Hvor *piece* indeholder så sin egen koordinat, så vi kan arbejde med denne, uden at skulle starte forfra med optællingen. Disse *pieces* indeholder endvidere et *Content*, som er det aktører, eller andre ting på banen, der skal gemmes. Når en aktør bevæger sig fra et *piece* til et andet, betyder det, at den bytter plads med en *empty content*. Dette kan kort illustreres i nedenstående kode.

```
//Fra Model
if (getPiece(newX, newY).getContent().isEmpty())
{
   Content tmp = getPiece(newX, newY).getContent();
      getPiece(newX, newY).setContent(getPiece(oldX, oldY).getContent());
      getPiece(oldX, oldY).setContent(tmp);
//snip
}
```

Tidligere lavede vi et nyt *empty content* og ”glemte” det gamle *content* til *garbagecollectoren*, men den ovenstående, er en bedre metode, da den er hurtigere.

Vi bruger også en *enumerator* klasse til at lagre vores retninger (Direction), hvilket betyder, at vi kan kalde funktioner såsom `Direction.N`, da disse er statiske. Vi har endvidere defineret visse metoder, såsom *turn* i denne klasse, hvilket betyder, at vi kan kalde en funktion som denne: `Direction.N.turn(2)`, hvilket vil returnere `Direction.W`. Dette betyder, at en aktør kan gemme en *Direction* i en variabel, som så kan drejes. Denne retningsmodel er helt relativ, hvilket betyder, at man kan dreje x til hver side, plus/minus, og så drejes der henholdsvis med og mod uret.

Bygninger og populering

Vores populeringsproces er blevet godt effektiv og fungerer på følgende måde. Metoden `createBuildings(int numberOfBuildings)` skaber en ny bygning for hver `numberOfBuildings`, der er givet med. Disse bygninger er defineret til at have en ramme på to *pieces*, der er defineret som *empty*, hvilket derfor minimerer antallet af indelukkede aktører. Dette betyder, at bygninger sagtens kan overskrive hinanden, og der her er et mulig optimeringspunkt.

Efter alle bygninger er placeret, gennemløbes hvert enkelt *piece* af *`compileFreeList()`*, som samler koordinaterne for hver *piece*, der indeholder et *empty content*, i en *ArrayList freeList*. Denne *freeList* benyttes så til at populere banen med aktører.

```
//Fra Model.populate(…)
for (int i = 0; i < numberOfZombies; i++)
{
   index = Model.random(0, freeList.size());
   int[] coord = freeList.get(index);
   freeList.remove(index);
   pieces[coord[0]][coord[1]].setContent(new Content("Zombie"));
}
```

Det kan her ses, at der vælges en vilkårlig koordinat fra *freeList*, som bruges til at indsætte en ny zombie. Derefter bruges en *`remove(index)`* funktion, som er meget intuitiv, men desværre meget ressourcetung, da den kræver, at alle indgange i den givne *ArrayList* skal kopieres i baggrunden, hvilket vil sige denne har en potentiel kørselsltid på $O(n)$ pr. zombie. Denne *`remove(index)`* funktion har vi haft andre problemer med, som det vil blive vist nedenstående.

Aktører

Som det kan ses bilag IIX, IX og X, og i det tidligere *flowchart* om *move*, fungerer en bevægelse i vores simulation, ved at modellen henter information om synslængde og synsretninger fra hver enkelt aktør, ”ser” for denne, og returnerer resultaterne til aktøren, som så finder ud af hvilken retning denne vil, og returnerer denne retning til modellen. Denne bevæger aktøren i den valgte retning, hvis det er muligt. Når en bruger har lavet sin bevægelse, skal der derefter efterses, om en handling skal udføres, som ænder den pågældende aktør til noget andet. Dette er også anskueliggjort i *flowchartet* om *move* i et tidligere afsnit. I fald af en given aktør bliver ændret, skal denne tages ud af sin types bevægelsesliste og eventuelt over i en anden. Oprindeligt brugte vi den meget ressourcetunge funktion *`remove(index)`*, som har den førnævnt store kørselstid. Dette betød, at for hver enkel aktør, der skulle ændre tilstand i en given tur, skulle der bruges mindst $O(n)$, hvilket løber op i uoverskuelig tid, og medførte at vores simulation var reduceret til et statisk maleri.

Vi optimerede dette, ved at omgå metoden, således at vi altid har to lister for alle aktører, der kan bevæge sig. For hver anden tur opdaterer vi den modsatte liste samtidig med vi løber den første liste igennem. Dette betyder, at vi kan fjerne aktører fra en given liste ved *ikke at tilføje dem* til den anden liste. Dette sparer kolossal tid og gør vores simulation brugbar – eller rettere *afviklebar*.

Vi har givet de forskellige aktører hver deres specifikke kunstige intelligens, hvilket betyder, at de har deres egne unikke bevægelsesmønstre baseret på, hvad der er omkring dem.

Zombier og soldater har stadig en konkret handling at lave, da disse skal kunne henholdsvis smitte og skyde. Zombiers metode er relativt enkel, da *Model* blot efterser alle felter direkte ved siden af hver enkelt zombie og ændrer eventuelle smitbare mål til zombier. Dette er meget enkelt, eftersom vi har en *ArrayList* med *pieces*, der indeholder zombies, og hvert *piece* kender sit eget koordinat. Derved kan vi bruge relative koordinater, og gå ind i de specifikke indgange i vores todimensionelle *Piece* array. Dette vises i nedenstående kode:

```
//Fra Model
private void zombieAction()
{
   newZombieList.clear();
   for (Piece zombiePiece : getRelevantList("zombie"))
   {
      int[] coord = zombiePiece.getCoord();
      for (int x = -1; x <= 1; x++)
      {
         for (int y = -1; y <= 1; y++)
         {
            int infectX = coord[0] + x;
            int infectY = coord[1] + y;
            if (isLegalLocation(infectX, infectY))
            {
               Content target = pieces[infectX][infectY].getContent();
               if (target.getName().equalsIgnoreCase("human")
                  || target.getName().equalsIgnoreCase("humanpanic")
                  || target.getName().equalsIgnoreCase("soldier"))
               {
                  target.changeState("zombie");
                  newZombieList.add(pieces[infectX][infectY]);
                  zombiePiece.getContent().resetLife();
               }
            }
         }
      }
   }
}
```

Optimeringer

GUIproblemer

Undervejs har vi lavet flere optimeringer og, som altid, er der stadig plads til mange flere. Den største hastighedsbegrænsning lå i, at vi oprindeligt benyttede Swings interne `graphics` til at repræsentere vores *Piece[][]* direkte på et *JPanel*, som små firkanter. Denne tegne metode er ufattelig langsom, da hvert enkelt rektangel skal skrives til panelet enkeltvis. En af vores tidlige forsøg på at optimere dette gik ud på, at kun skrive de specifikke pixels, hvor det respektive *piece* i modellen blev opdateret. Dette betød, at vi lavede et tilsvarende todimensionelt *Color array*, som vi fyldte med de passende farver, og derefter skrev de passende opdaterede *colors* ud. Dette var dog stadig langsomt, og vores simulation ville kun kunne afvikles i fornuftigt tempo som maksimal 200x200 med få hundrede aktører i. Dette var ikke acceptabelt, da vi ville have simulationen op på minimum 500x500 og mulighed for at indsætte mange tusinde aktører.

Løsningen præsenterede sig med en teknik kaldet *doublebuffering,* hvorved vi først skriver et `BufferedImage` i en variabel *image*, og derefter skriver hele denne til et panel (*SimPanel*). Dette i sig selv sætter hastigheden i vejret, men udover dette, optæller vi hver enkelt opdaterede piece i en liste, som vi hver tur løber igennem og benytter til at opdatere hvert relevante pixel i *image*. Dette kaldes at *blitte* et billede og giver en massiv forøgelse i ydelse, da der spares markant *overhead*. Denne optimering medførte, at vores simulation næsten udelukkende er begrænset af vores bagvedliggende model.

Modelproblemer

Som tidligere nævnt havde vi påfaldende problemer med `ArrayList.remove(index)`, da denne er fundamentalt langsom. Vi omgik de store problemer med tilstandsskift i forbindelse med observeringer af andre aktører, da vi lavede det løbende med hver bevægelse. Vi har dog ikke gjort dette med alle tilstands-

skift, hvilket betyder at disse stadig er pinligt langsomme. De drejer sig præcis om `zombieAction()` og `SoldierAction()`, som henholdsvis ændrer *human*, *humanPanic* og/eller *soldier* til *zombie*, og *zombie* til *deadZombie*.

Det skal i øvrigt nævnes, at vi forsøgte at lave den umiddelbare løsning, med at benytte *LinkedList* i stedet for *ArrayList*, da den førstnævnte kan lave *remove* i $O(1)$ tid. Desværre er *LinkedList* iøjefaldende langsommere end *ArrayList* til at vokse i størrelse, hvilket er en feature, der er meget behov for i denne simulation.

Vi har overvejet visse forskellige løsninger til disse problemer, og har fundet to forskellige metoder til at omgå *remove* funktionen. Hvis vi ser på `SoldierAction()`, kunne dette løses, ved at lade soldater skyde zombier, og lade zombierne blive i zombielisten til næste gennemgang. Dette ville medføre, at der ville være et vist antal døde zombies fra den foregående tur, der blev renset ud af listen ved næste gennemgang, igen ved *ikke at tilføje dem* til den næste turs liste. Denne løsning kunne dog også benyttes til `zombieAction()`, da samme princip bare skulle gøres i tre forskellige gennemløb, dvs. for henholdsvis *soldier*, *humanPanic* og *human*.

Det eneste umiddelbare problem med denne løsning er, at der vil være en enkelt tur, som hele tiden halter mere end den bør, da der er et *overhead* på alle de tilstandsændrede aktører. Endvidere ville visse aktører ikke få lov til at bevæge sig, afhængigt af prioriteringslisten, selvom dette kunne løses ved at lade zombier bevæge sig til sidst.

En anden løsning er at følge eksemplet fra de andre tilstandsændringer og udføre dem direkte efter en bevægelse, før næste turs lister opdateres.

På grund af deadlines er ingen af de ovenstående optimeringer dog nået med i koden, hvilket betyder de er gemt til første udvidelsespakke eller den uundgåelige efterfølger.[18]

Udvidelsesmuligheder

Vi har mange idéer til udvidelsesmuligheder, hvilket også er nævnt i afsnittet om overvejelser, men der er visse ting, der er mere realistiske at implementere i en given udvidelse.

Vi har allerede indlagt knapper til tre features, som vi ville have haft med i implementeringen, men måtte bukke under for tidspresset. Disse er *Paranoia*, *Militia* og *Breakable Buildings*. De tre var i vores *featurelist* indtil meget sent i processen, hvor vi desværre blev nødt til at skære dem fra.

Paranoia er, hvad vi kalder den effekt, at et menneske kan gå i panik ved at se et andet menneske i panik. Denne har vi haft implementeret i en konsoludgave, men den viste sig at være uinteressant, da mennesker konstant gik i panik. Featuren blev derfor nedprioriteret, og da GUI'en blev sammensat, var der ikke tid til at tilslutte denne feature.

Militia, er hvad vi kalder det, at soldater skyder alt andet end soldater – for nationens sikkerhed! Denne er relativt enkelt at implementere, men igen blev den nedprioriteret indtil, der ikke var tid til at få den med.

[18] 28 Minutes Later.

Breakable buildings siger sig selv og er den feature, vi er mest ærgerlige over ikke nåede med. Der er stadig rester af denne i koden, som f.eks. at *building* klassen definerer *breakpoints* og en relativt avanceret `damageThis()` metode, som beregner en ny farve til bygningen baseret på, hvor meget skade den har fået.

En fjerde udvidelse, som der også er koderester af rundt omkring, er statistik på antal aktører i simulationen. Det er en relativt triviel opgave at tilføje de pågældende tal i en liste og repræsentere dem på en måde efter simulationen er slut, og det er derfor ærgerligt, at denne feature ikke er med. Denne kunne være repræsenteret løbende under simulationen, hvilket blot ville være en anden repræsentation af tallene i informationsdelen i simulationen.

En sidste mere omfattende feature er, at vores baner er defineret som at have tomme felter på sorte områder og bygninger på grå, idet man kunne *male* en bane i et tegneprogram eller en *mapeditor*. Disse baner kunne gemmes i almindelige billedfiler, som senere kunne indlæses i simulationen og populeres. På samme metode kunne man lave en måde at gemme og indlæse simulationen på ved at skrive den alligevel RAM-lagrede *BufferedImage* til harddisken. Det er igen trivielt at lave en genkendelse af farver, til at fylde et *piece array* med passende *content*, som dog ville mangle eventuel intern hukommelse på indlæsnings-tidspunktet.

Refleksion over forløbet

Inception

Idéerne fra inceptionsfasen ændrede sig i senere elaborationsfasens iterationer, da vi foretog grundlæggende ændringer i forhold til systemet, hvilket er et eksempel på anvendeligheden af *Unified Process* og tilmed understreger, at inceptionsfasen ikke har til mål at fastlægge alle kravspecifikationer. Vi lavede allerede et udkast til en domænemodel i inceptionsfasen, hvilket viste sig at være noget optimistisk, da vi i elaborationsfasen måtte konstatere at mange af vores oprindelige idéer ikke var holdbare i forhold til vores vision, som løbene ændrede sig. Dette medførte tilmed, at inceptionsfasen og elaborationsfasen overlappede hinanden på visse områder, da vi måtte revidere dele af vores grundlæggende vision og brugsmønstre. Vi benyttede os i af Larmans princip om *timeboxing,* hvilket krævede megen disciplin, men det medførte gennem forløbet en konkret partiel målsætning for projektet. Det blev dog noget uoverskueligt at overholde disse i forbindelse med det nedprioriterede fokus på projektet, vi havde under arbejdsprocessen omkring et andet eksamensprojekt.

Vi har løbende revurderet vores idéer og overvejelser på grundlag af erfaring med at implementere disse samt indvirkningen fra nye idéer, og vi har gennem forløbet droppet mange af de oprindelige forestillinger. Derudover var vi som allerede nævnt nødsaget til at undlade en række udvidelsesmuligheder pga. tidsmæssige årsager.

Elaborationsfasen

I forlængelse af projektets anden iteration foretog vi en grundlæggende omstrukturering af en række af vores idéer, hvilket bevirkede, at vi måtte bevæge os tilbage til tankegangen fra første iteration af elaborationsfasen. Vi valgte, at droppe hele konceptet med zoneinddeling og i stedet definere, at bygningerne har en ramme på to tomme felter omkring sig, som løser problemet med, at de placeres oveni

hinanden, hvilket ville skabe aflukkede områder. De to tomme felter medførte, at væsenerne kunne gå imellem to bygninger på trods af, at de overlappede hinanden. Ulempen ved dette er dog, at der med denne population ikke bliver skabt en ligeså realistisk simulation af en by, da bygningerne ved populeringen ikke har en ligeså fast struktur. Beslutning var et resultat af, at implementeringen af denne idé ikke havde samme effekt som først antaget, da aspektet med gange imellem bygninger er tilsvarende interessant og mindre komplekst at implementere. Derudover valgte vi at opgive konceptet med tre forskellige typer af bygninger og i stedet blot definere en bygning ved en række vægge. Dette forenklede softwarestrukturen og bevirkede, at vi i stedet kunne fokusere på andre højrisikoområder i programmet. Disse omfattende ændringer gennem iterationerne er foretaget, da vi vurderede, at kompleksiteten af disse idéer ikke var overensstemmende med det udbytte, det ville medføre, og projektets tidsmæssige omfang var tilmed en begrænsning. Det var muligt at foretage ændringerne uden, at hele projektet skulle laves om pga. brugen af de forskellige iterationer. Derudover medførte det, at vi fik diskuteret en række muligheder igennem og dannede os et overblik over en række problemstillinger. Vi udformede igennem de første to iterationer nogle konkrete egenskaber for de forskellige aktører, som endte med at blive lidt for specifikke i den analytiske del af processen, hvor vi brugte for mange resurser på at udarbejde de enkelte detaljer i disse idéer. Dette afspejles i den endelige simulation, hvor disse egenskaber, på samme måde som idéerne omkring bygninger, er blevet simplificeret.

Ændringerne vi har foretaget i projektforløbet har løbene indvirket på vores domæne- og designmodeller, som er blevet radikalt ændret gennem den iterative proces. Som et eksempel på dette havde vi i vores første klassediagram en idé om, at de forskellige typer af væsener skulle nedarve fra en *Creature* klasse. Dette blev i anden iteration ændret til brugen af en statementklasse, hvilket understreger, hvorledes projektet løbene har ændret sig i de forskellige iterationer.[19]

GUI

Vi har tilmed løbene ændret vores GUI-prototype, som vi lavede et udkast til allerede i første iteration af elaborationsfasen. Vi nåede i slutningen af anden iteration frem til, at det oprindelige estimerede format på 800*600 var for stort og har i stedet sat selve simulationens maksimale størrelse til 600*600.[20] Udover at vores første estimerede størrelse af selve simulationen simpelthen fyldte for meget på skærmen, ville det kræve en betydelig mere effektiv kode for at kunne opnå en glidende fremstilling af simulationen. Endvidere erfarede vi, at der ikke er plads til den løbene information nedenunder selve simulationen, og vi har derfor valgt at placere information til højre for simulationen og derudover have de forskellige opsætningsmuligheder i en separat brugergrænseflade. Udviklingen af vores GUI prototyper har været en glidende proces, hvor vi ud fra vores overvejelser og erfaringer har justeret layoutet. Vi har hele tiden forsøgt at bruge os selv som potentielle brugere af simulationen og derudfra lave et GUI layout, som vi finder intuitivt.

Design og analyse

Vi diskuterede i starten af projektforløbet mange aspekter af simulationen igennem, inden vi påbegyndte kodeskrivningen, hvilket resulterede i, at vores analytiske del blev vægtet lidt for højt. Vi koncentrerede os meget om de enkelte detaljer i systemet, som f.eks. hvorledes de enkelte menneskers synsfelt og

[19] Se bilag XI
[20] Se bilag XV

bevægelsesmønster skulle defineres. Et resultat af dette var, at vi forholdsvis sent måtte ændre store dele af vores idégrundlag for systemet, og vi havde dermed en tildens til at forfalde til vandfaldsmodellen. Denne tendens skyldes efter vores opfattelse tilmed den manglende brugerinteraktion, som vi har forsøgt at simulere ved at sidestille os med projektet og agere kunder. Vi fik dog en bedre iterativ proces i tredje og fjerde iteration af elaborationsfasen, hvor vi i højere grad løbene fik testet kode og fik dermed fokus på højrisikoområderne. I denne sammenhæng var det tilmed en fordel, at vi gennem vores sekvensdiagrammer fik et kodenært fokus på de forskellige aktørers bevægelser og træk.

Refleksion over UP artefakter

Vi vil dele dette afsnit op i forhold til modeller og diagrammer og på den måde vil vi belyse artefakttyperne hver for sig og deres anvendelighed for gruppens projekt.

Brugsmønstre

Vi har lavet mange brugsmønstre gennem de forskellige iterationer, men artefaktet var dog hovedsageligt en hjælp i inceptionsfasen.

I denne fase bevirkede de, at vi fik overblik over en række grundlæggende handlinger i programmet, som f.eks. rangering af aktioner. Artefaktet har medført mange diskussioner om sådanne ting, hvilket ellers muligvis ikke var blevet belyst eller at hele kode processen var gået i stå for at finde en løsning på problemet. Vi udviklede brugsmønstrene i fællesskab, hvilket har skabt en fælles forståelse af programmet.

Uden disse mønstre ville der være høj risiko for ubalance i forholdet mellem soldat og zombie. Det er et eksempel på, at konversionerne om brugsmønstrene har været en vigtig del af vores inceptionsfase. Det gav os yderligere mulighed for at snakke om forskellige features, som kunne blive implementeret, og vi fandt i disse samtaler frem til undergrundsbaner, zone inddelinger og lignende detaljer.

Gloseliste

Fordelen ved en gloseliste, har været, at vi har fået fastlagt alle de usikkerheder, der har været om simulationens begreber. Som et eksempel kan en død zombie, bare være en død zombie, men de kan også blive inddelt efter hvordan de døde. Om de blev skudt eller døde, fordi de ikke havde spist eller smittet et menneske i et stykke tid.

Samtidigt med at give forståelse internt i gruppen, har vi også en komplet liste over ting, der enten er implementeret eller skal laves i en senere iteration.

Vi kunne godt have gjort lidt mere brug af dette artefakt, hvis vi tidligere havde haft en fuldt opdateret liste, ville vi have set at listen var omfattende.

Vision

Vi har haft en realisering i elaborationensfasen, der gjorde, at vi var nød til skære ned i vores visioner for simulationen. Det betød at ambitionerne blev mere realistiske i forhold til tidsrammen. Det vil sige, at vores første visionsliste, selvom den var lang og veludført blev skrottet. Så anden udførelse af blev en hurtigt gennemgang af hvad planen var for programmet.

Så egentlig brugte vi ikke visions liste direkte, men gennem klassediagrammer og arbejde foran tavlen med forskellige ideer tegnet og fortalt, kom vi frem til det endelige resultat.

Domænemodel

Domænemodellen siger ikke så meget i forhold til den egentlige implementering, da den indeholder nogle konceptuelle begreber. Domænemodellen har fået os i gang med at tænke på projektet, og på den måde er der kommet ideer om muligheder for systemet. Det har betydet, at vores liste med idéer har været lang og meget omfattende.

Sekvensdiagram

Vi har brugt sekvensdiagrammer til at lave relationer mellem de forskellige klasser, der bliver arbejdet ud fra en domænemodel. I domænemodellen kan vi se de konceptuelle klasser, og derefter arbejde med dem, så de bliver til mere kode nær handlinger, og det samarbejde mellem domænemodel og brugsmønstre har gjort at vores sekvensdiagrammer har haft større betydning i udviklingen af koden.

Sekvensdiagrammerne giver også mulighed for at finde de første muligheder for lav kobling og høj samhørighed. Fordelen ved sekvensdiagrammet er vi har fundet de datastrømninger mellem de forskellige klasser gennem arbejde med sekvensdiagrammet.

Systemsekvensdiagram

Ideen med et systemsekvensdiagram virker rigtig godt til systemer med stor brugerinteraktion. Vores system har lav påvirkning fra bruger, da den største del af systemet afvikler sig selv. Det betyder at udviklingen af systemsekvensdiagrammer har været en meget lille del af projektet. Derudover interagerer simulationen ikke med andre systemer, hvorved der generel ikke er meget at vise i et systemsekvens-diagram og vores fokus dermed ikke har lagt på dette artefakt.

Klassediagram

Vi har hele tiden arbejdet mod et komplet klassediagram, da vi oftest har startet med en domænemodel, men fik hurtigt lidt for meget informationer indsat, blev det mere et klassediagram end en domænemodel. Det har betydet at gruppen har lavet en række forskellige klassediagrammer, og at de har udviklet sig stille og roligt.

Vi fik glæde af klassediagrammet i forbindelse med, at vi overholdte designmønstre, det gav for eksempel et rigtig godt overblik over Model-View-Control, så vi kunne se om, det blev overholdt eller om, der skulle ændres i pakkestrukturen. Hvis der ikke blev lavet et klassediagram, har gruppen tendens til at glemme sådanne mønstre og lave klasser med rigtig mange metoder, hvilket vi erfarede i forbindelse med et tidligere projekt i kurset, som omhandlede et adventure spil.

Fordi alle klasser i *Model* delen er vist i klassediagrammet, giver det også et klart billede af GRASP, og det har lagt op til diskussioner om placering af metoder i de forskellige klasser. Den problematik er belyst ved *buildingCreator* klassen, den har skiftevis haft sin egen klasse og blevet implementeret i andre klasser som en metode.

Konklusion

Vi har i dette projekt tilvejebragt et program, der simulerer sygdomsspredning i en verden befolket med mennesker, soldater og zombier. Vi kan reflekterende konkludere, at vi brugte for mange resurser på systemspecifikke overvejelser, hvor de enkelte væsners synsfelt og bevægelsesmønster kan nævnes som et eksempel, og vi har generelt haft for store ambitioner for projektet i forhold til dets størrelse, hvilket har bevirket adskillige nedjusteringer af vores vision gennem forløbet. Omvendt har ambitionsniveauet samt den store vægtning af simulationens realisme, og de dertilhørende relativt komplicerede idéer fungeret som grobund for det produkt, vi har fået i sidste ende. Effekten af opdelingen i iterationer har bevirket, at vi har sparet tid på mange områder, da iterationerne har medført, at vi løbende har foretaget ændringer, ud fra Larmans idé om *proof-of-concept,* uden at hele koden skulle skrives om. Iterationerne, og de målsætninger vi har haft i disse, har tilmed medført, at vi adskillige gange har stoppet op og dannet et overblik over projektets omfang og vurderede kvaliteten af vores softwarestruktur. Det færdige program udgøres af to grafiske brugergrænseflader, som henholdsvis tager imod input fra en bruger og opsætter en simulation, som illustrerer spredningen af zombiesygdommen.

Litteraturliste

Larman, C. (2006). *Applying UML And Patterns.* Westford: Pearson Education, Inc.

Lewis, J., & Loftus, W. (2007). *java Software solutions.* Pearson Education, Inc.

Bilag I

Navn på brugsmønster	**Bruger kører simulation**
Scope	Systemet under udvikling
Level	User goal
Primary Actor	Bruger
Stakeholders and Interests	Bruger vil starte systemet og se simulationen.
Preconditions	Javas virtuelle maskine skal være installeret til styresystemet. Brugeren skal vælge indstillinger for simulationen.
Sucess Guarantee	Brugeren skal vælge de korrekte indstillinger.
Main Success Scenario	Simulationen populeres korrekt. Simulationen skal afsluttes korrekt - zombie eller de andre vinder.
Extensions	Programmet bryder ned. Programmet lukkes. Der vælges forkerte indstillinger.
Special Requirements	Kompatibelt Java styresystem.
Technology and Data Variations List	Vi har observeret en forskel på fra Java til Mac OS X til Windows XP**.**
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Bruger vælger indstillinger**
Scope	Systemet under udvikling
Level	User goal
Primary Actor	Bruger
Stakeholders and Interests	Bruger vil vælge indstillinger for simulationen.
Preconditions	Brugerens indtastede indstillinger må ikke overskrive det maksimale antal units.
Sucess Guarantee	Brugeren indtaster lovlige indstillinger.
Main Success Scenario	Simulationen er klar til at blive startet.
Extensions	Brugeren vælger forkerte indstillinger.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Menneske ser zombie**

Scope	Simulationen under udvikling
Level	User goal
Primary Actor	Menneske
Stakeholders and Interests	Menneske vil undgå at blive smittet. Menneske går i panik.
Preconditions	Simulationen skal være i gang . Der skal være en zombie indenfor menneskets synsvinkel.
Sucess Guarantee	Mennesket ser en zombie.
Main Success Scenario	Mennesket ser en zombie. Mennesket undgår at blive smittet.
Extensions	
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Zombie smitter menneske eller soldat**
Scope	Systemet under udvikling
Level	User goal
Primary Actor	Zombie
Stakeholders and Interests	Zombie: vil smitte menneske eller soldat. Andre zombier: skal ikke smitte den samme. Menneske: vil flygte fra zombie. Soldat: vil dræbe zombie.
Preconditions	Simulationen skal være i gang. Zombien skal være indenfor smitte-rækkevidde.
Sucess Guarantee	Zombien skal være indenfor smitte-rækkevidde. Zombien smitter før den bliver dræbt
Main Success Scenario	Zombie bevæger sig indenfor rækkevidde. Mennesket eller soldaten omdannes til zombie. Zombie bevæger sig indenfor rækkevidde. Mennesket eller soldaten omdannes til zombie.
Extensions	Flere mennesker/soldater indenfor rækkevidde. En vælges vilkårligt Zombie er lige blevet smittet Kan ikke smitte med det samme (samme tur)
Special Requirements	Actions (zombie smitter, soldat dræber/giver våben) foregår asynkront: 1) Zombie 2) Soldat dræb ELLER soldat våben
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Soldat dræber zombie**

Scope	Systemet under udvikling
Level	User goal
Primary Actor	Soldat
Stakeholders and Interests	Zombie vil smitte soldat. Soldat vil dræbe zombie. Mennesker flygter fra zombie
Preconditions	Simulationen skal være i gang Soldaten skal være indenfor skudrækkevidde af zombien. Skal have set zombien.
Sucess Guarantee	Soldaten skal være indenfor skudrækkevidde af zombien. Ingen zombie er indenfor smitterækkevidde af soldaten.
Main Success Scenario	Ser zombien fra "lang" afstand. Bevæger sig imod zombien indtil indenfor skudrækkevidde. Skyder zombien.
Extensions	Flere zombier indenfor rækkevidde En vælges vilkårligt Soldat har lige fået våben Kan ikke skyde med det samme (samme tur)
Special Requirements	Actions (zombie smitter, soldat dræber/giver våben) foregår asynkront: 1) Zombie 2) Soldat dræb ELLERsoldat giv våben.
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Soldat giver våben til menneske**
Scope	Systemet under udvikling
Level	User goal
Primary Actor	Soldat
Stakeholders and Interests	Soldat: Vil give våben til menneske (lave ny soldat) Menneske: Følger sin egen plan, men accepterer at modtage et våben, og blive soldat.
Preconditions	Simulationen skal være i gang. Soldaten skal være indenfor "give våben" rækkevidde af mennesket. Skal have set mennesket.
Sucess Guarantee	Soldaten skal være indenfor "give våben" rækkevidde af mennesket. Ingen zombie skal være indenfor skudrækkevidde. Ingen zombie er indenfor smitterækkevidde af mennesket/soldaten. Ingen zombie er indenfor smitterækkevidde af soldaten.
Main Success Scenario	Ser mennesket fra lang afstand. Bevæger sig imod mennesket. Giver våben til mennesket.
Extensions	Flere mennesker indenfor rækkevidde. En vælges vilkårligt.
Special Requirements	Actions (zombie smitter, soldat dræber/giver våben) foregår asynkront: 1) Zombie 2) Soldat dræb ELLERsoldat giv våben.
Technology and Data Variations List	

Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Zombie ser menneske**
Scope	Simulationen under udvikling
Level	User goal
Primary Actor	Zombie
Stakeholders and Interests	Zombie vil smitte menneske eller soldat. Zombie bevæger sig imod menneske.
Preconditions	Simulationen skal være i gang. Der skal være mennesker tilbage i simulationen. Zombie: skal ikke være i gang med at smitte et andet menneske eller en soldat.
Sucess Guarantee	Der skal være et menneske indenfor zombiens synsfelt.
Main Success Scenario	Et menneske bevæger sig indenfor zombiens synsfelt. Zombien ser et menneske
Extensions	Flere mennesker/soldater indenfor rækkevidde – her ser zombien alle og vælger den nærmeste.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Menneske foretager ryk**
Scope	Simulationen under udvikling
Level	User goal
Primary Actor	Zombie
Stakeholders and Interests	Zombie vil smitte menneske. Menneske vil flygte fra zombie. Menneske vil overleve.
Preconditions	Simulationen skal være i gang. Der skal være mennesker tilbage i simulationen.
Sucess Guarantee	Mennesket dræbes ikke af en zombie. Menneske er ikke omgivet af objekter.
Main Success Scenario	Mennesket bevæger sig til et nyt felt.
Extensions	Menneske kan ikke rykke pga. et statisk objekt. Menneske dræbes.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Menneske foretager ryk**
Scope	Simulationen under udvikling
Level	User goal
Primary Actor	Zombie
Stakeholders and Interests	Zombie vil smitte menneske eller soldat. Menneske vil flygte fra zombie. Soldat vil dræbe zombie.
Preconditions	Simulationen skal være i gang. Zombien skal have levetid tilbage til et ryk. Zombien må ikke være omgivet af statiske objekter.
Sucess Guarantee	Zombien har mulighed for at bevæge sig. Zombien smitter før den bliver dræbt.
Main Success Scenario	Zombien flytter sig til et andet felt.
Extensions	Menneske kan ikke rykke pga. et statisk objekt. Menneske dræbes.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

Navn på brugsmønster	**Zombie smitter menneske eller soldat (revideret)**
Scope	Simulationen under udvikling
Level	User goal
Primary Actor	Zombie
Stakeholders and Interests	Zombie vil smitte menneske eller soldat. Menneske vil flygte fra zombie. Soldat vil dræbe zombie.
Preconditions	Simulationen skal være i gang. Zombien skal være indenfor smitte-rækkevidde.
Sucess Guarantee	Zombien skal være indenfor smitte-rækkevidde. Zombien smitter før den bliver dræbt.
Main Success Scenario	Zombie bevæger sig indenfor rækkevidde. Mennesket eller soldaten omdannes til zombie.
Extensions	Zombien skal have levetid tilbage til et ryk.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

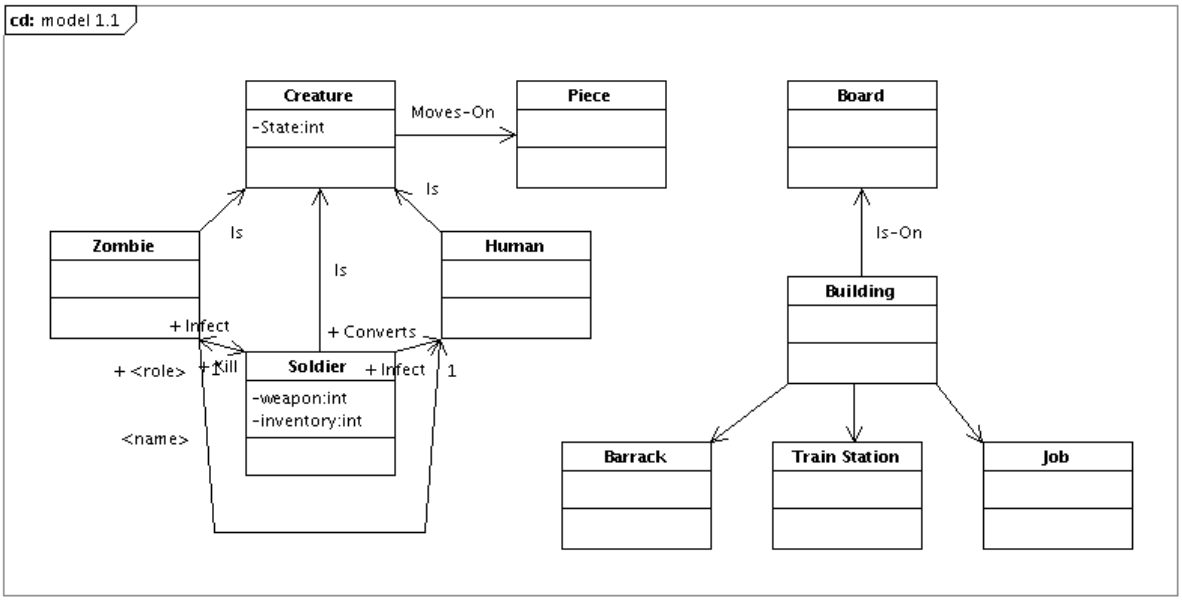
Navn på brugsmønster	**Soldat dræber zombie(revideret)**
Scope	Simulationen under udvikling

Level	User goal
Primary Actor	Soldat
Stakeholders and Interests	Zombie vil smitte soldat. Soldat vil dræbe zombie. Mennesker flygter fra zombie.
Preconditions	Simulationen skal være i gang. Soldaten skal være indenfor skudrækkevidde af zombien. Skal have set zombien.
Sucess Guarantee	Soldaten skal være indenfor skudrækkevidde af zombien. Ingen zombie er indenfor smitterækkevidde af soldaten.
Main Success Scenario	Ser zombien. Bevæger sig imod zombien indtil indenfor skudrækkevidde Skyder zombien.
Extensions	Flere zombier indenfor rækkevidde -nærmeste vælges.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	
Miscellaneous	

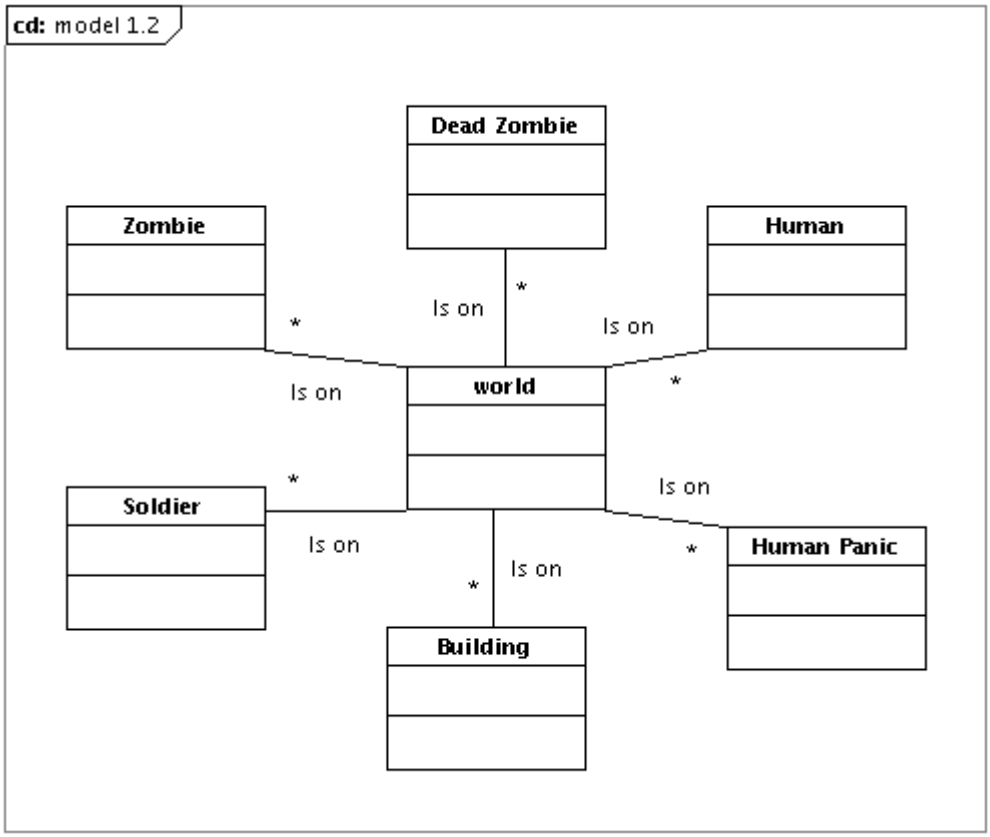
Bilag II - Gloseliste

Termer	Definition og information
Zombie	Illustreret med grøn farve i GUI og forsøger at smitte mennesker. De går med halv hastighed af mennesker.
Soldier	Illustreret med blå farve i GUI og forsøger at dræbe zombier.
Human	Illustreret med rød farve og går hele tiden efter et mål.
HumanPanic	Illustreret med gul farve og er en tilstand som mennesker indtager når de ser en zombie. Deres synsfelt bliver reduceret og deres hastighed øges til det tredobbelte.
Building	Et statisk objekt som aktørerne ikke kan gå igennem. Placeres tilfældigt i simulationen.
Aktør	Dækker over zombier, mennesker, mennesker i panik og soldater
Piece	Et felt i simulationen svarende til en pixel, hvor aktører kan bevæge sig på og Buildings kan være
ContentState	En klasse der giver mulighed for at de forskellige aktører kan skifte tilstand.

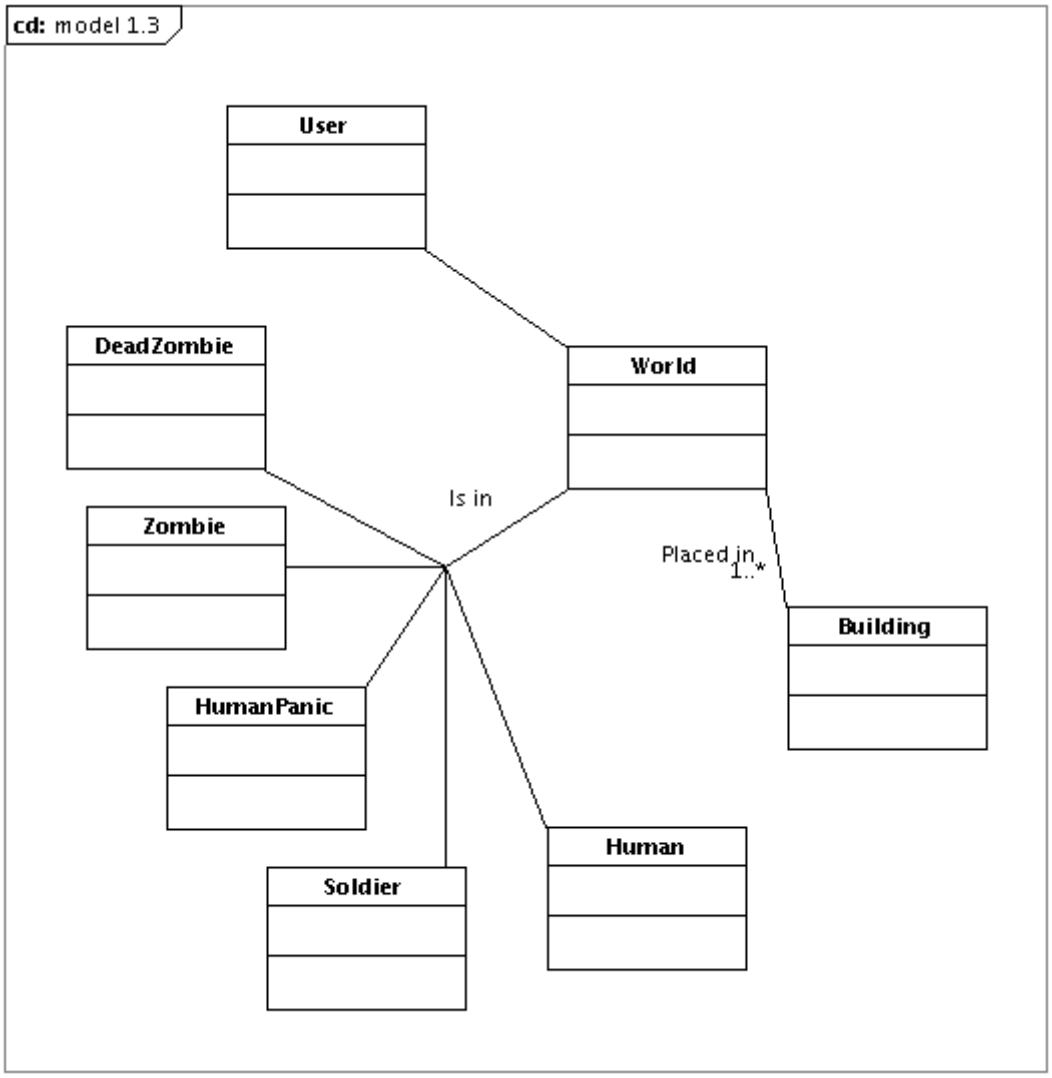
Bilag III – Domænemodel



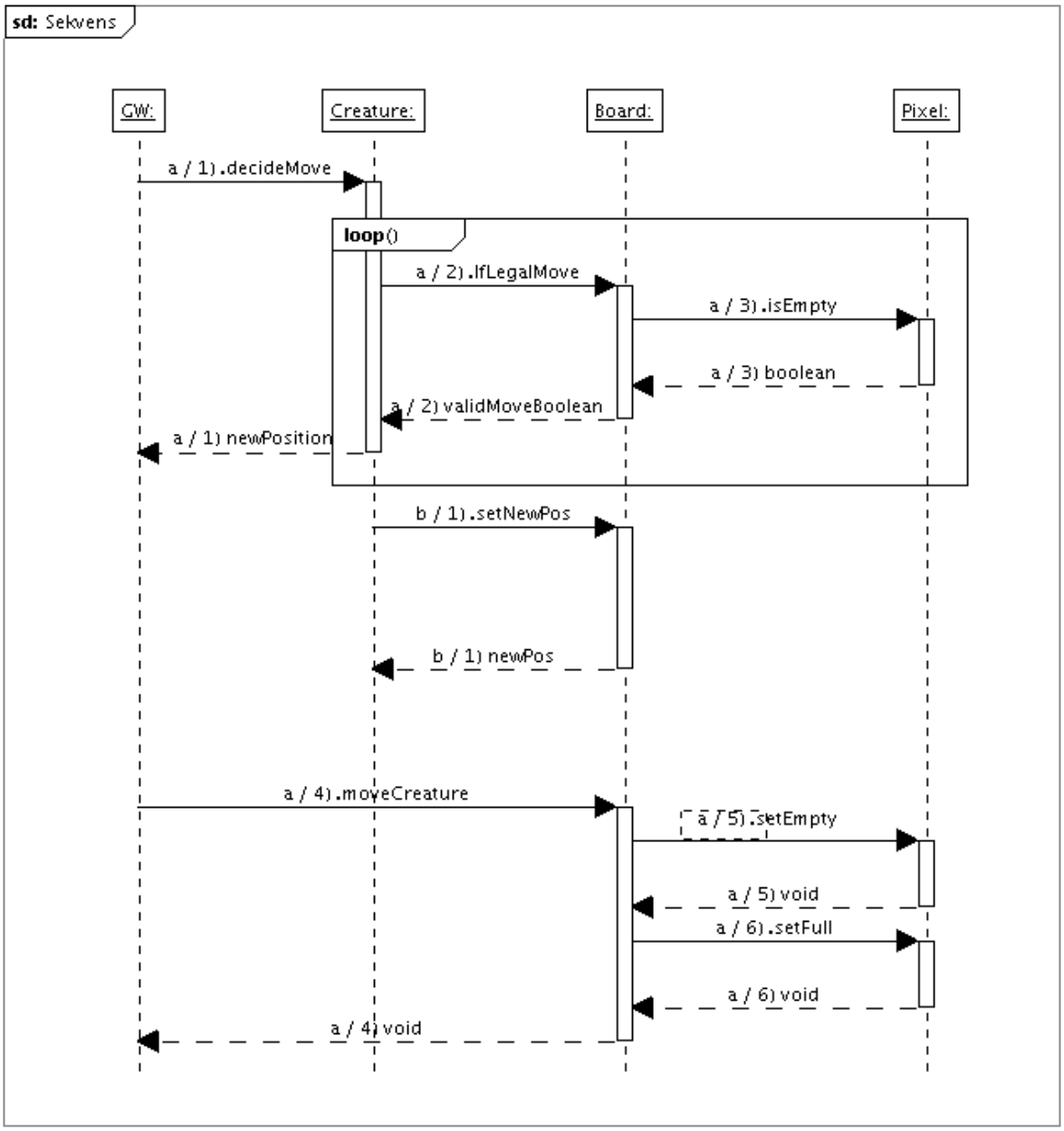
Bilag IV – Domænemodel



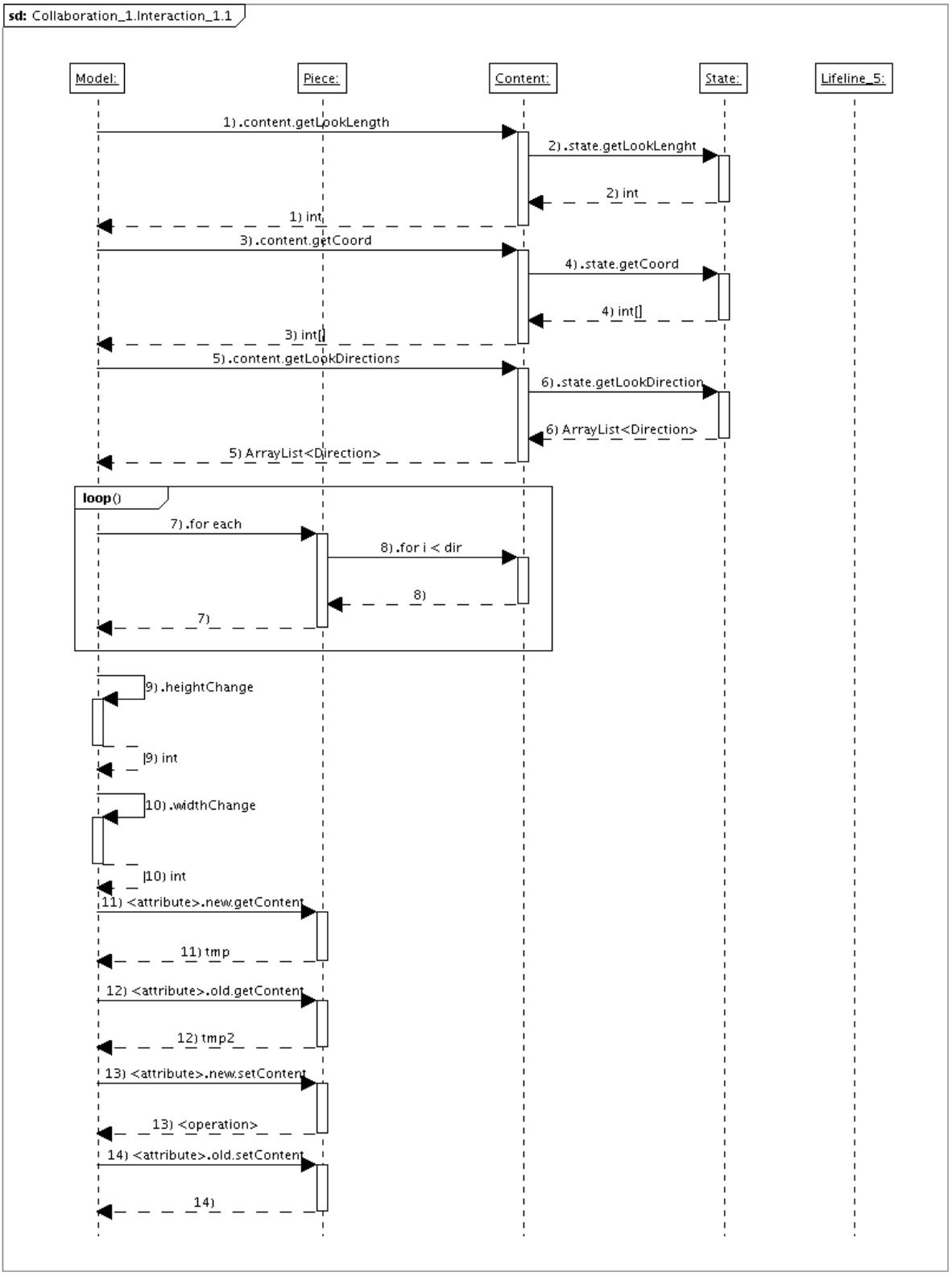
Bilag V - Domænemodel



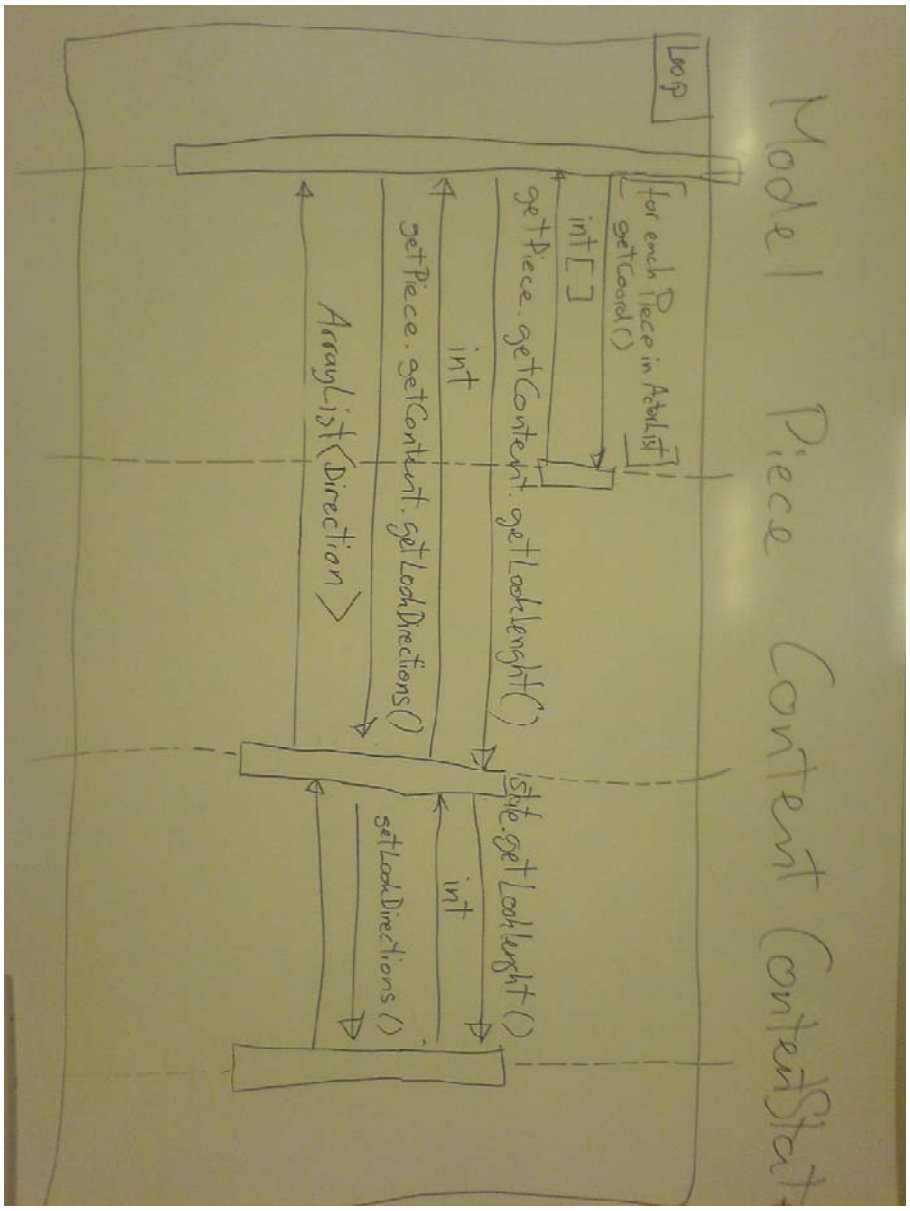
Bilag VI


sd: Sekvens
GW:
Creature:
Board:
Pixel:
a / 1).decideMove
loop()
a / 2).IfLegalMove
a / 3).isEmpty
a / 3) boolean
a / 2) validMoveBoolean
a / 1) newPosition
b / 1).setNewPos
b / 1) newPos
a / 4).moveCreature
a / 5).setEmpty
a / 5) void
a / 6).setFull
a / 6) void
a / 4) void

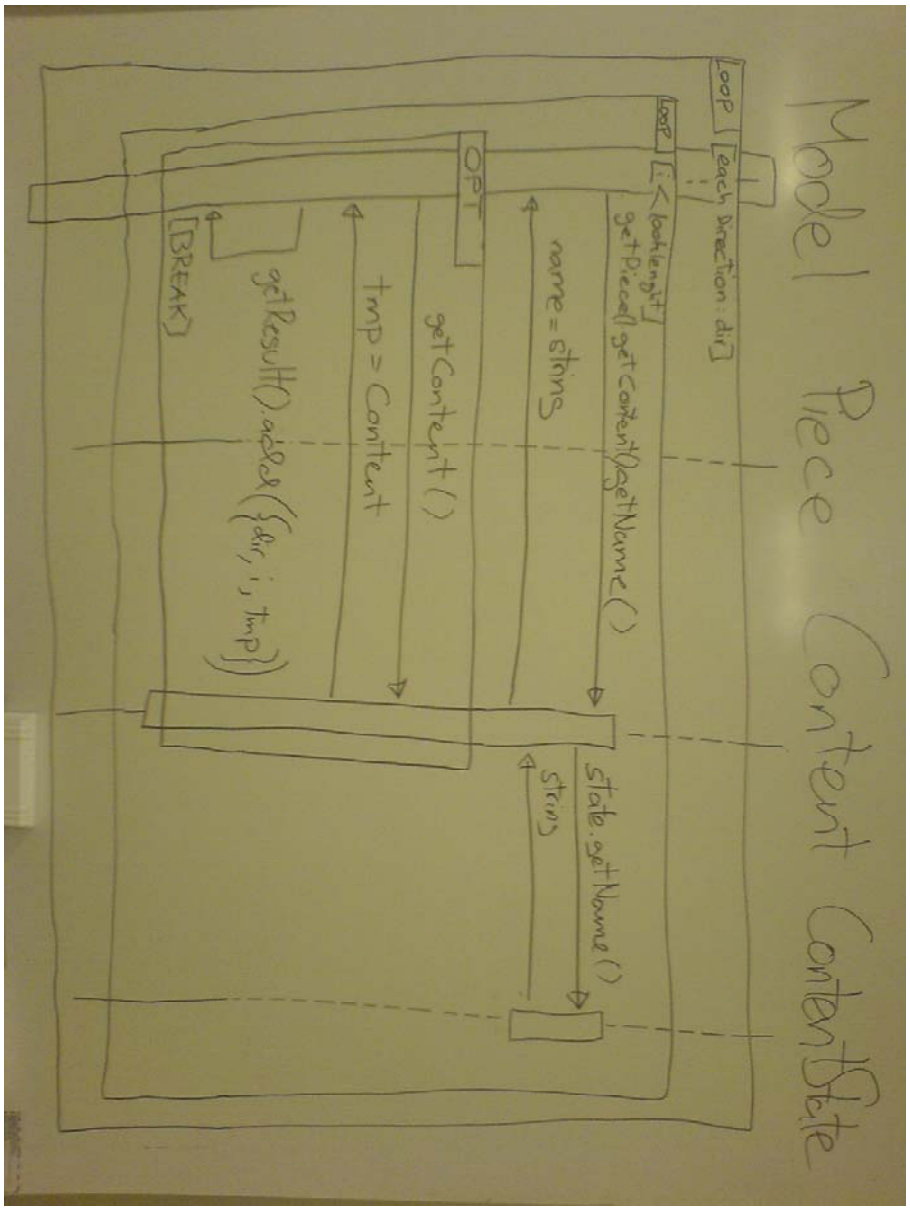
Bilag VII


sd: Collaboration_1.Interaction_1.1
Model:
Piece:
Content:
State:
Lifeline_5:
1) .content.getLookLength
2) .state.getLookLenght
2) int
1) int
3) .content.getCoord
4) .state.getCoord
4) int[]
3) int[]
5) .content.getLookDirections
6) .state.getLookDirection
6) ArrayList<Direction>
5) ArrayList<Direction>
loop()
7) .for each
8) .for i < dir
8)
7)
9) .heightChange
9) int
10) .widthChange
10) int
11) <attribute>.new.getContent
11) tmp
12) <attribute>.old.getContent
12) tmp2
13) <attribute>.new.setContent
13) <operation>
14) <attribute>.old.setContent
14)

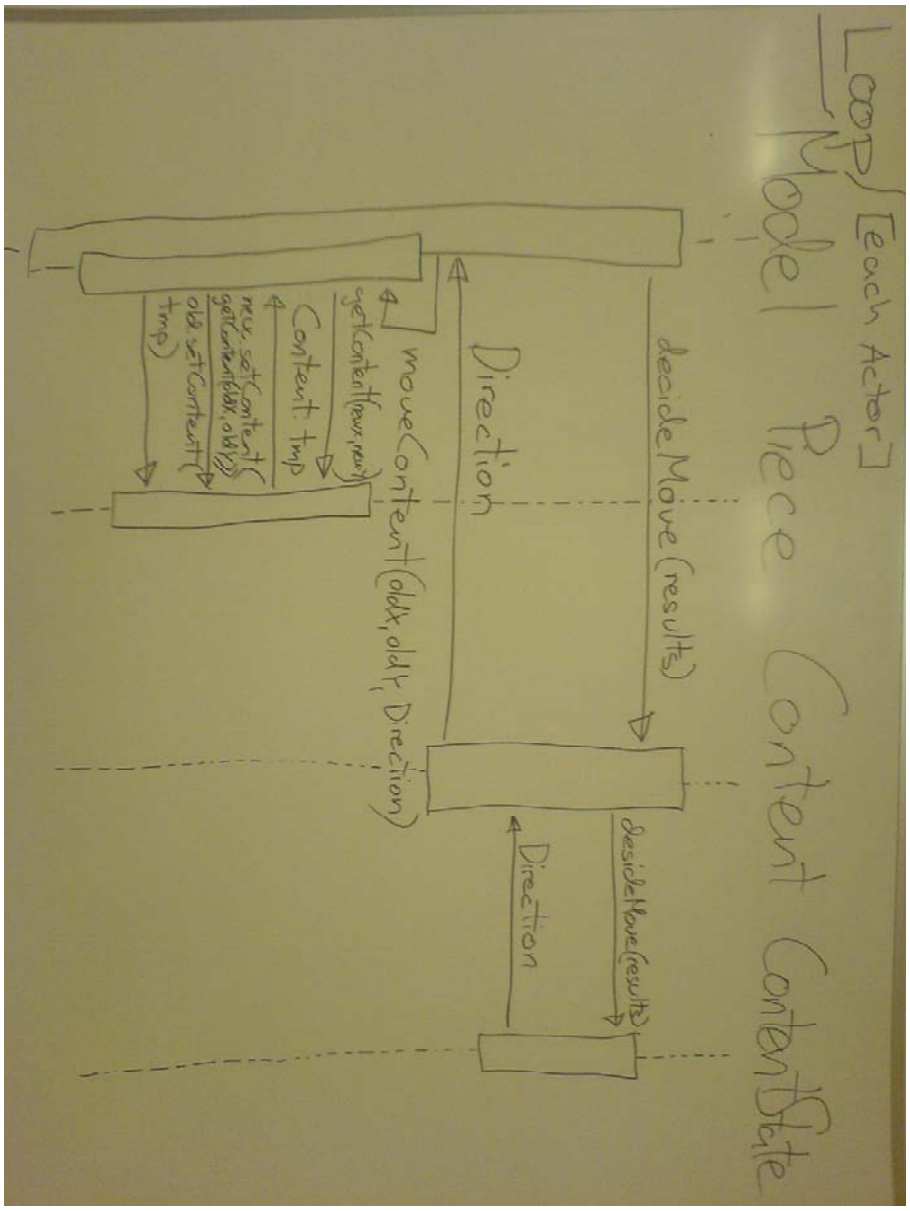
Bilag VIII


Model Piece Content ContentStat
Loop
for each Piece in ActorList
getCoord()
int[]
getPiece.getContent.getLookLenght()
int
getPiece.getContent.getLookDirections()
ArrayList<Direction>
State.getLookLenght()
int
setLookDirections()

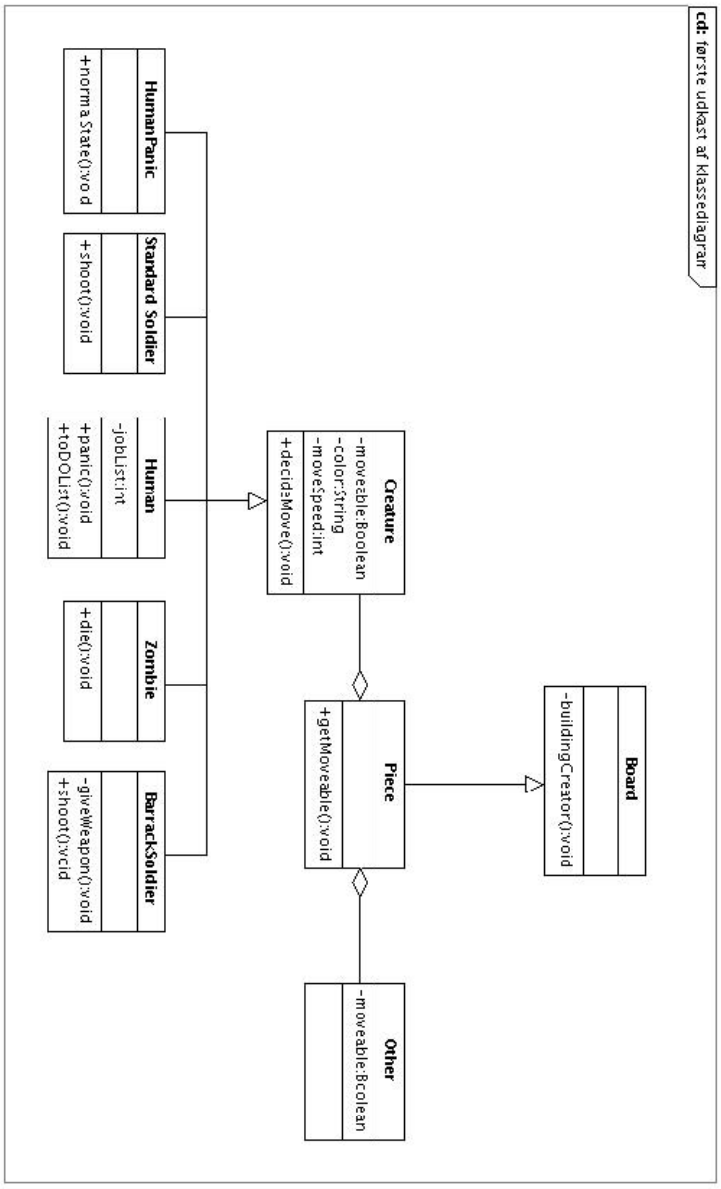
Bilag IX


Model Piece Content ContentState
LOOP
each Direction: dir
LOOP
i < lookLength
getPiece() getContent()/getName()
OPT
name = skins
getContent()
tmp = Content
getResult().add({dir, i, tmp})
[BREAK]
State.getName()
skins

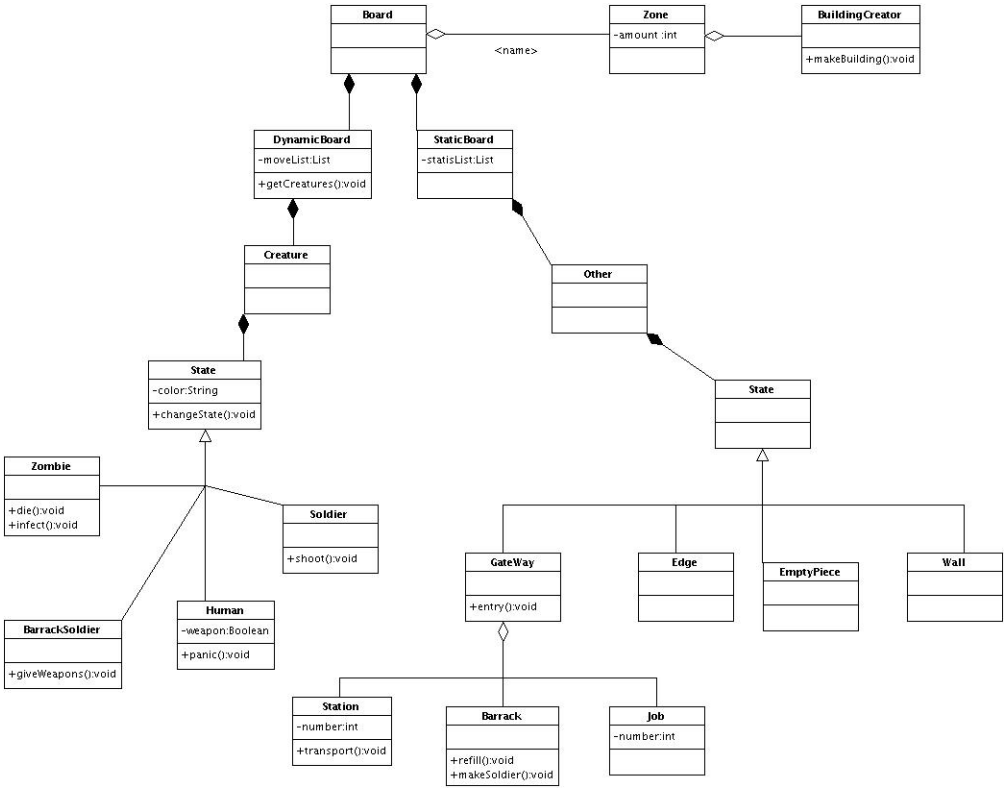
Bilag X


Loop [each Actor]
Model
Piece
Content
ContentState
decideMove(results)
Direction
moveContent(oldX, oldY, Direction)
getContent(newX, newY)
Content tmp
new.setContent(getContent(oldX, oldY))
old.setContent(tmp)
Direction
decideMove(results)

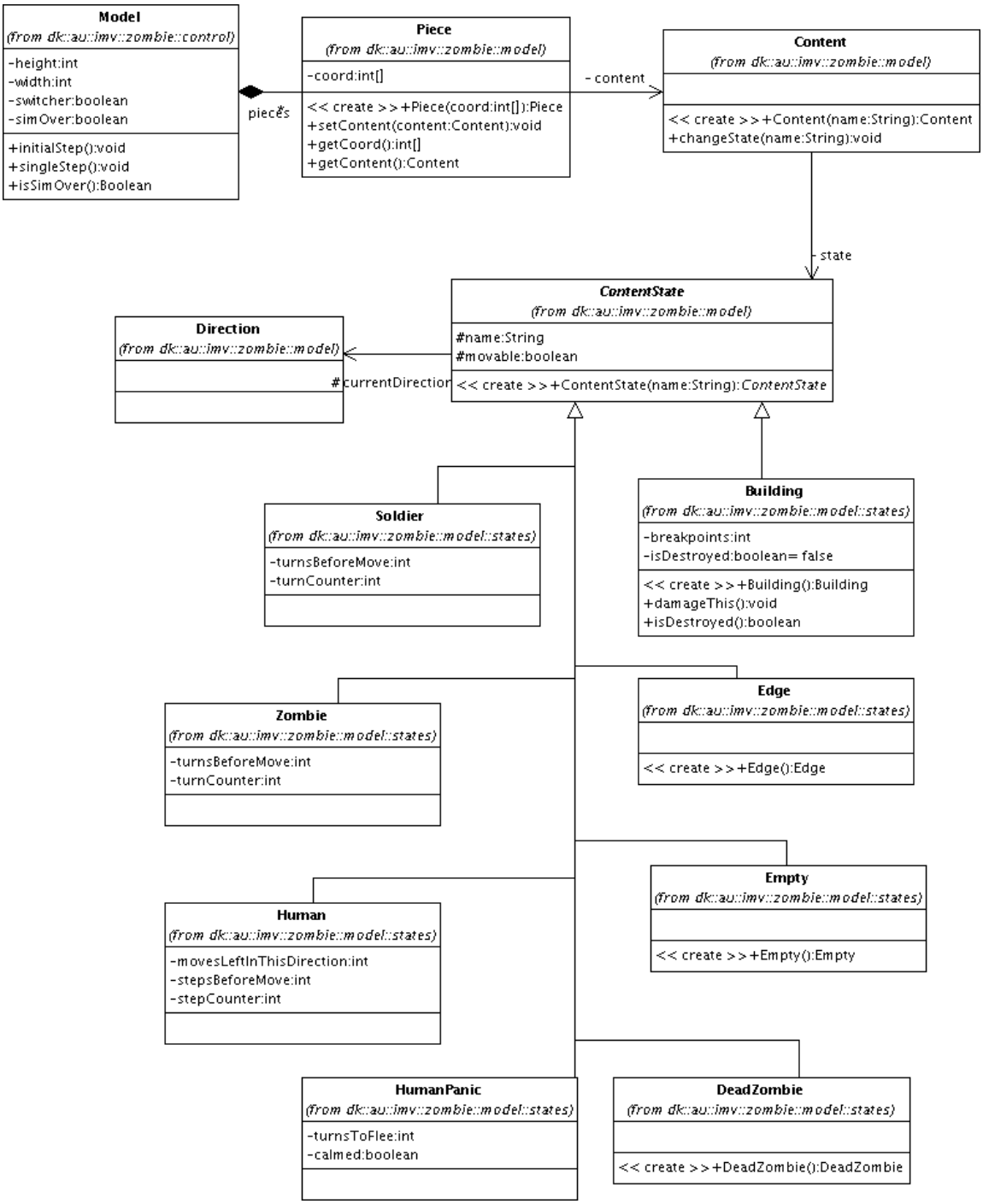
Bilag XI


cd: første udkast af klassediagram
HumanPanic
+normalState():void
Standard Soldier
+shoot():void
Human
-jobList:int
+panic():void
+toDOList():void
Zombie
+die():void
BarrackSoldier
-giveWeapon():void
+shoot():vcid
Creature
-moveable:Boolean
-color:String
-moveSpeed:int
+decideMove():void
Piece
+getMoveable():void
Board
-buildingCreator():void
Other
-moveable:Bcolean

Bilag XII


Board
<name>
Zone
-amount :int
BuildingCreator
+makeBuilding():void
DynamicBoard
-moveList:List
+getCreatures():void
StaticBoard
-statisList:List
Creature
Other
State
-color:String
+changeState():void
State
Zombie
+die():void
+infect():void
Soldier
+shoot():void
GateWay
+entry():void
Edge
EmptyPiece
Wall
BarrackSoldier
+giveWeapons():void
Human
-weapon:Boolean
+panic():void
Station
-number:int
+transport():void
Barrack
+refill():void
+makeSoldier():void
Job
-number:int

Bilag XIII – Endeligt klassediagram



Bilag XXIX

STEPS	1	2	3	4	5	6	7	8	9	10	11	12
HP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
H_1	✓			✓			✓			✓		
H_2		✓			✓			✓			✓	
S			✓			✓			✓			✓
Z_1	✓						✓					
Z_2			✓						✓			

HP = HumanPanic
H = Human
S = Soldier
Z = Zombie

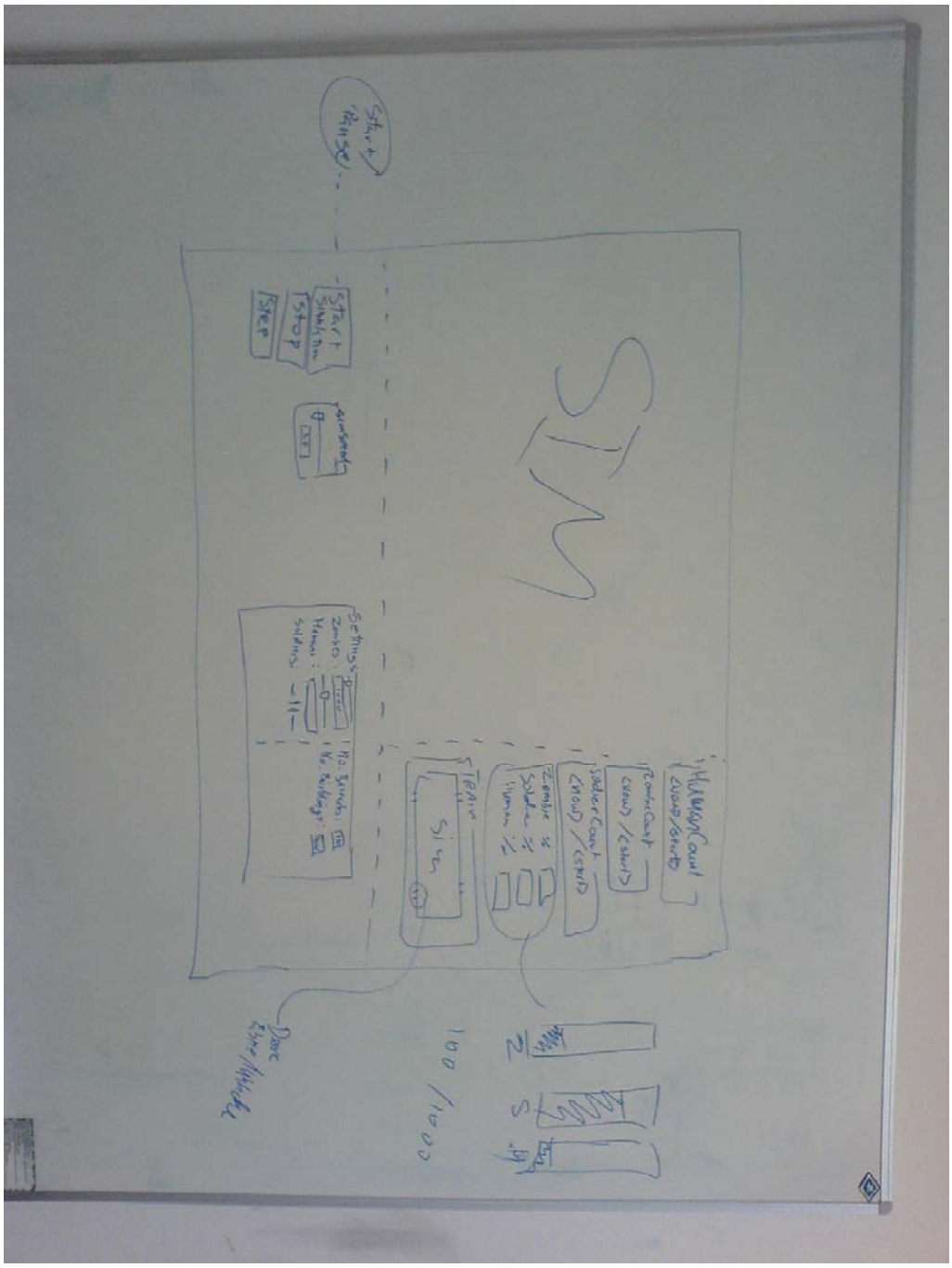
HpStart = 1

Zstart = Vilkårlig [1-6]

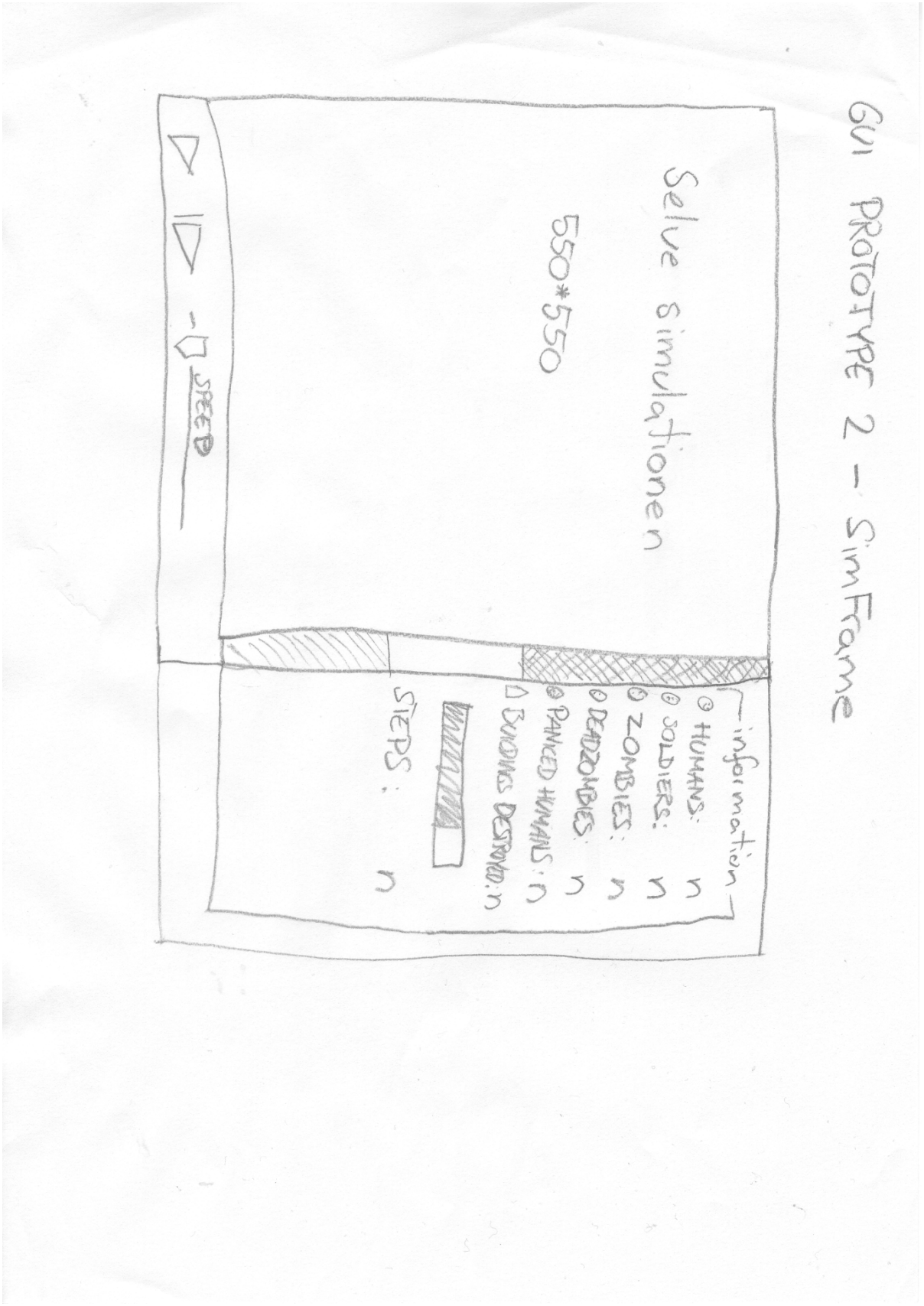
Hstart = Vilkårlig [1-3]

Sstart = Vilkårlig [1-3]

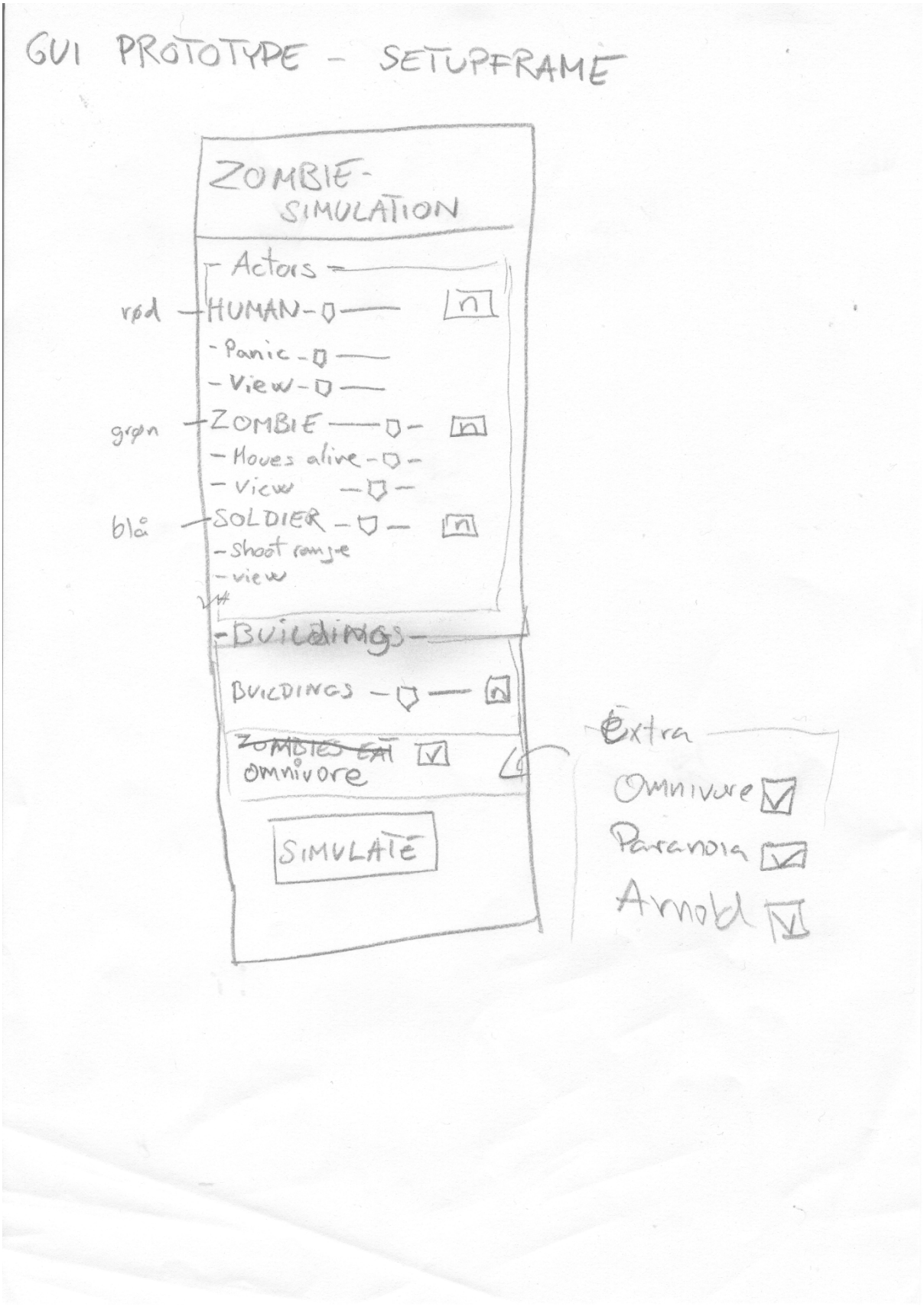
Bilag XV


SIMS
Start
Pause
Stop
Settings
Zombies
Humans
Buildings
Siren
Zombie %
Human %
Shelter Count
Human Count
Zombie Count
100 / 100

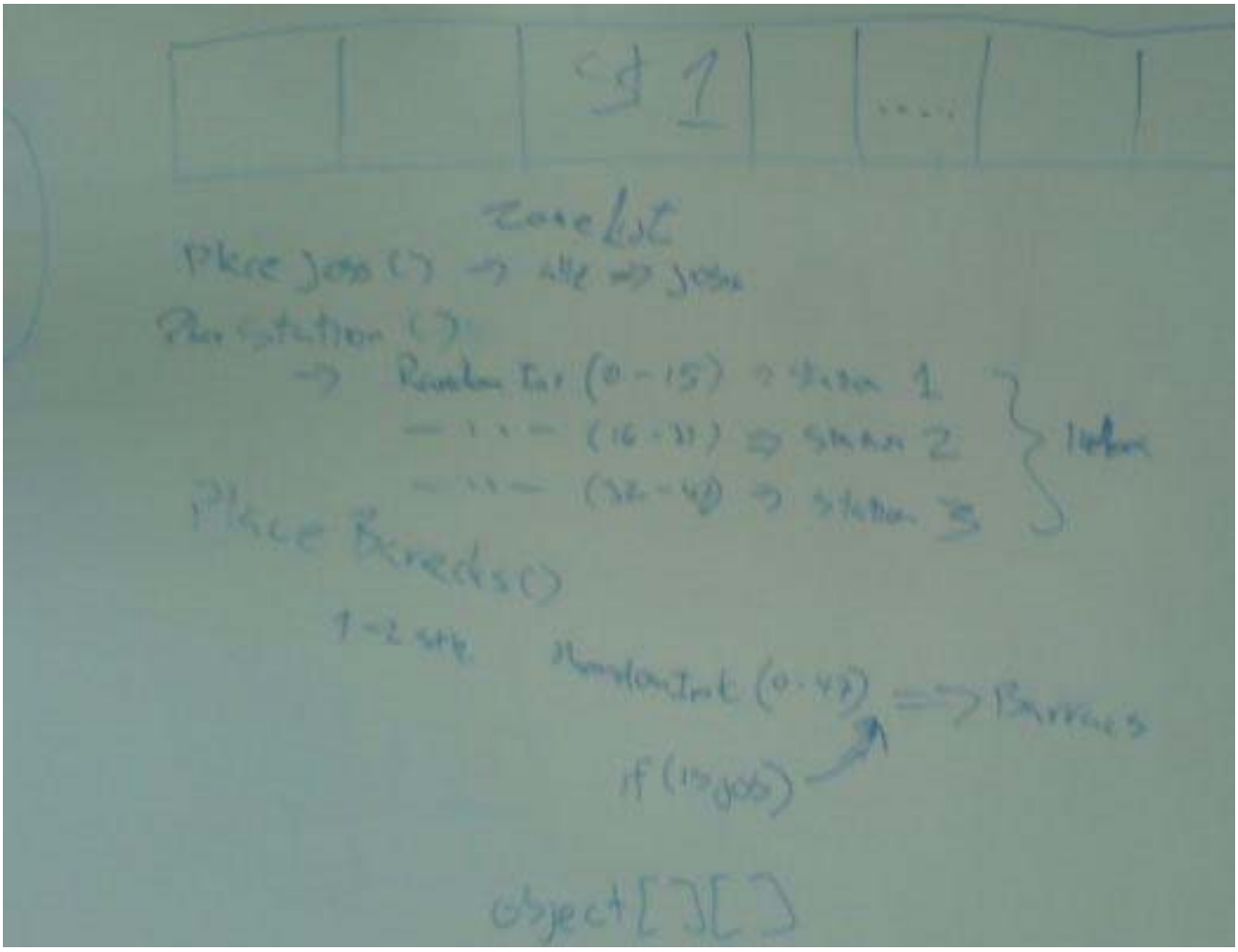
Bilag XVI

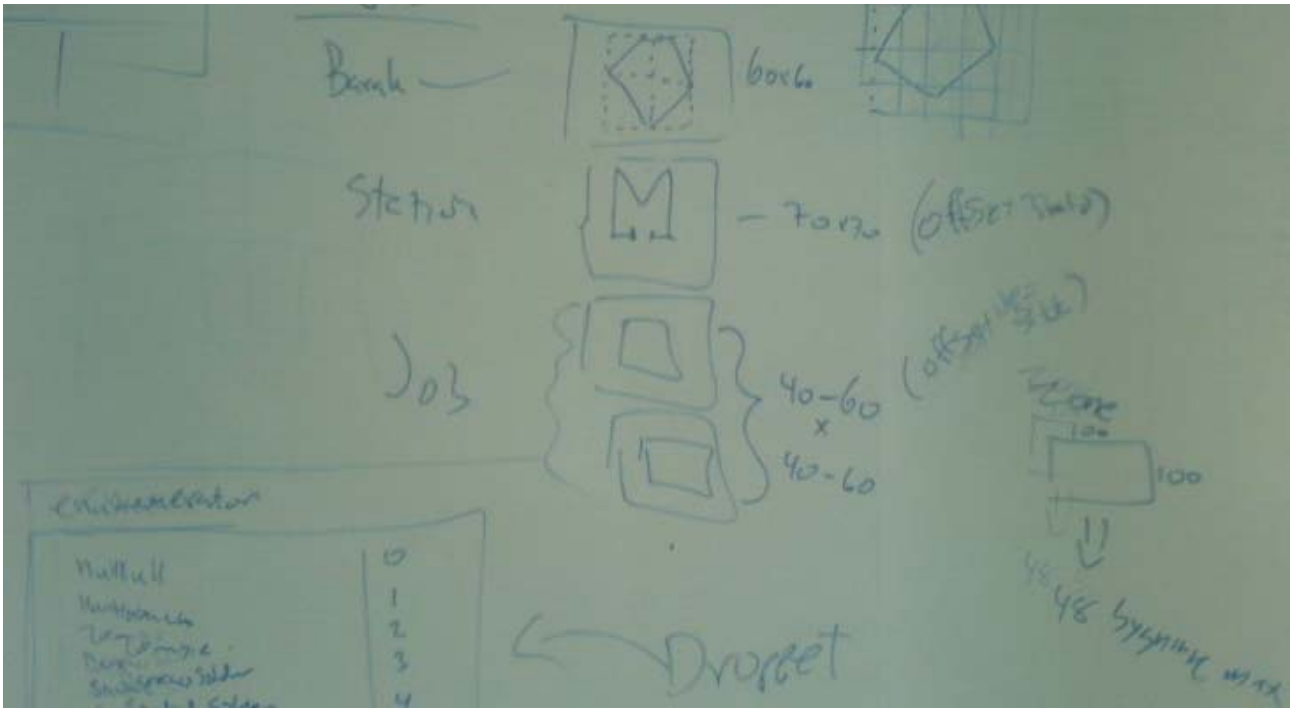

GUI PROTOTYPE 2 - SimFrame
Selve simulationen
550*550
SPEED
information
HUMANS: n
SOLDIERS: n
ZOMBIES: n
DEADZOMBIES: n
PANICED HUMANS: n
BUILDINGS DESTROYED: n
STEPS: n

Bilag XVII

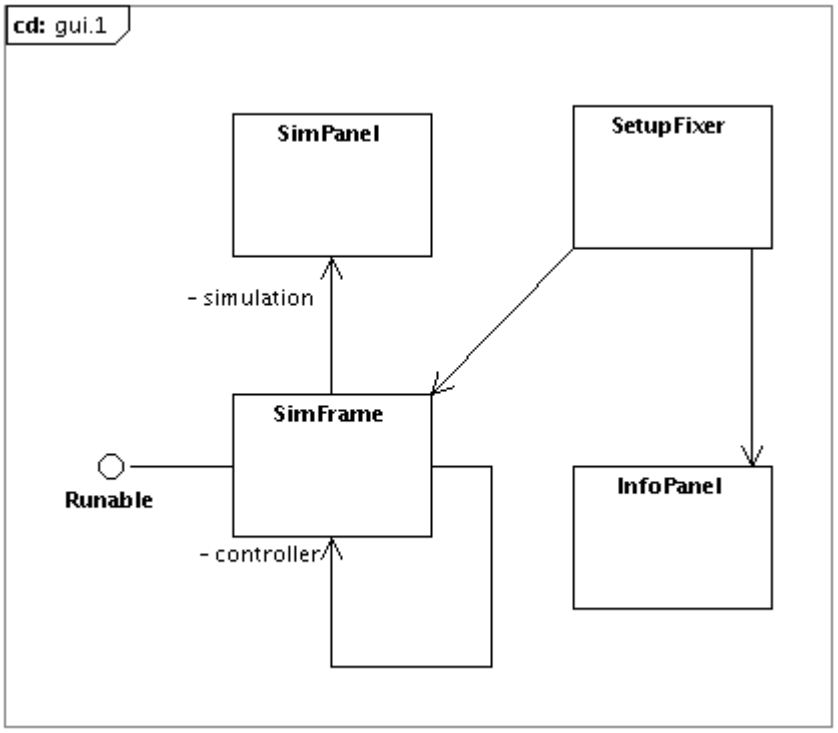

GUI PROTOTYPE - SETUPFRAME
ZOMBIE-SIMULATION
Actors
röd
HUMAN
- Panic
- View
grön
ZOMBIE
- Moves alive
- view
blå
SOLDIER
- shoot ranje
- view
BUILDINGS
BUILDINGS
ZOMBIES EAT
omnivore
SIMULATE
Extra
Omnivore
Paranoia
Arnold

Bilag XIIX


Place Barracks()
object[][]

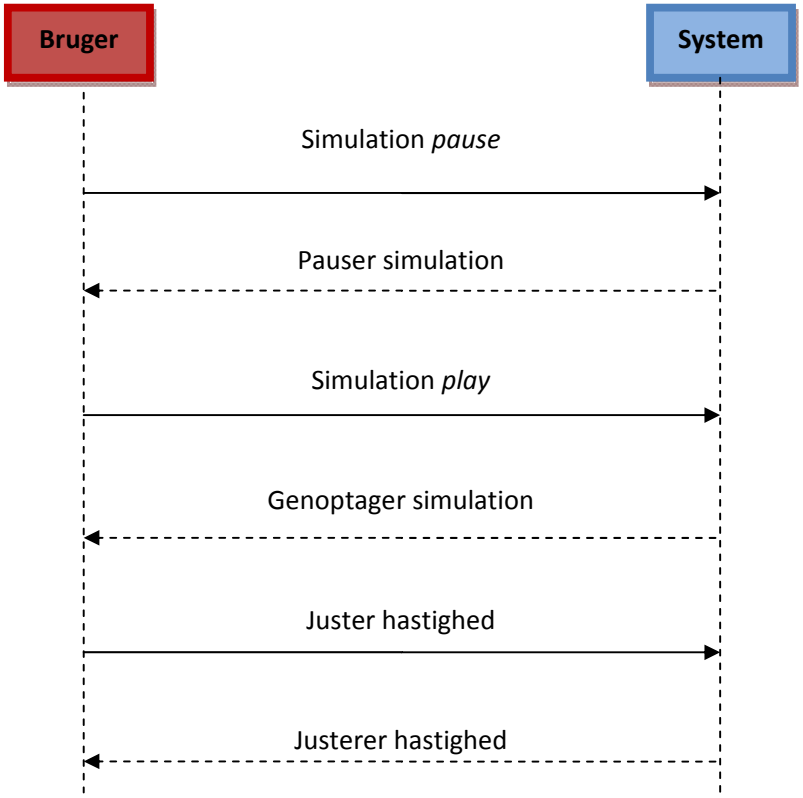

Bank
60x60
Station
40-60
x
40-60
Droppet

Bilag IXX

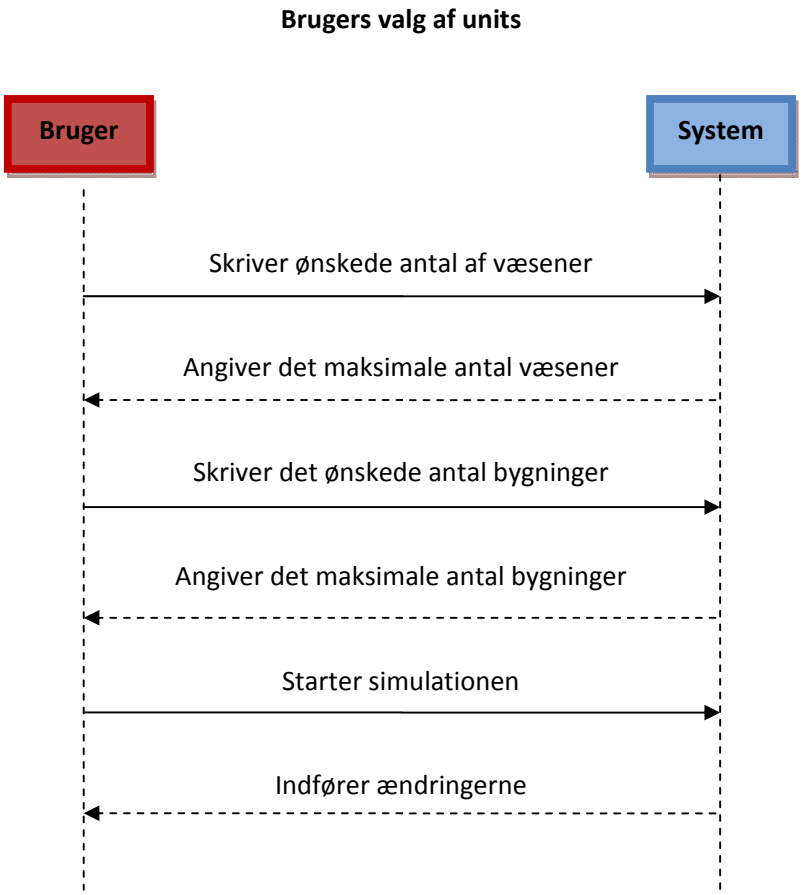

cd: gui.1
SimPanel
SetupFixer
- simulation
SimFrame
Runable
- controller
InfoPanel

Bilag XX

Brugerinteraktion med simulationen i proces



Bilag XXI


Brugers valg af units
Bruger
System
Skriver ønskede antal af væsener
Angiver det maksimale antal væsener
Skriver det ønskede antal bygninger
Angiver det maksimale antal bygninger
Starter simulationen
Indfører ændringerne