Informationsteknologi: Projekt e-læring
Projektkursus: Systemudvikling
Forår 2011

Arinbjørn Brandsson (hkt789)
Lasse Ahlbech Madsen (xsc606)
Naja Mottelson (vsj465)
Søren Pilgård (vpb984)

Gruppeid : LO6

Instruktor: Lasse Nørregaard

31. maj 2011

Indhold

1 Formål

1.1 Systemdefinition

Systemet består af et digitalt interaktivt læringsværktøj hvis mål er at indføre elever på gymnasiefaget Informationsteknologi i programmering af systemer i forskellig størrelse. Disse systemer vil primært være fokuseret på udvikling af spil, men bredt datalogisk kompetencegivende. Systemet opbygges som et hjælpeprogram, som kører ved siden af brugerens teksteditor og en række præetablerede programmeringsforløb som eleven kan deltage i. Et forløb består af en række trin - konkrete programmeringsopgaver som hver indeholder en beskrivelse, en række hints samt et sæt tests der holder styr på om brugeren har opfyldt målet for hvert trin. Når brugeren har gennemgået alle trinene vil vedkommende have et komplet spil der kan afvikles separat fra udviklingsværktøjet. Systemet vil blive udviklet som en skrivebordsapplikation der kan afvikles lokalt på den enkelte brugers datamat/foldedatamat eller på datamater i gymnasiets eventuelle it-rum.

1.2 BATOFF

- **Betingelser**: Gymnasieelever med begrænsede programmering- og systemudviklingserfaringer.
- **Anvendelsesområde**: Eleven som bruger af systemet. Hvis værktøjet installeres centralt på et gymnasiums datamater vil der typisk være en It-administrator hvis rolle er at opsætte systemet.
- **Teknologi**: Systemet designes til at kunne køre på almindelige hjemmedatamater/foldedatamater. Systemet vil blive baseret på python samt forskellige biblioteker dertil (i særdeleshed biblioteket pygame)
- **Objekter**: En bruger som udvikler et spil i forbindelse med et forløb af opgaver.
- **Funktion**: Udvikling af spil.
- **Filosofi**: Digitalt interaktivt læringsværktøj

2 Kravsspecifikation

Den kravsspecifikation vi har arbejdet med bygger på følgende krav:

1. Systemet skal kunne oprette nye forløb, samt tilgå gamle, ikke-gennemførte forløb lagret lokalt på brugerens datamat.
2. Systemet skal indeholde funktioner til at registrere om et forløb er gennemført af en bruger.
3. Systemet skal indeholde flere forskellige læringsforløb som brugeren kan gå i gang med.
4. Systemet skal ikke tilbyde en integreret teksteditor.

5. Systemet skal indeholde funktionalitet til at køre brugerens kode samt fremvise fejlbeskeder.
6. Systemet skal for hvert (relevant) trin i en programmeringsopgave tilbyde en tekst med hints og links til relevant dokumentation.
7. Brugergrænsefladen skal ligne (om end ikke være 100 procent ens med) figurerne (henvisning til figurerne)
8. Systemet skal være intuitivt at arbejde med - kun det programmeringsfaglige skal være krævende.
9. Systemet skal kunne køre på både Windows, Mac og Linux
10. Systemet skal ikke stille hardware-krav som ikke mødes af størstedelen af almindelige hjemme-PC'er.

Vi har i arbejdet med projektet lagt vægt på følgende designkriterier (opdelt efter vigtighed og hvorvidt det er opnået):

Kriterium	Meget vigtigt	Vigtigt	Mindre vigtigt	Irrelevant	Trivielt opfyldt
Brugbart		x			
Sikkert				x	
Effektivt			x		
Korrekt			x		
Pålideligt			x		
Vedligeholdbart			x		
Testbart				x	
Fleksibelt	x				
Forståeligt		x			
Genbrugeligt			x		
Flytbarhed		x			
Integrerbart	x				

Tabel 1: Kvalitetsattributter for klienten.

3 Den tekniske platform

Teknisk består vores program af 4 lag (startende nederst): python, pygame-biblioteket, spilskelettet og vores applikation sammen med brugerens editor.

Her vil python vil være den bærende del (da selve applikationen er kodet i dette sprog), og pygame-biblioteket bruger vi til at håndtere ting som 2d grafik, lyd og tastaturinput. Spilskelettet er den ufærdige kildekode til spillet som brugeren får udleveret og skal bygge videre på.
Øverst har vi et todelt lag bestående af vores applikation samt brugerens editor. Det vil være gennem dette lag at brugeren interagerer med de dybere lag.

4 Arkitektur

4.1 Komponentarkitektur

Vores komponentarkitektur er opdelt i tre lag: klienten (hvis ansvar det er at hente dataet fra de forskellige trin og vise dem), modelkomponentet (som opbevarer de forskellige trin og holder styr på hvor langt brugeren er i forløbet og således hvilke trin der kan blive vist og hvilke hints der kan blive givet) og så de tests der bliver kørt i forbindelse med hvert trin. To komponenter kan siges at agere sideløbende med vores program, navnligt brugerens eget tekstredigeringsværktøj og selve det spil som brugeren arbejder på, som lagres lokalt på brugerens datamat.

4.2 Procesarkitektur:

Vores system bliver udført på gymnasiets datamater eller elevernes egne folde-datamater. Potentielle problemer med samtidige processer forventes at blive løst af operativsystemet. Kontrollen over vores system ligger i modelkompnentet og klientkomponentet, hvilket er håndteret af operativsystemet (og i mindre grad, af den tekst editor som brugeren bruger).

4.3 Standarder:

Designet af de forskellige vinduer og fejlbeskeder vil følge ikke følge en bestemt standard, da vi vil integrere dem med vores system så at de passer med vores systems design.

5 Modelkomponent og Funktionskomponent

5.1 Modelkomponentet

Vores modelkomponenet holder styr på hvor langt brugeren er i forløbet og hvilke tests, der skal køres på dennes kode. Desuden holder det styr på, hvor mange hints, der er vist.

5.1.1 Klasser i modelkomponentet

Klassen "Bruger"er hvor det bestemmes hvilken opgaver man har og ikke har lavet, hvor klassen "Opgaver"bestemmer de forløb en bruger kan være i gang med eller have gennemført. Under klassen "Spil"valgte vi at generalisere de to stadier "Færdig"og "Ikke Færdig/Fejl", hvilket bestemmer om brugeren har lov til at fortsætte med den næste opgave, eller skal arbejde videre med den opgave som de er i gang med. Klassen "Test"refererer til de grupper af tests der er associeret med hvert enkelt trin, og som modellen kalder når klienten signalerer at brugeren er nået til et bestemt trin.

5.2 Funktionskomponent

Vi har som sådan ikke rigtig arbejdet med et separat funktionskomponent, eftersom vores modelkomponent indeholder de funktioner som kontrollerer programmet (i samarbejde med klienten). Man kan muligvis argumentere for at klienten

selv er en form for funktionskomponent, da det er denne del af programmet der signalerer til modelkomponentet hvor brugeren er i et forløb - dog er klienten også ansvarlig for opgaven at vise selve grænsefladen for brugeren, og kan ikke kaldes udelukkende et funktionskomponent.

6 Brugergrænsefladekomponent

Som det også forklares i afsnittet om projektarbejdet: Siden Delrapport 3 har vores indsats udelukkende været fokuseret omkring udvikling af de forskellige trin der udgør det første forløb (fem af disse er vedlagt som bilag til denne rapport). Vi har derfor på nuværende tidspunkt ingen kodet brugergrænseflade at fremvise.

Vores tanker vedrørende struktureringen af brugergrænsefladekomponentet er at det vigtigste er at grænsefladen er så enkel som muligt, for ikke at forvirre brugeren unødigt. Dette er heldigvis en overkommelig opgave da grænsefladen hovedsageligt vil skulle fremvise trin, som udelukkende består af nemt overskuelig tekst.

Vores plan er at dele grænsefladen op i 3 overordnede vinduer:

Det første vindue (kaldet start-up) vil stille brugeren overfor valget enten at starte et nyt forløb eller fortsætte på et der allerede er etableret. Hvis brugeren vælger at fortsætte på et tidligere startet forløb, vil en dialogboks komme frem hvori han kan navigere hen til forløbet og fortsætte på det. Hvis brugeren derimod vælger at starte et nyt forløb bliver han dirigeret til hovedvindue nummer 2. Vinduet består af 3 knapper for at holde tingene så simple som muligt:

- *Start nyt forløb* knappen
- *Fortsæt tidligere forløb* knappen
- *Hjælp* knappen der åbner en dialogboks hvor man kan læse om brugen af programmet.

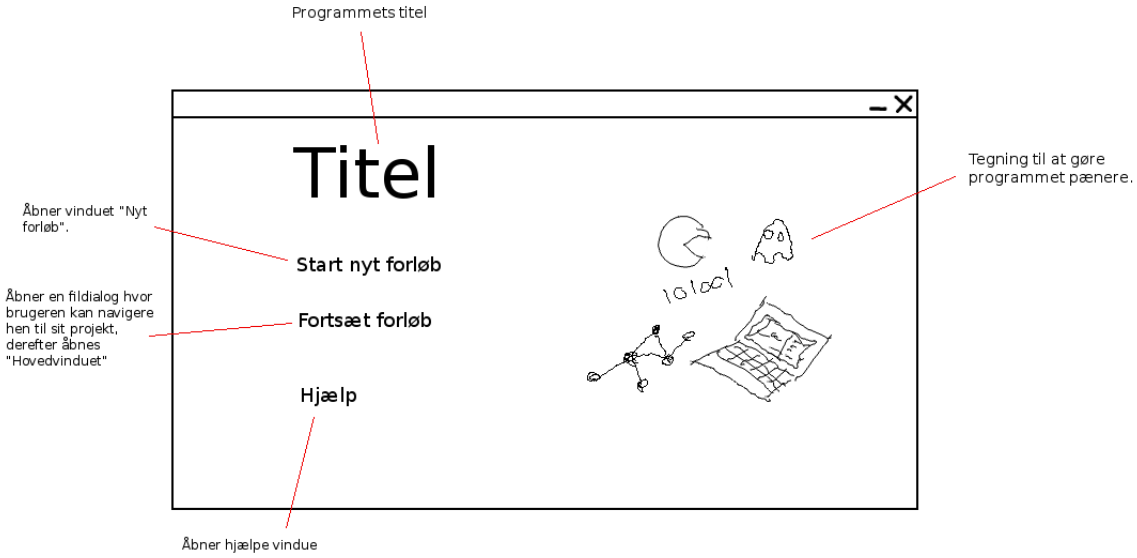
Det andet vindue (kaldet initialisation) vil være der hvor brugeren kan starte et nyt forløb og indtaste navn og sti i filsystemet (hvor forløbet skal gemmes). Det er også her hvor brugeren skal finde ud af hvilket forløb han vil gå i gang med: De forskellige forløb vil præsentere hvilket spil de laver, hvad man lærer af at lave dette spil samt hvor svært/hvilke krav der til brugeren før han kan gå i gang (f.eks. at brugeren har styr på de grundlæggende dele af python) Når brugeren er klar kan han trykke start.

Det tredje vindue er selve applikationen. Vinduet er tænkt at fylde i højden men ikke i bredden. Her vil man have en række knapper til at navigere med, samt et større område med tekst. Øverst vil der sidde 4 knapper:

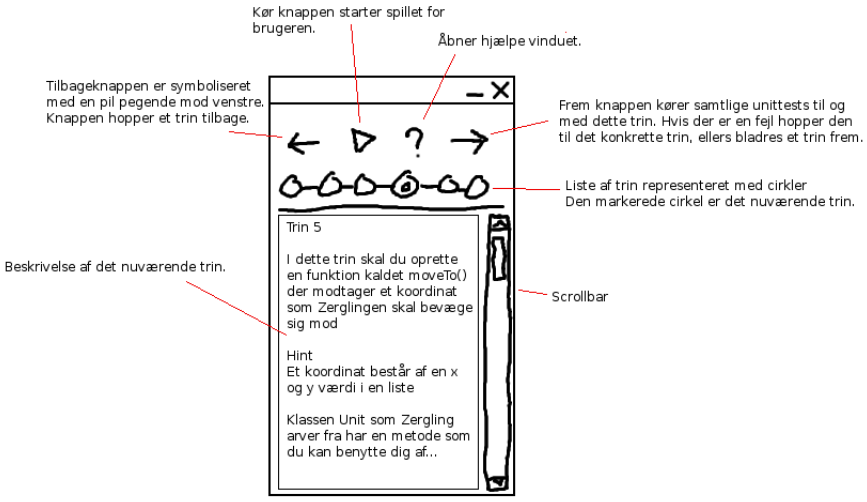
- En tilbage knap, symboliseret med en pil der peger mod venstre.
- En kør knap der starter spillet.

- En hjælp knap med et spørgsmålstegn der åbner hjælpesiden som i vindue 1.
- En frem knap symboliseret med en pil mod højre.

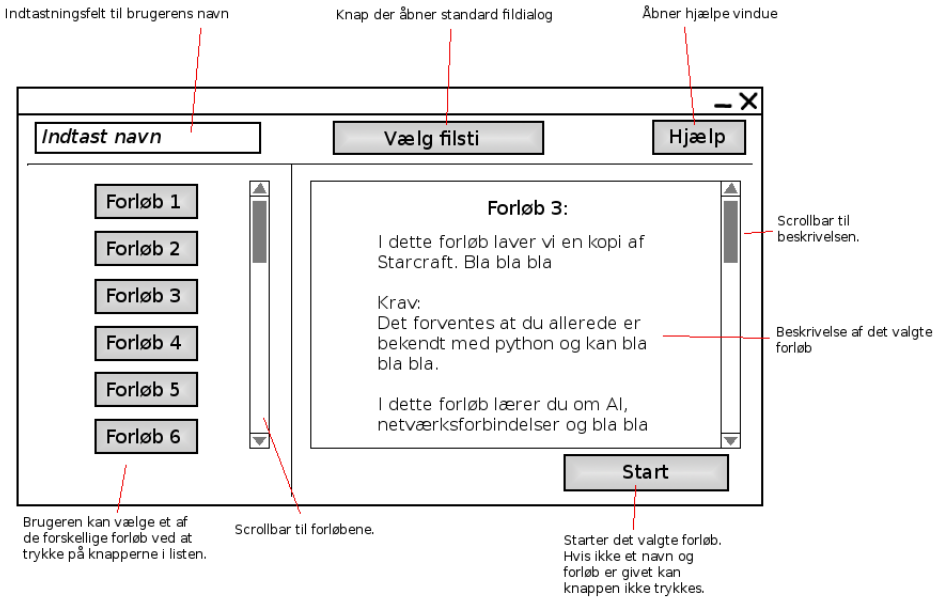
For at opnå en forståelse af hvordan vi har tænkt os at brugergrænsefladen kommer til at tage sig ud, har vi vedlagt figurene 1, 2 og 3 som viser vores tegnede mockups af grænsefladen.



Figur 1: Programmets startvindue



Figur 2: Programmets hovedvindue



Figur 3: Et nyt forløb

7 Implementering af IT-løsningen

Vores projekt delt op i 2 dele:

Først er der spillet som brugeren skal udvikle. For at kunne stille en ramme op for brugeren og for at kunne indele udviklingen i mindre opgaver, måtte vi lave en virkende udgave af spillet til at starte med.

Udover spillet er der selve programmet hvori brugeren bliver guidet igennem udviklingen, via små opgaver formuleret som trin i det forløb udviklingen udgør. Det er dette programs opgave at køre forskellige tests af brugerens kode, på den måde kan vi være sikre på at brugeren har løst opgaven korrekt så vi kan fortsætte videre i forløbet.

Status er lige nu at selve programmet (som kan ses som projektets egentlige produkt) desværre ikke er udviklet færdig. Hvad vi har fokuseret på at kode hidtil har været det spil der fungerer som det underliggende data for klienten, og som de enkelte trin tager udgangspunkt i.

Til at starte med havde vi kun en vag ide om hvordan forløbet om at skabe et spil bedst kunne struktureres, derfor var det naturligt at dele udviklingen op i 3 faser.

1. Udviklingen af et virkende spil.
2. Opdeling af kode opgaver i mindre trin som tilsammen udgør et forløb.
3. Udvikling af selve hovedprogrammet.

Denne rækkefølge var nødvendig da vi var nød til at vide hvordan spillet fungerede før vi kunne lave et forløb ud af det. Og vi var nød til at vide ret præcist hvordan et forløb med dets trin fungerede før vi kunne implementere det ordentligt i programmet.

Vi valgte derfor den ovenstående rækkefølge. Dette har dog også betydet at vi ikke kunne arbejde med projekterne parralelt og er derfor løbet ind i tidsproblemer. Vi er i gruppen kun nået til punkt 2 og har ikke fået udviklet selve programmet endnu.

Dette har i høj grad været en prioterings sag. Selve programmets funktionalitet er ret simpel men uden et forløb er det ubrugeligt. Det er derimod muligt at håndkøre et forløb hvilket gør at vi kan teste forløbene uden at have et virkende program.

7.1 Implementation af spillet

Vi valgte at udvikle spillet i python primært fordi det er et lettilgængeligt sprog, både for os, men i lige så høj grad for brugerne. Samtidig havde vi gode erfaringer med biblioteket pygame som er i stand til at håndtere mange grundlæggene ting som tegning af billeder, håndtering af input fra brugeren osv. Kombinationen af python og pygame gør det muligt at udvikle prototyper enormt hurtigt og gør det muligt at have en meget agil tilgang til udviklingen. Da vi på forhånd ikke havde så dybdegående ideer om hvordan man bedst kunne strukturere spillet så

det passede bedst ind i et forløb valgte vi at bruge en meget itterativ udviklings procces. Først blev der lavet en så simpel men virkende grundfunktionalitet som muligt. Herefter blev koden udbygget med mere funktionalitet samtidig med den blev refaktoreret. Ofte startede vi med at lave store funktioner der tog sig af al funktionaliteten, men som vi arbejdede videre blev disse funktioner splittet op i mindre bidder der gjorde koden mere overskuelig. Samtidig betød de mindre funktioner at vi fik nemmere ved at dele koden op i mindre opgaver. Dette er især tydeligt i *game.py* som startede med en funktion kaldet main som tog sig af alting. Funktionen blev dog løbende splittet op i mindre dele og hovedløkken blev skilt fra og delt op i mindre bider. Med hovedløkken delt op i timing, håndtering af input, opdatering af spillet og rendering, blev det pludseligt meget simplere og mere overskueligt. Så selv om vi kun startede med en ide om hvordan koden skulle ende lykkedes det os med en agil og itterativ tilgang at skabe en kodebase med små letforståelige funktioner som forhåbentligt gør spillet lettere at forstå for brugeren samt lige så vigtigt, gør det nemmere at lave tests af brugerens kode.

Vi valgte at gå ud fra Model-View-Control mønstret da vi lavede spillet. MVC er et ret udbredt mønster i spilverdenen og vi har derfor valgt at følge det.

Spillet selv er splittet op i en række forskellige moduler der tager sig af forskellige ting. Vi har valgt at være ret agressive med opdelingen, dels for at gøre det mere overskueligt for brugeren og dels for at vænne dem til at dele deres kode mere op. Opdelt kode er lettere at overskue, rette og teste - tre ting vi gør brug af i projektet.

Spillet består af følgende filer:

- *game.py* indeholder m̈ainm̈etoden, det er herfra spillet startes. Derudover indeholder den selve spilløkken.
- *model.py* indeholder alle de klasser der er i spillet, her er alle spilobjekterne defineret.
- *data.py* er en slags database over alt i spillet. Modulet holder styr på de forskellige spilkonstanter så som vinduestørrelsen, samt sæt af spillets objekter. Når et nyt objekt bliver oprettet skal det tilføje sig i de relevante grupper så de bliver opdateret og vist korrekt.
- *mapping.py* banerne til spillet består af simple tekstfiler der indlæses og gemmes i data modulet. Når banerne er blevet indlæst er der funktionalitet i mapping modulet der kan omsætte banen til faktiske objekter.
- *direction.py* Udgør enums for alle 4 verdenshjørner, modulet bliver brugt til bevægelse af spilleren.
- *functions.py* indeholder lidt blandede funktioner.
- *resources.py* indeholder alt funktionalitet der har med faktisk indlæsning af data at gøre.
- *ai.py* indeholder algoritmer til pathfinding.

Udover disse filer findes to mapper *images* som indeholder alt billedmaterialle, samt *maps* der indeholder banefiler.

Spillet startes ved at køre *game.py* i python. main metoden initialisere spillet og starter derefter hovedløkken. Hovedløkken er ret simpel, den sørger for at spillet kører med en jævn hastighed derefter håndteres input fra spilleren, alting i spillet opdateres hvorefter det tegnes på skærmen. Dette gøres i en uendelig løkke indtil spillet er ovre. *game.py* indeholder den funktionalitet der har med control delen at gøre.

I *model.py* findes selve spilklasserne til spillet og udgør dermed model delen. Hver klasse er barn af pygames Sprite klasse hvormed vi kan få pygame til at tage sig af mange af de praktiske ting, så som collision detection og rendering. Fælles for alle objekter er at de har en konstruktør hvor de henter deres sprite, indsætter sig i deres respektive grupper samt positionerer sig korrekt på banen. Både Player klassen og Monster klassen som er de to äktiveklasser indeholder derudover en update metode. Update metoden bliver kaldt hvert frame og sørger for at udfører alt logikken. Player klassens logik består i at rykke sig hvis brugeren ønsker det og der ikke er en væg i vejen. Tilsvarende vil Monster klassen prøve på at komme hen mod spilleren hvis denne er inde for rækkevide, dette gøres med at kalde pathfindingalgoritmen i *ai.py*, det skal dog siges at denne algoritme er ekstremt dårlig da den på nuværende tidspunkt blot returnerer en tilfældig retning. Det er ideen at vi bruger en bedre algoritme her såsom A* eller et genereret Leemap som der så kan slås op i.

I *data.py* lagres alt informationen om spillet, denne konstruktion sikrer at alle dele af spillet har let adgang til spillets data. Det vigtiste her er de såkaldte spritegroups som er en datastruktur pygame bruger for hurtigt at kunne arbejde med sprites. Spritegroups har indbygget funktionalitet så man f.eks. kan kalde alle medlemmers update metode med et enkelt kald. Derudover bruges spritegroups også til collisiondetection samt hurtig rendering af flere sprites. Sprite og Spritegroupsne er implementeret i c og brugt rigtigt kan de være med til at få spil lavet i python til at køre forholdsvist hurtigt.

direction.py indeholder en slags enum over de fire verdenshjørner samt retningen NONE, derudover er der funktioner til f.eks. at lave en retning om til en vektor.

functions.py, *resources.py* og *mapping.py* er hjælpe moduler med ekstra funktionalitet. *resources.py* tager sig af alt hvad der har med indlæsning af filer at gøre. *mapping.py* tager sig af at lave en indlæst tekstfil om til spilobjekter. *functions.py* indeholder pythagoras funktionen samt funktionen quit der afslutter spillet og udskriver spillerens score til terminalen.

Hele spillet er på nuværende tidspunkt kørende og afprøvet, der mangler kun at blive implementeret en ordentlig pathfinding algoritme. Dette har dog ikke været en hindring for at gå igang med at indele koden i forskellige opgaver til vores forløb. Udover koden er graffiken til spillet også færdig, grafikken består af .png filer i images mappen. maps/ mappen indeholder eksempler på hvordan

en bane lagres. Tanken er at brugeren kan manipulere disse filer som han har lyst.

Kildekoden der er vedhæftet denne rapport er fra spillets model.py-modul.

7.2 Hovedprogrammet

Selve programmet har vi ikke nået at udvikle, det er ærgeligt men ikke katastrofalt. Det er muligt at afprøve forløbene ved at udlevere koden og opgavetekssterene til hvert trin et ad gangen, man kan så manuelt se om det brugeren har lavet er korekt. Det er altså muligt at teste vores program uden automatiserede unittests og visning af opgave beskrivelser.

For at forstå hvordan programmet virker er det nødvendigt at have en ide om hvordan forløbet skal organiseres. Forløbet består af en række trin som hver er en opgave som brugeren skal løse. Når brugeren mener han har løst opgaven trykker han på videreknappen, herved køres en række unittests af hans kode, hvis testsne gennemføres ved vi at brugeren har løst opgaven korrekt og han føres videre til næste trin, hvis en test fejler visses en besked med hvad der gik galt og hvad der evt. kan gøres for at rette fejlen.

Hvert trin skal derfor have en beskrivelse af opgaven, for at gøre værktøjet så fleksibelt som muligt er tanken at dette laves i et markup sprog ala html der kan håndtere ting som hyperlinks, grundlæggene typografi, billeder, tabeller, lister osv. Testsne er et python script der køres, hver test er så en funktion der enten lykkedes med succes, eller returnerer en fejl meddelelse som vises i et popup vindue eller lignende. Ideen er at værktøjet er så fleksibelt at man kan proppe mange forskellige typer forløb ind i det. Det kunne f.eks. være at man istedet for at skulle lave et spil skulle lave et andet program, værktøjets fleksibilitet kan faktisk udnyttes til alt der kan sættes på formen bestående af et forløb af opgaver med efterfølgene tests.

Programmet vil være let at udvikle. Ved at bruge et gui bibliotek med funktionalitet til at tegne vinduer med html indhold, selve testsne er nemme at håndtere da man blot skal indlæse dem og lade pythons fortolker afvikle dem. Meget mere funktionalitet er ikke nødvendig.

8 Projektsamarbejdet

8.1 Projektstyring

Siden sidste rapport har langt størstedelen af vores arbejde været rettet imod fremstilling af trinene til projektet og klargøring af test af disse. Fem af trinene er vedlagt nærværende rapport (se bilag). Processen omkring begge disse opgaver har været langt fra optimal: Selve udfærdigelsen af trinene trak ud længere end det burde have gjort, eftersom samtlige gruppemedlemmer har været under forskellige former for tidspres, og generelt været dårlige til at delegere tilstrækkelig tid til projektet. Selve afprøvningen er blevet presset af at vores kundes elever gik på læseferie inden vi kunne igangsætte afprøvningen.

Vi udviklede trinene ved først at uddelegere 2-3 trin til hvert enkelt gruppemedlem, og derefter satte vi os sammen og strømlinede dem, så de blev ens i format og tilgang. Dette viste sig at være en rigtig god måde at tvinge alle til at være kreative, og komme med deres bud på en løsning. På nuværende tidspunkt

er 10 ud af 13 trin i det første forløb blevet færdigproducerede. Dette er ikke optimalt, men tilstrækkeligt til at kunne udføre sigende tests.

8.2 Møder med brugeren

Siden delrapport 3 har vi afholdt ét yderligere møde med kunden, i form af den afprøvningssession der beskrives i afsnit 8. Alt i alt vil dette sige at vi har afholdt 3 møder med vores kunde (udover vores korrespondance pr. mail og telefon) - et indledende møde hvor vi mødte eleverne og blev enige med deres underviser om projektets udformning, et møde hvor vi afprøvede papir-mockups og afslutningsvis det møde der beskrives nedenfor.

8.3 Referat- og dokumentationsstrategi

Alt i alt har vores referater under projektets forløb været forholdsvis kaotiske. I projektets begyndelse har vi ført meget grundige (og hyppige) referater, som på nuværende tidspunkt ikke fremstår så nyttige eftersom en stor del af selve projektet har ændret sig drastisk siden da.

I Delrapport 3 fremsatte vi hensigten at uddele selve projektopgaven i 4 hovedområder (projektstyring, kode, afprøvning, dokumentation) og uddelegerede ansvar for hvert område til et enkelt gruppemedlem. Siden da har det vist sig at denne idé måske var for ambitiøs: Vi har slet ikke haft tid eller overskud til at arbejde på alle områder, og sidste ende ikke været nogen grund til at holde hinanden i tovene på de forskellige områder. I stedet har vi kørt med den samme model som hidtil, hvor vi har en person med det overordnede ansvar. Det er stadig ikke optimalt, og vi har stadig problemet at det ikke er alle gruppemedlemmer som har fuldt overblik over projektet. Det skaber meget mere arbejde for dem der rent faktisk har det, og sætter de resterende medlemmer værre i en eksamenssituation, når de skal redegøre for forløbet og de forskellige projektdele.

I de møder vi har haft, har vi nok engang haft sløsede referater, men de har heller ikke været så relevant, da alt vores arbejde og tanker røg ned på papir i forbindelse med de trin vi har udviklet. Vi har siden sidste rapport ikke oplevet problemer med opmøde, ud over at folk stadig kommer for sent. Ideen med at lægge møderne i forlængelse med undervisningen var som sådan ikke en dårlig ide, men det er som regel mest praktisk at lægge møderne tidligere på dagen, og folk har det tilsyneladende bedre med at komme for sent til disse end undervisning.

Vores valg af versionsstyringsværktøj fungerer til gengæld glimrende. Vi har nu alt vores arbejde delt og sorteret i mapper og alt er let tilgængeligt. Folk har en meget god forståelse for hvordan GitHub fungerer, og der er på nuværende tidspunkt ikke nogen der oplever problemer med at bruge det. Den eneste problematik vi oplever på den front er at nogle gruppemedlemmer ikke altid pusher så snart de er færdige med en opgave, hvilket gør det besværligt at danne sig et overblik over hvor langt vi er med enkelte delopgaver uden at skrive ud på mailinglisten eller at pinge folk provat. Dette er meget frustrerende, i sær eftersom det er et problem der er blevet påpeget og kritiseret gentagne gange i løbet af projektet.

8.4 Perspektivering

8.4.1 Projektstyring generelt

En overordnet ting vi alle sammen har oplevet i dette projekt er at vores valg af generel ansvarsfordeling ikke har fungeret. Modellen hvor et enkelt menneske har ansvar for selve projektstyringen fungerer sandsynligvis udmærket i professionel sammenhæng og på større projekter, men i vores størrelsesorden og niveau har det simpelthen ikke fungeret. Når man arbejder sammen så få mennesker, og i så løbende kontakt som man gør som studerende, gør det næsten mere skade end gavn at have et enkelt menneske med mere ansvar end resten af gruppen. Da vi traf beslutningen om at organisere os på denne måde var vi i en lidt anden situation, med et yderligere medlem i gruppen. Havde vedkommende ikke faldet fra projektet kunne vi muligvis have haft nytte af denne måde at organisere os på, men det kan jo ikke blive andet end spekulation på nuværende tidspunkt.

Vi har gennem forløbet arbejdet en del med at forbedre vores dokumentationsstrategi, til tider fungerede den godt, med aktions-reaktions skemaer for vores interviews, men til vores møder formåede vi aldrig at få skabt en god standard. De fleste af vores referater kan ikke bruges til meget mere end at tjekke egentlige beslutninger. I et fremtidigt projekt ville vi absolut udbedre dette, og bl.a. få argumenter og begrundelser med for de valg vi traf, så vi i mod- sætning til nu, kan se hvorfor vi valgte at gøre som gjorde.

Vi valgte i processen at bruge en betragtelig mængde energi på det pædagogiske indhold af vores projekt, og på at finde ud af hvad selve spillet skulle gå ud på o. lign. På denne måde er vores implementationsarbejdet blevet meget forhalet, resulterende i at vi ikke kommer til at have en kørende prototype på det planlagte tidspunkt. I retrospekt ville en god idé have været at påbegynde implementeringen sideløbende (eller, potentielt, inden) analysearbejdet, så at vi ville have en base af fungerende programmel at udvide på i takt med at projektets indhold blev mere veldefineret.

I et fremtidigt projekt vil vi endvidere forsøge at være hårdere med deadlines da dette har været et gennemgående problem i gruppen, og har skabt stress og dårligere produkter. Det er dog svært at rette på, når det er et generelt problem i gruppen, og vi ikke har adgang til andre konsekvenser end at forlade gruppen. Det bliver uundgåeligt noget med at forsøge at holde hinanden i ørerne, og forsøge at holde konstant opsyn med status på forskellige opgaver, hvilket er generende for alle i gruppen.

Alt i alt er det der generer os mest, at vi har set os nød til at fokusere mere på at skrive rapporter, end at udvikle et produkt vi kunne være tilfreds med. Her et par uger før eksamen står vi ikke med noget der ligner et færdigt program, og det er ikke, fordi det er et voldsomt stort eller kompliceret program, men fordi rapporterne syntes at fylde mere end udviklingen.

8.4.2 Valg af kunde og kundekontakt

Vi kunne godt have brugt mere kontakt med kunden, eftersom vi syntes at det blev meget "vores"projekt i stedet for et vi lavede i fællesskab. Specielt i starten virkede det heller ikke som om, at vi havde den samme forståelse af meningen med projektet, og det ønskede produkt. Dette kunne bestemt gøres bedre, og ville forventeligt være anderledes i en situation, hvor vi var hyret til en opgave, i stedet for at vi er ude og sælge et projekt, som vi godt kunne tænke

os at lave. Hvis der var en kontrakt involveret, ville det nok også have været anderledes, da kunden ville være mere opsøgende i forhold til status og den videre udvikling af projektet, i stedet for passivt at vente på, hvad vi nu finder på at aflevere. Det var en risiko vi tog, da vi valgte at arbejde sammen med en enkelt gymnasielærer i de stressede månederne op til gymnasieeksamenerne. Dette skulle vi måske have været mere bevidste om.

Ved indgangen til projektet stod vi overfor to alternativer til den kunde vi endte med at vælge at samarbejde med. En anden samarbejdspartner vi kunne have valgt at arbejde med var foreningen DKUUG, som var meget interesseret i et projekt af lignende tilskæring, om end med andet fagligt fokus end spiludvikling. Endvidere havde vi, udover tilbuddet fra Greve Gymnasium, muligheden for at samarbejde med et yderligere gymnasium og på denne måde fordoble mængden af gymnasieelever at bruge som testpersoner.

Vi har generelt haft besvær med at komme ordentligt i kontakt med vores kunde på Greve Gymnasium. Dette kan der have været mange forskellige grunde til - bl.a. kom vi jo ind i deres undervisningscyklus på et meget sent tidspunkt, hvilket måske bidrog til at dæmpe både underviserens og elevernes interesse i at deltage i projektet. I sær her til sidst har vi haft meget svært ved at få fat på gymnasiet, af den simple grund at studentereksaminerne er gået i gang og både lærerens og elevernes prioriteter nu (forståeligt nok) ligger et helt andet sted. Vi forestiller os at vi ville have haft præcis de samme problemer hvis vi havde valgt et andet gymnasium at samarbejde med, samt hvis vi havde valgt at arbejde med to sideløbende. Faktisk ville samarbejde med to gymnasier potentielt have været ret skadeligt for projektet, da det ville fordoble den (i forvejen betragtelige) opgave at holde kontakt til kunden, informere om fremskridt og deltage i løbende møder.

I retrospekt ville et samarbejde (enten eksklusivt eller som én kunde sideløbende med et gymnasium) med DKUUG måske have været en god idé. Et problem vi oplevede løbende med Greve Gymnasium var at gymnasielæreren vi havde størstedelen af vores kontakt med lod til at opfatte det som 'vores' projekt i højere grad end hans som kunde. Han havde derfor ofte ikke så meget at bidrage med, hvilket gjorde at vi var nødt til at bestemme størstedelen af kravene til systemet samt det pædagogiske indhold selv. Dette har vi ikke været tilfredse med, fordi det i høj grad har tilføjet til vores arbejdsbyrde i projektet. Folk fra DKUUG ville potentielt havde kunnet fratage os en del af dette ansvar, af den simple grund at de ville have bedre idé om præcis hvilken slags program de kunne tænke sig at ende op med.

8.4.3 Analysearbejdet

En mere overordnet (og egentlig mere problematisk) observation vi har gjort os er at det projekt vi har valgt ikke passer særlig godt på den analysemetode vi benytter på kurset. Selvom spillet selv er objektorienteret er den applikation som er det egentlige program ret svært at beskrive med de værktøjer og modeller der stilles til rådighed i bogen. Dette har betydet at en del af de opgaver der har været i forbindelse med forfatning af delrapporter har fungeret som yderligere arbejdsopgaver snarere end at hjælpe os med at udvikle og forstå programmet vi har arbejdet på. Når vi næste gang påbegynder et projekt vil vi derfor gøres os umage med at vælge projekt og analysemetode der rent faktisk understøtter hinanden på en produktiv måde.

9 Afprøvning med brugerne

Som nævnt i delrapport tre har vores afprøvning med brugerne (i dette tilfælde gymnasieelever) været en del anderledes end afprøvning beskrevet i kursuslitteraturen af den grund at vi ikke har en kørende prototype af selve klienten til vores program. Hvad vi valgte at gøre var derfor i stedet at fokusere på at afprøve det pædagogiske indhold i de trin vi har udfærdiget til programmets første udviklingsforløb. Dette involverede dog også en del forberedelse, i særdeleshed i forbindelse med at udvælge det materiale der skulle afprøves.

Som nævnt i forgående afsnit havde vi indledningsvist meget svært ved at få kontakt til de testpersoner vi gerne ville benytte i Greves datalogiklasse. Vi endte dog med at aftale en afprøvningssession på to timer med tre elever d. 31. maj. Heraf dukkede to af dem op. Dette viste sig dog at fungere meget godt, da vi hurtigt løb tør for tid og det potentielt ville have krævet flere ressourcer at have flere testpersoner.

9.1 Overvejelser og forberedelse inden afprøvningen

En stor del af de trin vi har udarbejdet indeholder python- eller pygamespecifik information (se vedlagte bilag), som vi gerne ville undgå at forudsætte for meget af. Vi valgte derfor at fokusere på de helt grundlæggende funktionaliteter i spiludviklingen, som nemmest kunne abstraheres og gøres forståelige. Et kriterium for de ting vi stillede spørgsmål til var at de såvidt muligt var nemme at forestille sig uden at tænke for meget på specifikke implementeringsdetaljer (fx. ville vi gerne undgå ting som input handling, som kræver et vist kendskab til håndtering af tastaturinput, og ting som fokuserede for meget på matematiske udregninger, da mange har svært ved at udføre den slags på stående fod). Vi valgte derfor at fokusere på at stille spørgsmål til opbygningen af de forskellige entitetsklasser i spillet (spilleren selv, vægge, monstre, guld), udvalgte dele af klassernes opførsel i spillet (hovedsageligt collision detection), samt den grundlæggende funktionalitet i main-funktionen.

Ca. en uge inden mødet havde vi sendt dem en mail med en kort introduktion til hvordan afprøvningen ville foregå, samt links til information som de med fordel kunne sætte sig ind i inden afprøvningen (grundlæggende objektorientering, pygame). Med i denne mail var desuden en kort beskrivelse af det spil vi ville stille dem spørgsmål til.

9.2 Afprøvningen

Afprøvningen foregik ved at vi indledningsvist gentog de grundlæggende rammer for det spil vi ville arbejde med (hvad en spiller skulle kunne, hvordan reglerne ifbm. monstre og guld var). Herefter foregik afprøvningen af de udvalgte trin ved at vi læste højt/forklarede den beskrivelse der indleder hvert trin, og stillede de spørgsmål som trinnet indeholdt. I tilfælde hvor trinnene fokuserede på teknisk specifikke ting (fx. pygames klasser eller datastrukturer) valgte vi at omformulere spørgsmålene så at de blev mere generelle. I tilfælde hvor opgaverne i trinnene afhang af kendskab til data eller funktionalitet i andre dele af programmet valgte vi enten at skitsere kort hvad der var i bemeldte del af programmet, eller omformulere opgaven så at den blev uafhængig af den programdel. Når testpersonerne havde svært ved at forstå opgaven eller ikke kunne

komme på en måde at løse opgaven gav vi dem de hints der ligeledes indgår i hvert trin.

Selve afprøvningen udspillede sig ikke 100 procent optimalt. Ved begyndelsen sagde begge testpersoner at de ikke havde haft tid til at sætte sig ind i python (dette havde vi heller ikke som sådan forudsat), pygame eller havde nogen erfaring med spiludvikling overhovedet. Dette viste sig at blive problematisk, fordi vi gentagne gange blev nødt til at forklare dem hvordan overordnede ting i spiludviklingen fungerer, såsom nedarvning og konstruktion af klasser og tilhørende metoder. Disse forklaringer var i høj grad af information som var at finde i de links der blev henvist til i trinnene, men det optog en del tid og vi havde indtryk af at testpersonerne havde svært ved at håndtere så store mængder information på én gang.

Spørgsmålene til de allerførste trin forløb forholdsvis kaotisk. Selve mainloopet blev vi nødt til at forklare i ganske høj detaljeringsgrad for at vores testpersoner kunne forstå det, og selv herefter blev vi hurtigt nødt til at fokusere på modellen i spillet fordi selve main-funktionen var svær for dem at forestille sig. Spørgsmålene til de enkelte klasser forløb en del bedre: Begge elever var i stand til at komme med bud på hvilke metoder de forskellige klasser skulle indeholde, og efter noget tid (og yderligere forklaring) kunne den ene af testpersonerne give et bud på hvordan Player-klassen kunne initialiseres, samt hvordan man kunne lave funktionalitet til at få spilleren til at samle guld op og blive angrebet af et monster. Hans bud på hvordan man kunne gøre det var anderledes end den måde vi selv har implementeret det, men i testens kontekst var det så meget som vi håbede på.

Som afprøvningen skred frem blev vi ret hurtigt nødt til at stille spørgsmål til opgaver der lå udenfor de trin vi havde planlagt at spørge til. Dette var af den simple grund at vi forholdsvis hurtigt løb tør for spørgsmål som eleverne kunne løse på stående fod, og var nødt til at prøve at finde på andre ting at spørge om. I særdeleshed havde begge elever svært ved at forstå spørgsmålene til collision detection og hvordan/hvor de klasser de havde beskrevet skulle initialiseres og sættes i brug. Vi forsøgte os med at stille spørgsmål til de udvidede metoder til de forskellige klasser (move og update) men selv efter gentagne forklaringer og hints kunne ingen af eleverne give bud på hvordan nogen af metoderne kunne implementeres.

Begge elever sagde at en stor del af deres besvær med at løse de opgaver vi stillede dem havde at gøre med at de ikke følte sig sikre nok til at kunne komme med bud på algoritmer uden at kunne sidde og lege med det på egen hånd. De mente begge to at de ville kunne løse (de nemmere af) opgaverne alene, hvis de havde den fornødne tid til at læse de links der indgik i trinene og kunne sætte sig mere ind i python, pygame og objektorientering generelt.

9.3 Overvejelser efter afprøvningen

Alt i alt synes vi at vores afprøvningssession med de to elever blev ret kaotisk. Af de specifikke ting de to elever havde svært ved er det svært at skelne imellem hvad der har at gøre med det faglige niveau i opgaverne og hvad der har at gøre med vores testpersoners manglende forberedelse og forvirring over den uvante måde at skulle løse datalogiske problemer. Det lader dog til at de opgaver der har at gøre med de mere matematiske/abstrakte dele af programmet (fx. collision detection og move-metoden) skal forklares meget mere grundigt, og måske deles

op på flere forskellige trin for at uvante programmører kan følge med. Det samme kan siges om de trin der har at gøre med selve main-loopet. En måde at løse problematikken ved main-funktionen og main-loopet ville være at give brugeren mere funktionalitet i skelettet der udleveres, så at de kan se hvordan det bliver gjort i stedet for at skulle lave så meget af det fra bunden.

Begge elever virkede dog positivt stemte overfor idéen om programmet som helhed, og syntes at (de mindre krævende af) opgaverne var sjove. Begge af dem var af den mening at de, hvis de havde bedre tid og måske fik mere udførlige beskrivelser i de svære trin, ville kunne gennemføre forløbet i sidste ende.

10 Diskussion af artikel

Artiklen "Designing the Handheld Maritime Communicator"handler om processen som omfattede at lave et system som ville være ansvarlig for at sørge for sikkerheden når risikabelt arbejde skulle udføres i store fragtskibe. Artiklen går i dybde med dette, hvor den beskriver alle trinene bag systemet, fra udformningen af systemet, til interviews med klientellet, indsamlingen af det information som skulle bruges, designet af systemet, og til sidst den måde som designerne af systemet besluttede at teste systemet, (både brugertests og bugtesting). Fremgangsmåden som programmørerne brugte til at designe systemet vil blive analyseret samt sammenlignet med kursuslitteraturen.

I stedet for interviews med deres klientel tog nogle medlemmer af holdet med på nogle af skibene for længere perioder, (Det siges ikke hvor længe), hvor de tog noter om hvordan opgaver var klaret, (som for eksempel hvis sømændene skulle få deres last om bord på skibet). Ud over dette tog de også noter om problemer med den måde som sømændene klarede disse problemer på, som så skulle løses. Holdet tog også lyd-og-video analyser, hvor de tog lyd segmenter eller video segmenter fra om bord på et skib, og kiggede dem igennem. Dette var gjort så at holdet kunne få en bedre idé om forholdene på skibene, samt også for at se hvilken ordrer der blev typisk givet. Deres plan var at bruge disse til at kunne lave forprogrammerede kommandoer. Denne måde som holdet brugte til at få information vedrørende det det system som deres kunde havde brug for er meget anderledes end den metode som Back to Thinking Mode havde argumenteret for, hvor man skulle have adskillige interviews med sin kunde, hvor man skulle føre en dagbog. Her brugte man hovedsageligt et 'action-reaction' type format. Dog kan man argumentere at denne fremgangsmåde ville ikke virke for dette projekt, da holdet havde brug for at have en klar idé om hvad arbejderne havde brug for, og ikke hvad deres chefer havde brug for, samt hvis de havde foretaget interviews i stedet for feltarbejde ville holdet ikke få at vide hvilken kommandoer var typisk givet fra kaptajnen eller lignende højrestående arbejdere.

Mens systemet stadig var i koncept-fasen lavede holdet et klassediagram og et papir mockup om systemet for at vise hvordan systemet skulle virke. Klassediagrammer er et udbredt fænomen i større projekter, som er beskrevet i bogen "Objekt Orienteret Analyse og Design", som er brugt til at give et samlet overblik over problemområdet, hvilket er nyttigt for dette system da systemet skulle kunne holde styr på skibet, opgaverne og det team som skulle udføre disse opgaver. Klassediagrammer er effektive da de hjælper med at holde et overblik over systemet, og også til at hjælpe folk med at se hvilken klasser har hvilken effekt på hvilken klasser. Papir mockups, som også er diskuteret i "Cardboard

Computers: Mockingit-up or Hands-on the Future", hjælper med at få folk til at forstå hvilken grænseflader holdet syntes vil være den bedste IT-løsning. Dette er en effektiv måde at vise et holds kunder hvilken grænseflader de syntes vil være effektive, da de er hurtigere at lave i forhold til at lave prototyper, hvilket betyder at et hold kan eksperimentere med forskellige design. Og da dette systems design er altafgørende, (da systemet skulle være nemt og hurtigt at bruge for arbejderne), var det en umådelig god idé at lave forskellige papir mockups i stedet for at lave mange forskellige prototyper.

Holdet som designede systemet brugte et såkaldt 'iterativ design'. Iterativ design fungerer ved at holdet laver en prototype. Denne prototype bliver så testet, analyseret og refineret, hvorefter holdet kigger på hvad der skal laves om eller skal ændres. Disse ændringer bliver inkluderet i en ny prototype, som gennemgår de samme trin som før indtil man har et færdigt produkt. Denne form for design er meget anderledes end et traditionelt vandfaldsmodel, (hvor man antager at hvert trin bliver udført perfekt, og skal ikke ændres), eller et spiralmodel, (fra "A Spiral Model of Software Development and Enhancement"), hvor man lavede en prototype for hver gang man havde startet et nyt trin. Selvom et iterativ design er mere tidskrævende en et vandfaldsmodel eller et spiralmodel, vil et iterativ design sørge for at når systemet bliver udleveret vil der være færre bugs, problemer og ineffektive designelementer. Dette er meget vigtigt for et sikkerhedssystem, da det kan koste liv hvis der sker en fejl. Artiklen "Designing for Usability: Key Principles and What The Designers Think"er enig med at et iterativt design er den mest effektive designproces, da et iterativt designproces vil eventuelt finde mange, hvis ikke alle, designproblemer, som for eksempel tvetydige beskeder. Artiklen give endda et eksempel på et problem som et hold havde, hvor brugerne misforstod nogle af de kommandoer som de kunne bruge, som iterativt design rettede op på. Tvetydige kommandoer eller funktioner ville have været et stort problem for projektet for brugerne, så det var en god idé at bruge et iterativt designproces til systemet.

11 Bilag

12 Trin

12.1 Trin 2: Oprettelse af Player-klassen

12.1.1 Skelet:

```
Class player(pygame.sprite.Sprite):
    def __init__(self, x, y):
```

12.1.2 Beskrivelse:

Du skal nu oprette en af de mest grundlæggende ting i spillet: selve spilleren. Vi gør dette ved at oprette en klasse (Player) som nedarver fra en klasse der er indbygget i pygame (Sprite). Det er hvad du kan se i det indbyggede skelet. Eftersom din Player arver alle de metoder som Sprite-klassen har, bliver en god del af din opgave at finde ud af hvilke metoder og attributter du vil bruge. Som det første skal du sørge for at din Player får indhentet et ikon, så den kan blive

tegnet på en bane (yderligere funktionalitet følger senere).

12.1.3 Hints:

1. Kig i resources.py. Er der funktionalitet der gør noget af det du har brug for der? Hvad returnerer de funktioner der er defineret i det modul?
2. Overvej hvilke attributter en spiller egentlig har brug for, udover et billede. Hvor er han henne? Hvor er han på vej hen?
3. Overvej om du kan kombinere indhentningen af billedet med indhentning af anden information om spilleren.
4. Kig grundigt på dokumentationen til pygames Sprite-klasse. Er der noget du kan bruge?

12.1.4 Links:

1. Pygames dokumentation til Sprite-klassen: http://www.pygame.org/docs/ref/sprite.html
2. Introduktion til Sprite-klassen: http://www.pygame.org/docs/tut/SpriteIntro.html
3. Generel spil-tutorial som (bl.a.) benytter Sprite–klassen: http://pygame.org/docs/tut/chimp/ChimpLin

12.2 Trin 7: Oprettelse af Wall-klassen

12.2.1 Skelet:

```
class Wall(pygame.sprite.Sprite):
    def __init__(self, x, y):
```

12.2.2 Beskrivelse:

Nu hvor du har en bane og en spiller der kan rykke rundt på den er det på tide at lave nogle mure som kan gøre banen lidt mere spændende. Som enhver trænet koder tænker du øjeblikkeligt på at oprette en klasse til at modellere vægge.
En væg deler visse karakteristika med en spiller: De kan begge nedarve fra pygames Sprite-klasse, og alle væggens grundlæggende metoder vil også kunne findes hos spilleren.
Når du har oprettet væg-klassen skal du sørge for at væggene bliver lagret det rigtige sted i spillet.

12.2.3 Hints:

1. Prøv at overvej hvad en mur egentlig består af, samt hvad den skal kunne. Skal den kunne flytte sig? Skal den have et billede? Hvordan spiller disse to ting sammen?
2. Hvor ligger informationerne om de ting der er at finde på en bane? Hvordan sørger du for at væggen bliver placeret dér når dens init-metode bliver kaldt?

12.3 Trin 8: Collision detection

12.3.1 Skelet:

```
Intet skelet udleveres til dette trin.
```

12.3.2 Beskrivelse:

Du skal nu tilføje 'collision detection'. Dette skal bruges til at sørge for at din spiller f.eks. ikke kan stå oven i mure. Dette skal implementeres som en del af move-metoden i player klassen, som du oprettede tidligere. Collision detection kan man med fordel kode med udgangspunkt i de biblioteker du får fra pygame.

1. Et godt udgangspunkt er metoden spriteCollide fra den gammelkendte Sprite-klasse.
2. Når du specifikt skal undgå at din spiller går ind i mure, hvilke argumenter skal din metode så bruge?
3. Hvordan sørger du for at der ikke er mere én sprite på en bestemt position?

12.3.3 Links:

1. Wikipedias artikel om collision detection: http://en.wikipedia.org/wiki/Collision_detection
2. Tutorial om collision detection i pygame: http://www.switchonthecode.com/tutorials/collision-detection-with-pygame
3. Grundlæggende pygame-tutorial indeholdende helt basal introduktion til collision detection og rects: http://pygame.org/docs/tut/newbieguide.html
4. Endnu en tutorial (også indeholdende sprite-grupper): http://www.devshed.com/c/a/Python/PyGame-for-Game-Development-Sprite-Groups-and-Collision-Detection/

12.4 Trin 6: Input handling

12.4.1 Skelet:

```
def handle_input():
    for event in pygame.event.get():
            if event.key == pygame.K_UP:
                ''Få spilleren til at skifte retning''
            elif event.key == pygame.K_DOWN:
                ''Få spilleren til at skifte til en anden retning''
```

12.4.2 Beskrivelse:

Du skal nu kode den del af spillet der håndterer input fra spilleren - dvs. at du skal få spillet til at reagere på det data det modtager fra tastaturet (fx. når spilleren trykker på piletasterne). I handle_input() skal du konkret finde ud af at stoppe spillet, samt at få spilleren til at bevæge sig vha. input fra tastaturet. // Måden vi gør dette i pygame er vha. såkaldt hændelseshåndtering (event handling). Dette står beskrevet i nærmere detaljer under links-sektionen. // Som du kan se af kodeskelettet lægger vi ud med at indhente en liste (også kaldet en

kø) af hændelser fra tastaturet (ln 1). Denne undersøger vi herefter, og angiver hvordan spillet skal reagere på beskeder fra de forskellige taster (eksempel: ln 2-5).

12.4.3 Hints:

1. Størstedelen af det data der skal manipuleres her er spillerens retning. Hvor lagres det?
2. Hvilket tastaturinput virker bedst til at bestemme retning på spilleren? Hvornår vil du indhente dataet fra tasterne - når de er blevet trykket ned eller når de er blevet sluppet igen?
3. Du skal (ud fra en hændelse fra tastaturet) få spillet til at stoppe. Hvor har du en funktion der stopper spillet?
4. Du kan hente inputtet til at stoppe spillet på flere måder - bl.a. har pygame en indbygget quit-konstant. Hvordan kan du ellers stoppe spillet? Er der behov for flere måder at stoppe spillet på?
5. Undersøg forskellen på de to hændelsestyper pygame.KEYUP og pygame.KEYDOWN. Skal spillet reagere forskelligt på de to?

12.4.4 Links:

1. Pygames hændelseshåndteringsmodul: http://www.pygame.org/docs/ref/event.html
2. Pygames liste af tastatur-konstanter: http://www.pygame.org/docs/ref/key.html
3. Kort opsummering af begerebet hændelseshåndtering: http://en.wikipedia.org/wiki/Event_handler
4. Mini-tutorial om hændelseshåndtering: http://lorenzod8n.wordpress.com/2007/05/30/pygame-tutorial-3-mouse-events/
5. Pygames egen tutorial om hændelser mm. : http://inventwithpython.com/chapter18.html

12.5 Trin 10: Oprettelse af Gold-klassen

12.5.1 Skelet:

```
Intet skelet udleveres.
```

12.5.2 Beskrivelse:

Nu hvor vi har en spiller og en bane med mure kan du oprette præmier som din spiller kan samle op. Som det både var tilfældet med spilleren og væggene skal du oprette en klasse til at modellere dine præmier. // Udover de standardattributter som klassen skal have (billede, rect etc.), bør du overveje hvordan denne klasse skal relatere sig til Player-klassen. Hvad skal der ske når spilleren samler en præmie op? Hvilken klasse skal opdateres i det tilfælde? // Når du har oprettet klassen skal du (selvfølgelig) sørge for at den bliver tilføjet til data.py det rigtige sted. Du skal dog også kigge på hvor du kan initialisere dine præmier, så at de bliver tegnet i den bane der bliver spillet på.

12.5.3 Hints:

1. Overvej helt præcis hvad der skal ske når din spiller samler en præmie op. Har du brug for en metode til at simulere det?
2. Er der andre klasser der skal opdateres nu hvor du har en præmie-klasse?
3. Overvej hvilken del af programmet der lagrer information om hvad der er til stede på selve banen. Hvordan kan du sørge for at dine præmier bliver initialiseret dér?

12.5.4 Links:

1. Der er ingen links til dette trin.

13 Kildekode

```
1
2  import random
3  import pygame
4  import direction, data, resources, functions, ai
5
6  class Wall(pygame.sprite.Sprite):
7      def __init__(self, x, y):
8          pygame.sprite.Sprite.__init__(self)
9          self.image, self.rect = resources.load_image("wall.png
               ")
10         self.rect.move_ip(x,y)
11
12         self.add(data.walls)
13
14
15 class Gold(pygame.sprite.Sprite):
16     def __init__(self, x, y):
17         pygame.sprite.Sprite.__init__(self)
18         self.image, self.rect = resources.load_image("gold.png
               ",-1)
19         self.rect.move_ip(x,y)
20
21         self.add(data.gold)
22         self.add(data.entities)
23
24
25 class Monster(pygame.sprite.Sprite):
26     def __init__(self, x, y):
27         pygame.sprite.Sprite.__init__(self)
28         self.image, self.rect = resources.load_image("monster.
               png", -1)
29         self.rect.move_ip(x,y)
30
31         self.add(data.entities)
32         self.add(data.monsters)
33
```

```
34          self.SPEED = 3
35          self.SENSES = 100
36          self.timetochangedirection = data.gamespeed * 10
37          self.timeleft = self.timetochangedirection
38
39      def update(self):
40
41          if functions.distance_between(self.rect,data.player.
                rect)> self.SENSES:
42              if self.timetochangedirection <= 0:
43                  self.direction = direction.random()
44
45              self.timeleft -= 1
46
47          else:
48              self.move(direction.to_vector(ai.pathfinding(self.
                    rect, data.player.rect),self.SPEED))
49
50
51      def move(self, vector):
52          rect = self.rect
53          self.rect = self.rect.move(vector[0], vector[1])
54          if len(pygame.sprite.spritecollide(self,data.walls,
                False)) > 0:
55              self.rect = rect
56
57
58  class Player(pygame.sprite.Sprite):
59      def __init__(self, x, y):
60          pygame.sprite.Sprite.__init__(self)
61          self.image, self.rect = resources.load_image("player.
                png",-1)
62          self.rect.move_ip(x,y)
63          self.add(data.entities)
64          self.direction = direction.NONE
65          self.SPEED = 2
66          self.gold = 0
67          self.add(data.entities)
68          data.player = self
69
70      def update(self):
71          vector = [0,0]
72          if self.direction == direction.NONE:
73              vector = [0,0]
74          elif self.direction == direction.EAST:
75              vector = [self.SPEED,0]
76          elif self.direction == direction.WEST:
77              vector = [-self.SPEED,0]
78          elif self.direction == direction.NORTH:
79              vector = [0,-self.SPEED]
80          elif self.direction == direction.SOUTH:
81              vector = [0,self.SPEED]
82
83          self.move(vector)
```

```
84          if len(pygame.sprite.spritecollide(self,data.monsters,
                False)) > 0:
85              functions.quit()
86
87      def move(self, vector):
88          rect = self.rect
89          self.rect = self.rect.move(vector[0], vector[1])
90          if len(pygame.sprite.spritecollide(self,data.walls,
                False)) > 0:
91              self.rect = rect
92
93          self.gold += len(pygame.sprite.spritecollide(self,data
                .gold, True))
```