Dania
Academy of Higher Education

Academy Profession (AP) Degree in Computer Science

TITLE:

Breakout in Unreal Engine 4

PROJECT PERIOD:
GrnXXdatXX,
December 2015

PROJECT GROUP:
XX

STUDENTS:
Nichlas Bruun
Mathias Forsberg
Bjarne Kristensen

SUPERVISOR:
Jonathan

COPIES: 1

REPORT PAGES: 22

APPENDIX PAGES: 1

TOTAL PAGES: 23

ABSTRACT:

Problemformuleringsspørgsmålet?

Hvorfor dette emne er spændende.

Hvad vi har lavet og fundet ud af.

Indhold

1 Forord **1**

2 Problemformulering **3**
2.1 Problemformulering 3
2.2 Afgrænsning 3

3 Introduktion **5**
3.1 Unified Process 5
3.2 Process 5
3.3 Versionsstyring 5
3.4 Object Orienteret Analyse og Design 6

4 Object Orienteret Analyse **7**
4.1 Systemdefinition 7
4.2 Funktionstabel 8
4.3 Klasse Diagram 8
4.4 Hændelsestabel 8
4.5 Use Cases 9

5 Object Orienteret Design **13**
5.1 Gameplay 13
5.2 Grafik 13
5.3 Blueprints 14

6 Implementering **15**

7 Testing **17**

8 Reflektion **19**

9 Konklusion **21**

Referencer **22**

Kapitel 1

Forord

Kapitel 2

Problemformulering

2.1 Problemformulering

Hvordan udvikler man et breakout type spil i Unreal Engine 4?

- Hvilke analytiske værktøjer vil være værd at benytte?
- Hvilke systemudviklingsværktøjer er gode at bruge i sådan et projektforløb?
- Hvilke Unreal Engine 4 værktøjer kan optimere udviklingen?

2.2 Afgrænsning

Da projektets varighed er 4 uger, vil et fuldt testet og balanceret spil nok ikke kunne nås at lave, samtidigt med der skal skrives en systemudviklingsrapport. Der er fokus på koden i spillet, så grafikken er ikke nødvendigvis noget der vil blive brugt tid på, lyddessign er heller ikke en prioritet. Der vil heller ikke være tid til markedsanalyse, dette er fravalgt grundet produkt og systemudvikling er i fokus. Undervisningen har bestået af en uge, og herefter et projektforløb på en uge. Derfor vil noget af tiden også blive benyttet, til at blive familiær med nogle af unreal engine 4's værktøjer.

Kapitel 3

Introduktion

3.1 Unified Process

3.2 Process

Udviklingsforløbet er der planlagt at benytte unified process, og eksamens projektet kommer til at illustrerer en enkelt UP-iteration. I denne iteration vil der være fokus på planlægning og kode. Planlægningen vil bestå både af UP-, objektorienteret analyse- og objektorienteret design-værktøjer. Nogle af de værktøjer der vil være taget i brug, er f.eks. GANT-chart til tidsplanlægning og et analyse-diagram. Disse værktøjer og deres brug, vil blive beskrevet i systemudviklingsrapporten. Efter planlægningen og rapportskrivningen vil udviklingsprocsessen træde i kraft, hvor selve spillet vil blive udviklet. Efter udviklingsprocessen vil der være en kort testfase, hvor diverse fejl og mangler vil blive udredt. Til sidst vil udviklingsprocesen samt reflektioner og konklusioner blive ajourført i systemudviklingsrapporten. En kort opsumering i opremsning: Planlægning, rapportskrivning, udvikling, test og til sidst rapportskrivning.

3.3 Versionsstyring

Versionsstyring er et værktøj, der gør det nemt for flere personer at arbejde på de samme filer, og gør at der er backups man kan vende tilbage til. Ved versionsstyring er der altid mulighed for at tage en ældre version af filerne, hvis der skulle være noget som er gået galt i en nyere version. Når der er to eller flere som sidder og arbejder på samme program, gør versionsstyring det muligt at sammenflette text baserede filer automatisk, men også manuelt skulle der være konfikter på linier.

Vi har anvendt versionstyring til udarbejdelsen af rapporten, men ikke spillet. Rapporten består af mange rå tekst filer, og disse er optimale at anvende versions-

tyring på. Spillet består til gengæld af blueprints og levels til Unreal Engine 4, og ved disse har vi ikke selv kontrol over tekst linier som ved ren kode. Derfor kan vi ikke bruge vores egne versionsprogrammer til at sammenflette koden mellem 2 udgaver af den samme level eller blueprint. Unreal engine har et versionstyring integreret som hedder Perforce, så hvis man skal bruge det skal man tilmelde sig en tjeneste der passer til det. Eller selv betale for at sætte en server op og betale licenser. Det virker ikke til at være en færdig feature i Unreal Engine 4 endnu, og det ser ud til at der ofte er spærgsmål på deres forum om hvordan det virker og at det crasher eller ikke gør noget[1]. Vi har valgt at bruge Github som depot for vores rapport tekst, da det er gratis at anvende ved offentlige projekter. Skal koden være privat skal man betale for en opgraderet bruger[3]. Github har også selv deres egen git klient som man kan bruge, men man kan også bruge andre programmer der understøtter git protokollen.

3.4 Object Orienteret Analyse og Design

Kapitel 4

Object Orienteret Analyse

4.1 Systemdefinition

Functionality
-Bevægelse af paddle, Fjernelse af bricks, Scoring, Liv.
Application Domain
-Klassisk arcade spil, i første iteration fokus på kode og funktionalitet.
Condition
-Udvikles til windows, baseret på klassisk arcade spil.
Technology
-Udvikles i Unreal Engine 4, styres med keyboard, ikke krav til kraftig PC.
Objects
-Paddle, ball, Bricks.
Responsibility
-Et virkende spil.

4.1.1 Sytemdefinition

Unreal Breakout er et klassisk arkade spil, som giver spilleren mulighed for at opleve det klassiske gameplay fra Atari Breakout. Som består af at spilleren bevæger en "paddle"i bunden af skærmen, for at forsøge at holde en bold inden for skærmens rammer, samtidigt med at der skal opnås point ved at fjerne "bricks"i toppen af skærmen. Der vil være fokus på spillets kode og funktionalitet i første UP-iteration. Spillet udvikles til windows pc i Unreal Engine 4, og styres med keyboard. Spillet vil kunne køre på en windows-pc købt indenfor de sidste 5 år. Ansvaret over for spilleren vil i første UP-itereation være et meget basalt virkende spil.

Tabel 4.1: Funktionstabel

Navn	Kompleksitet	Type
Start Game	simpel	update
Exit Menu	simpel	update
Exit Game	simpel	update
Release Ball	medium	update / compute
Bounce Ball	medium	update / compute
Move Paddle	simpel	update / compute
Break Brick	simpel	update
Change Score	simpel	update / compute
Show Score	simpel	read

4.2 Funktionstabel

Vi har udarbejdet en funktionstabel (se tabel 4.1) ud fra de funktioner, som vi har planlagt skulle indgå i programmets første iteration. Vi har inddelt disse efter navn, kompleksitet og type. Navnet er hvad funktionen kommer til at omhandle og kompleksitet er hvor udfordrende, vi forventer, det bliver at programmere den pågældende funktion. Typen er til at bestemme, om det er noget der skal opdateres, læses, sende et signal eller beregne noget. Vi har ikke noget komplekst i vores spil, men vi har to funktioner som vi regner med er medium, og det er relateret til boldens interaktion med andre objekter. De andre funktioner virker til at være simple at implementere.

4.3 Klasse Diagram

4.4 Hændelsestabel

En hændelse, i objekt orienteret kontekst, er en situation hvor en eller flere objekter er involveret. Det er f.eks. hændelser hvor spilleren giver noget input som objekter i spillet skal agere på. Det kan også være hændelser i spillet som sker når 2 objekter kolliderer med hinanden og en aktion ønskes. I tabel 4.2 er der opstillet hvilke hændelser vi regner med vil ske, og hvilke objekter kan være involveret i disse hændelser.

Tabel 4.2: Hændelsestabel

Events/Klasser	Paddle	Ball	Brick	Menu	Gameworld	Camera	UI
Player clicks on menu obj				X			
Player moves paddle	X				X		
Player releases ball	X	X			X		
Ball collides with object	X	X	X		X		
Score updates		X	X		X		X

4.5 Use Cases

Start Spillet:

Scope:
i menuen.

Description:
Spilleren trykker på *play*-knappen, med musen, i menuen for at komme til *gameworlden*.

Preconditions:
Spilleren skal befinde sig i menuen for at use casen kan startes.

Success Guarantee:
Når use casen er opfyldt vil spillet gå fra menuen til *gameworlden*.

Main Success Scenario:
Spilleren befinder sig i menuen og klikker med musen på *play*-knappen og *gameworlden* vises frem på skærmen.

Extensions:
Befinder spilleren sig allerede i *gameworlden* vil det ikke være muligt at trykke på play knappen da den ikke er vises.

Luk Spillet.

Scope:
i menuen.

Description:

Spilleren kan lukke spillet ved at trykke på *exit*-knappen med musen.

Preconditions:
Spilleren skal befinde sig i menuen for at use casen kan startes.

Success Guarantee:
Når use casen er opfyldt vil spillet blive lukket.

Main Success Scenario:
Spilleren befinder sig i menuen og klikker på *exit*-knappen, med musen, hvorefter spillet vil lukke.

Extensions:
Befinder spilleren sig i *gameworlden* vil *exit*-knappen ikke være vist og spilleren kan derfor ikke interagere med den.

Bevægelse af paddle.

Scope:
I *Gameworld*.

Description:
Spilleren kan ved hjælp af højre og venstre piletast bevæge *paddlen* til henholdsvis højre og venstre.

Preconditions:
Spilleren har startet spillet igennem menuen.

Success Guarantee:
Bliver use casen opfyldt korrekt vil *paddlen* blive bevæget til siderne i takt med at spilleren trykker piletasterne ned.

Main Success Scenario:
a. Spilleren trykker højre piletast ned og *paddlen* bevæger sig til højre.
b. Spilleren trykker venstre piletast ned og *paddlen* bevæger sig til venstre.

Skyd.

Scope:
I *Gameworld*.

Description:
Bolden skydes væk fra *paddlen* ved brug af *space*-knappen, dette kan kun gøre når spillet startes eller efter bolden har været uden for spilbanen.

Preconditions:
Spilleren har startet spillet igennem menuen og har ikke skudt endnu, eller bolden har lige været udenfor banen.

Success Guarantee:
Bliver use casen opfyldt korrekt vil bolden blive skudt væk fra *paddlen*.

Main Success Scenario:
Spilleren har startet spillet og bolden er stadig placeret på *paddlen*, eller bolden har lige været uden for spilbanen. spilleren trykker på *space*-knappen og bolden bliver skudt væk fra *paddlen*.

Extensions:
Bolden er blevet skudt væk fra *paddlen* og har ikke været udenfor spilbanen og kan derfor ikke blive skudt fra *paddlen* igen da den ikke er placeret der.

Gå tilbage til menuen.

Scope:
i *Gameworld*.

Description:
Spilleren går tilbage til menuen fra *gameworlden* ved at trykke på *escape*-knappen.

Preconditions:
Spilleren har startet spillet og befinder sig i *gameworlden*.

Success Guarantee:
Bliver use casen opfyldt korrekt vil spilleren blive vist menuen igen og *gameworlden* vil lukke.

Main Success Scenario:

Spilleren har startet spillet og trykker på *escape*-knappen, herefter vil menuen blive vist og *gameworlden* vil stoppe.

Extensions:
Befinder spilleren sig allerede i menuen vil *escape*-knappen ikke have nogen effekt.

Kapitel 5

Object Orienteret Design

5.1 Gameplay

Unreal Breakout er et spil med et bat[1], en bold[2], en masse brikker[3], tre lukkede sidder, og en åben side. Formålet er at skyde bolden op og ødelægge alle brikkerne og sørge for at bolden ikke ryger ud af banen i bunden hvor spillerens bat og den åbne side er. Når man rammer en brik, går den i stykker eller skifter farve, og så får man point tilføjet sin score. Når spillet starter har man 3 forsøg(bolde) og bolden er låst fast på éns bat i midten. Når man trykker på en dertil bestemt knap skyder man bolden afsted, og med nogle andre knapper bevæger man battet fra side til side for at undgå at bolden ryger ud af spilområdet. Hvis bolden ryger ud af spilområdet, mister man et forsøg(bold) og en ny bold bliver låst fast til batet og klar til at blive skudt afsted på ny. Når man har opbrugt alle forsøg og den sidste bold ryger ud af spilområdet, slutter spillet og man kan se sin score. Får man til gengæld alle brikker fjernet vinder man banen og den samme bane kommer igen uden at ændre på antal forsøg man har tilbage eller nulstille éns score, og bolden bliver igen fastlåst til batet.

5.2 Grafik

Spillets grafik er fundet på *opengameart.org*, som er en hjemmeside hvor folk lægger grafik op til fri afbenyttelse. Dette betyder at det er uden licens men man kan dog donere penge til de brugere der har lagt grafikken tilgængelig på siden.
Breakout er et gammelt spil udviklet af Atari og derfor er der fundet grafik som er tro mod det klassiske spils grafik. Grafikken ligger nogle spritesheets som er kollager

[1]Paddle på engelsk, bliver brugt i vores paddle klasse.
[2]Ball på engelsk, bliver brugt i vores ball klasse.
[3]Bricks på engelsk, bliver brugt i vores bricks klasse.

af billeder som bliver brugt til at skabe de forskellige elementer i spillet. Når spritesheetet er delt op vil de forskellige brikker blive bygget op til et level. Eftersom spilleren rammer brikkerne med bolden vil de skifte farve, farverne indikere hvor mange gange en brik skal rammes for at blive ødelagt. De grafiske elementer er ikke noget der vil blive sat stor fokus på i spillet, da det er spillets gameplay der skal være grundpillen i det.

5.3 Blueprints

Blueprints er en speciel type asset som giver en node baseret interface til at lave nye klasser og udvide klasser, så man kan scripte objekter i levels og widgets. Blueprints er et værktøj der giver designers og gameplay programmører mulighed for hurtigt at kunne iterere deres idéer uden at skulle skrive kode.

Det er stadig mulig selv at oprette klasser med C++ kode uden at bruge blueprints. Blueprints kan bruges til at udvide klasser som et skrevet i C++, gemme og modificeret properties, og håndtere objekt instancer i klasser.

Ligesom i C++ og C# har blueprints member variables/fields, member functions, og en constructor.

Der er 3 typer Blueprints:

Blueprint Class

Data-Only Blueprint

Level Blueprint

Blueprint Class er den blueprint man bruger når man vil have et object til at gøre noget inde i en level. Man laver noget funktionalitet i den og tilføjer den til et objekt i en level. Data-Only Blueprints indeholder kun nodes, variabler og componenter som den har nedarvet og der kan ikke tilføjes nyt. Det den bruges til er at ændre på objekter i en level og lave variationer som alle har samme interface, men forskellig adfærd. Level Blueprint er en blueprint som altid findes når man opretter en ny level. Det er et overordnet blueprint som eksisterer sammen med de Blueprints som sidder på objekter i banen. Man kan bruge Level Blueprint til at oprette events som andre objekters blueprints kan agere på. Man kan også oprette instancer af andre assets gennem Level Blueprint, som f.eks. et UI der skal tegnes i Camera Actor'en i banen. Se [2] for mere information.

Kapitel 6

Implementering

Kapitel 7

Testing

Kapitel 8

Reflektion

Kapitel 9

Konklusion

Referencer

[1] Epic Games. *Blueprint Merge Topics*. 2015. URL: https://answers.unrealengine.com/questions/topics/merge.html.

[2] Epic Games. *Blueprint Overview*. 2015. URL: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/Overview/index.html.

[3] GitHub. *GitHub Terms of Service*. 2015. URL: https://help.github.com/articles/github-terms-of-service/.