Slutrapport för Breakout

Introduktion till Datalogi

Anders Hagward och Fredrik Hillnertz

Innehåll

1 Programbeskrivning

1.1 A

Programmet skall underhålla och stjäla tid från användaren. Detta genom att låta denne styra ett racket för att upprepade gånger studsa en boll mot ett antal förstörbara block. Lär även vara gynnsamt för användarens pricksäkerhet och reflexer.

1.2 B

Programmet har lyckats uppfylla A, men erbjuder också så kallade "power ups" som ändrar spelupplevelsen under spelets gång och således kan verka ännu bättre för att skärpa användarens sinnen. Det kan till exempel röra sig om att bollen åker fortare eller långsammare, dödliga dödskallar som faller från skyn, extrabollar som gör entré eller liknande.

1.3 C

Egentligen är det inga direkta skillnader mellan programbeskrivningarna. "Power ups":en gör bara att användaren får koncentrera sig ännu mer än beräknat för att inte gå hädan, samt att spelet (förmodligen) är ännu bättre på att underhålla och själa tid.

2 Användarbeskrivning

2.1 A

Användaren förutsätts ha "lagom" datorvana, det vill säga vetskap om hur man startar en dator, kör ett program och interagerar med mus och tangentbord. En ungefärlig åldersskala är 6-60 år. Användaren bör även ha grundläggande kunskaper i det engelska språket då menyerna och hjälpavsnittet är på engelska.

2.2 B, C

Användarbeskrivningen har ej ändrats.

3 Användarscenarier

3.1 A

Nedan följer två vanliga scenarier.

3.1.1 Scenario I

1. Användaren startar programmet och möts av en huvudmeny med menyvalen *New game*, *Settings*, *Help* och *Quit*.
2. Användaren väljer det första alternativet och en ny vy framställs på skärmen. Denna består av ett racket längst ned som går att styra med musen, en boll fastsittandes mitt på racketet, en hord av förstörbara block längst upp samt solida väggar och tak.
3. Användaren trycker på vänster musknapp och får bollen att fara iväg uppåt, studsandes mot blocken, väggarna och taket.
4. Blocken försvinner successivt i och med att bollen studsar på dem.
5. Alla block har slutligen försvunnit och användaren vinner spelet.
6. En ny 'karta' presenteras (det vill säga, blocken är uppställda i en annan formation än tidigare).
7. Användaren väljer att avsluta spelet genom att trycka `Esc` för att få upp spelmenyn och sedan välja *Quit*.

3.1.2 Scenario II

1. Användaren startar programmet och möts av samma huvudmeny som i föregående scenario.
2. Användaren väljer menyvalet *Help* och får reda på information om hur spelet fungerar.
3. Användaren backar tillbaka till huvudmenyn genom att trycka `Esc`.
4. Användaren väljer menyvalet *Settings* och väljer att stänga av ljudet.
5. Användaren backar återigen till huvudmenyn på samma vis som tidigare men väljer nu *New game*.
6. Samma spelvy som i föregående scenario visas och användaren spelar på samma sätt.
7. Användaren är inte så pricksäker och lyckas inte få bollen att studsa på racketet. För varje gång han missar förlorar han ett 'liv'.
8. Användaren förlorar tre liv och därmed även spelet.
9. En meny med valen *Play again*, *Settings*, *Help* och *Quit* visas. Användaren väljer det sistnämnda.

3.2 B

Nedan följer de två scenarierna från A, fast som de slutligen blev.

3.2.1 Scenario I

1. Användaren startar programmet och möts av en huvudmeny med menyvalen *New game*, *High Score*, *Settings*, *Help* och *Quit*.
2. Användaren väljer det första alternativet och en ny vy framställs på skärmen. Denna består av ett racket längst ned som går att styra med musen, en boll fastsittandes mitt på racketet samt en hord av förstörbara block längst upp.
3. Användaren trycker på vänster musknapp och får bollen att fara iväg uppåt, studsandes mot blocken, väggarna och taket.
4. Blocken försvinner successivt i och med att bollen studsar på dem.
5. Alla block har slutligen försvunnit och användaren har därmed besegrat den nuvarande 'kartan'.
6. En ny karta presenteras (det vill säga, blocken är uppställda i en annan formation än tidigare).
7. Användaren väljer att avsluta spelet genom att trycka `Esc` för att få upp spelmenyn, välja *Back to Main Menu* för att komma till huvudmenyn och där välja *Quit* igen för att avsluta spelet.

3.2.2 Scenario II

1. Användaren startar programmet och möts av samma huvudmeny som i föregående scenario.
2. Användaren väljer menyvalet *Help* och får reda på information om hur spelet fungerar.
3. Användaren backar tillbaka till huvudmenyn genom att trycka med musen på knappen `Back`.
4. Användaren väljer menyvalet *Settings* och väljer att stänga av ljudet.
5. Användaren backar återigen till huvudmenyn på samma vis som tidigare men väljer nu *New game*.
6. Samma spelvy som i föregående scenario visas och användaren spelar på samma sätt.
7. Användaren är inte så pricksäker och lyckas inte få bollen att studsa på racketet. För varje gång han missar förlorar han ett 'liv'.

8. Användaren förlorar tre liv och därmed även spelet.

9. En meny med valen *Play Again* och *Back to Main Menu* visas. Användaren väljer det sistnämnda för att komma tillbaka till huvudmenyn, och väljer där återigen *Quit* för att avsluta spelet.

3.3 C

För det första så är huvudmenyn utökad med ännu ett alternativ, nämligen *High Score* som föga förvånande tar spelaren till listan över de högsta uppnådda poängen (detta gäller båda scenarierna). Den andra märkbara skillnaden är att *Back to Main Menu*-alternativet i den så kallade "in game"-menyn som tar spelaren tillbaka till huvudmenyn har ersatt *Quit*-alternativet, som skulle avslutat hela programmet (gäller också båda scenarierna). De två sista delarna som skiljer sig återfinns i Scenario II och är att man inte 'backar' i menyerna med `Esc`-knappen samt att alternativen som visas när man förlorat är färre.

4 Testplan

4.1 A

Förutom de två scenarierna ovan kommer menyerna testas ordentligt, så alla menyval fungerar korrekt. Inställningsmöjligheterna kommer testas, så att ljudeffekterna verkligen försvinner när man gör den inställningen, och verkligen kommer tillbaka om man gör inställningen igen.

Själva spelmekaniken kommer givetvis testas grundligt så att inte bollen verkligen studsar mot väggarna, taken, blocken och racketet, så att blocken försvinner vid kontakt med bollen, så att ett 'liv' försvinner och att bollen återställs på racketet vid varje 'miss'. Vidare skall testas så att rätt meny med fungerande menyval dyker upp vid förlorat spel, så att en ny bana laddas när alla block är borta, så att en poänglista visas när man klarat alla fem banor och att den nya poängen sparas.

Självklart skall spelet testas av en tredjepart.

4.2 B

Vid det första användartestet läts en kille i 20-årsåldern med god datorvana testa spelet. Han provade samtliga olika menyalternativ, och kollade även snabbt igenom hjälptexten med kontrollerna, innan han startade ett nytt spel. Väl igång fick han i uppdrag att få ett poäng högre än 100,000 och komma med på highscorelistan, någonting han inte hade några problem att klara av. Vi noterade dock följande saker under spelets gång:

- Han kunde inte lista ut vad "fireball"-powerup:en gjorde enbart baserat på dess ikon.
- Poängen skall långsamt räkna ned hela tiden för att motivera spelaren att klara av kartorna så fort som möjligt, men ibland slutade den helt plötsligt med detta vilket förvirrade testpersonen. En bugg funnen med andra ord.
- Han skulle mitt i spelet pausa spelet genom att öppna "in game"-menyn, vilket han misslyckades med då en sådan inte var implementerad vid testtillfället. Vi noterade dock att han försökte få fram den genom att trycka på `Esc`-tangenten utan att det var förklarat någonstans vilket indikerar på bra val av tangent.
- När han klarat alla banor och skulle ange sitt namn poppade inmatningsdialogen upp i *bakgrunden* istället för längst fram vilket gjorde att spelet först såg ut att ha kraschat. Testpersonen insåg dock snabbt vad som hade hänt – datorvan som han är – och letade upp inmatningsrutan.
- Medan dialogen för inmatning av namn var uppe fick han syn på att det i spelet stod att han fortfarande hade ett liv kvar vilket han snabbt påpekade. Det visade sig att texten inte uppdaterar sig sista gången vilket kan skapa förvirring.

Den andre testpersonen var också en datorvan typ i 20-årsåldern. Liksom testperson nummer ett försökte han få upp en meny medan han spelade, vilket konstaterats inte fungerar än, men han kommenterade även följande:

- Han tyckte det var svårt att gissa vad de olika powerups:en gjorde.
- Det var oklart när han kunde skjuta laser eller inte.
- När det blev många objekt på skärmen – många bollar och powerup:s i rörelse – tenderade det att bli rörigt, och svårt att snabbt avgöra vilka powerup:s att ta och vilka att låta bli.

4.3 C

Skillnaden mellan den planerade testningen och den utförda är marginella. Den enda egentliga är att uppdraget "få mer än 100,000 poäng" lades till vid det första användartestet för att motivera användaren.

Användartestningen har dock givit oss en hel del värdefull information om hur spelet kan förbättras:

- Vi har insett att det kan vara en bra idé att förklara vad de olika powerups:en gör i hjälpmenyn, så att användaren kan känna igen dem och ha ett hum om vad de gör när spelet väl är igång.

- Indikationen på att man kan skjuta laser kan göras tydligare.
- De upptäckta buggarna skall lösas.

5 Programdesign

5.1 A

Programmet kommer bygga på spelbiblioteket *Slick*. Nedan följer en lista på de grundläggande klasserna och dess viktigaste metoder.

- `MainMenuState, GameplayState`: Då biblioteket Slick använder sig av så kallade *states*, eller *tillstånd*, kommer minst två 'state'-klasser implementeras. Deras viktigaste metoder är `init`, där alla bilder och ljud laddas, `render` där allt ritas upp på skärmen och `update` där alla beräkningar sker (för kollisioner, positionsuppdateringar med mera).
- `GameObject`: En abstrakt klass som representerar ett objekt på skärmen. Viktigaste metoder: `getImage, setImage` som hanterar objektets bild, `getXPos, getYPos` returnerar objektets position.
- `Ball, Block, Racket`: Klasser som ärver `GameObject`.
- `Movable`: Ett interface med metoden `move`, där implementerande klasser får beskriva hur objektets position skall uppdateras.
- `Highscore`: Sköter all skrivning och läsning till och från textfilen där poängen sparas. Viktigaste metoder är `addScore` som lägger till ett poäng, `getScores` som returnerar en lista med poängen.
- `BreakoutGame`: 'Huvudklassen' som innehåller `main`-metoden.

5.2 B

Vi höll oss till biblioteket Slick, men klasstrukturen skiljer sig en del från den ursprungligen planerade:

- `MainMenuState, GameplayState, HighScoreState`: Se motsvarande förklaring i A-delen.
- `LevelHandler`: Har hand om allt som rör kartorna; ladda in en karta från en fil och hålla koll på alla objekt på den (som bollar, block och så vidare).
- `Ball, Block, Racket, PowerUp`: Klasser som representerar synliga spelobjekt. Alla dessa ärver Slick-klassen `Shape` på ett eller annat sätt, samt implementerar gränssnittet `Movable` för att kunna röra på

sig. Förutom de positionsrelaterade metoderna ärvda från `Shape` har de den viktiga metoden `draw`, som ritar ut objektet på rätt plats på skärmen.

- `Movable`: Ett interface med metoden `move`, där implementerande klasser får beskriva hur objektets position skall uppdateras.
- `HighscoreHandler, Score`: Den förstnämnda har hand om själva highscorelistan, och att läsa och skriva den till en fil. Varje poäng representeras av ett `Score`-objekt, som innehåller poängen samt namnet på spelaren som uppnått dem.
- `Player`: Representerar spelaren med ett namn, hur många liv spelaren har kvar samt dennes poäng.
- `SoundPlayer`: Sköter uppspelning av ljud. Har en metod för varje ljud som används i spelet.
- `ResourceLoader`: Laddar in alla bilder för snabb åtkomst. Har den ovärderliga metoden `load` som gör detta.
- `BreakoutGame`: 'Huvudklassen' som innehåller `main`-metoden.

5.3 C

De huvudsakliga skillnaderna i punktform:

- Ett så kallat 'state' lades till, nämligen `HighScoreState` som presenterar poänglistan.
- Klassen `LevelHandler` skapades för att ha hand om allt som rör kartorna.
- Så kallade "power ups" lades till i spelet, där varje sådan representeras av ett `PowerUp`-objekt.
- Den abstrakta klassen `GameObject` togs helt bort och ersattes med Slick-bibliotekets `Shape`-underklasser, detta för att göra kollisionsdetekteringen mer solid.
- Den tänkta klassen `Highscore` förkastades och istället skapades `HighScoreHandler` samt `Score`.
- Klassen `Player` skapades för att representera spelaren.
- Klassen `SoundPlayer` skapades för ljuduppspelning.
- Klassen `ResourceLoader` skapades för bildinladdning.

6 Tekniska frågor

6.1 A

Nedan följer en lista med tänkbara tekniska problem och eventuella lösningar.

Problem: Vid eventuellt bildbyte på objekt medan spelet körs (till exempel om ett block kräver flera bollstudsar för att försvinna) skulle en viss fördröjning kunna uppstå innan programmet har läst in den nya bilden. **Ev. lösning:** Ladda in flera bilder för ett objekt vid initiering?

Problem: Hur 'kartorna' skall representeras. **Ev. lösning:** Läsa in dem från filer?

Problem: Exakt vilka klasser som skall finnas och vad var och en skall sköta i spelet, samt undvika att skapa klasser det redan finns motsvarigheter till i spelbiblioteket. **Ev. lösning:** Träna objektorientering, tänka efter ordentligt och läsa bibliotekets dokumentation.

Problem: Hur skall programmet bete sig om någon väsentlig fil saknas? **Ev. lösning:** Ljudfil - meddela användaren, highscorefil - skapa en ny fil utan att meddela användaren, kartfil eller bildfil - meddela användaren och avsluta.

6.2 B

Problem: Vid eventuellt bildbyte på objekt medan spelet körs (till exempel om ett block kräver flera bollstudsar för att försvinna) skulle en viss fördröjning kunna uppstå innan programmet har läst in den nya bilden. **Löstes så här:** Klassen `ResourceLoader` skapades som vid programstart får ladda in samtliga bilder för att ha dem i beredskap.

Problem: Hur 'kartorna' skall representeras. **Löstes så här:** Kartorna skapades med hjälp av programmet *Tiled Map Editor* (http://www.mapeditor.org/), och informationen om dem sparades i XML-format. För att ladda en bana läses bara dess respektive XML-fil in och `Block`-objekt skapas efter informationen i filen.

Problem: Exakt vilka klasser som skall finnas och vad var och en skall sköta i spelet, samt undvika att skapa klasser det redan finns motsvarigheter till i spelbiblioteket. **Löstes så här:** Själva konceptet 'objektorientering' har vi fått träna på allteftersom arbetet fortskridit för att inte koden skall bli alltför gräslig. Slicks dokumentation har studerats mer eller mindre som planerat.

Problem: Hur skall programmet bete sig om någon väsentlig fil saknas? **Löstes så här:** Ljudfil - meddela användaren, highscorefil - skapa

en ny fil utan att meddela användaren, kartfil eller bildfil - meddela användaren och avsluta.

Problem: Hur skall vi få spelet att köras lika snabbt på olika kraftfulla datorer? **Löstes så här:** Använde metoderna `setMinimumLogicUpdateInterval` och `setMaximumLogicUpdateInterval` från Slick-klassen `GameContainer` för att precisera uppdateringsintervallet.

6.3 C

Det 'enda' större fundamentala problemet som stöttes på som inte tagits med i projektplanen var det sista i listan i B-sektionen; nämligen hur man skulle få spelet att köras lika snabbt vare sig datorn i fråga är av modellen snabbare eller långsammare.

7 Arbetsplan

7.1 A

Arbetet kommer delas upp på det viset att var och en får några bestämda klasser att implementera. De skall implementeras efter en specifikation som bestäms i förväg så att båda har koll på vad de gör och vad de skall användas till. Om eventuella problem skulle uppstå och en fastnar med någon del av implementationen kan givetvis den andre hjälpa till.

Arbetsplanen är upplagd på följande vis:

7 - 14 april: Nödvändiga klasser skall skapas och grundläggande spelmekanik implementeras. Spelet skall vara 'minimalt' spelbart, det vill säga att racketet skall kunna styras och bollen skall studsa korrekt mot de övriga objekten.

15 - 22 april: I första hand skall spelmekaniken putsas på. I andra hand skall ljud, poängsystem, inställningsmeny och hjälpmeny implementeras (så många som hinns med).

23 - 30 april: Slutrapporten skall skrivas och eventuell putsning av spelet skall göras inför prototypvisning.

1 - 7 maj: Eventuell implementation av detaljer som inte hunnits med samt finputsning.

7.2 B

Vi låg i fas med den förbestämda arbetsplanen i princip genom hela arbetet. Uppdelningen skedde spontant allt eftersom vad vi kände för att göra och

vad som behövde göras. I stora drag hade Anders hand om menyerna, kollisionshanteringen och poängsystemet, medan Fredrik fixade "power ups":en och kartorna.

7.3 C

Då arbetsplanen inte behövde ändras är det ingen skillnad mellan A och B.

8 Sammanfattning

Vi har lärt oss mycket av projektet, här är de viktigaste punkterna:

Versionshantering Då vi använde programmet *Git* för att hålla ordning på alla uppdateringar som gjordes har vi fått lära oss (grundläggande) om hur den typen av mjukvara fungerar och hur det är att arbeta tillsammans på ett projekt.

Tredjepartsbibliotek Som nämnt tidigare i texten användes spelbiblioteket *Slick* vilket har gjort oss tvungna att ta reda på hur man importerar ett tredjepartsbibliotek i Java (eller kanske främst hur det går till med utvecklingsverktyget *Eclipse*) samt gräva i dokumentationen för att få ett hum om hur det är uppbyggt.

Planering Vi har noterat att koden lätt svämmar över alla bredder om man inte sköter om den med jämna mellanrum, och framför allt har en tydlig bild om hur strukturen skall se ut. Att försöka hålla klasserna *löst kopplade* och undvika redundans är nödvändigt för att enklare kunna bygga ut och ändra i projektet. Detta är någonting vi kunde löst på bättre vis.