Projektplan: Flongout

Robin Gunning, Mikael Forsberg, Jonathan Yao Håkansson

Kurs DD1339, Introduktion till datalogi, VT 2015
Kungliga Tekniska Högskolan
Grupp 7
rgunning@kth.se, miforsb@kth.se, jyh@csc.kth.se

22 maj 2015

Sammanfattning

Flongout är ett spel för två spelare. Spelet bygger på element från flipperspel samt de klassiska spelen ”Breakout” och ”Pong”.

Innehåll

1 Programbeskrivning A

Flongout är ett spel för två personer. Spelet är en blandning av Pong och Breakout (Arkanoid) med flipperpaddlar. Med hjälp av ”paddlarna” ska man försöka ”göra mål”, alltså få in bollen bakom motståndaren. Till sin hjälp har man ett antal ”power-ups” som vänder spelet till sin fördel.

När programmet startas visas en meny där användaren får välja mellan att starta spelet, konfigurera kontroller och avsluta spelet. Väljer spelaren att konfigurera kontroller visas en skärm där man kan för vardera spelare välja mellan olika sätt att kontrollera paddeln.

När spelet startar så visas vardera spelares paddlar vid skärmens kortsidor, den ena till vänster och den andra till höger. Till en början befinner sig bollen i mitten av spelplanen och när en startknapp trycks ned så inleds en nedräkning. Efter nedräkningen skjuts bollen iväg med slumpvald riktning och hastighet, och spelet är i full gång! Spelets gång yttrar sig genom att spelarna försöker undvika att släppa in bollen på sin kortsida. Detta undviks genom att rotera och flytta sin paddel för att slå till bollen i önskad riktning.

Enligt ett visst intervall så tillförs ett antal brickor till spelplanen. Brickorna utgör ett fast hinder för bollen som vid kontakt tvingas till studs. Efter ett visst antal kontakttillfällen mellan boll och enskild bricka så förstörs brickan, och lämnar därmed spelplanen. Vissa typer av brickor kan efterlämna en så kallad ”power-up” då de förstörs. Denna visas grafiskt på skärmen i den position brickan tidigare fanns, för att sedan förflytta sig i riktning mot den kortsida där den spelare befinner sig som anslog bollen i det skede precis innan brickan förstördes. Spelaren får därmed uppgiften att med sin paddel infånga sagd power-up. Om spelaren lyckas fånga in en power-up så tillförs spelaren någon form av effekt. Det kan till exempel röra sig om att få en förstorad paddel, att få sina tillslag på bollen förstärkta eller att bollens storlek ändras.

Kommer bollen i kontakt med skärmens långsidor studsar bollen tillbaka in på spelplanen. Vid kontakt med en kortsida lämnar bollen spelplanen, och en poäng tilldelas den spelare som befinner sig vid motstående kortsida. Efter en kort paus startar en ny nedräkning innan bollen släpps iväg igen. Då en av spelarna har uppnått en viss mängd poäng avslutas matchen genom att spelarna inte längre tillåts kontrollera paddlarna, bollen plockas bort och en enkel meny visas. Denna meny tillåter spelarna att välja om de vill starta om eller återgå till huvudmenyn.

Det går även att pausa spelet under en pågående omgång. Då spelet pausas visas en meny snarlik som den som visas då en omgång är slut, utökad med ett menyval för att återuppta spelet. Om spelet återupptas sker först en nedräkning innan bollen börjar röra sig.

Spelarna kan välja mellan en rad olika sätt för att kontrollera paddlarna. Dessa inkluderar tangentbord, mus och handkontroll.

Figur 1: En mockup av spelet

2 Programbeskrivning B

Flongout är ett spel för *en till två* personer. Spelet är en blandning av Pong och Breakout (Arkanoid) med flipperpaddlar. Med hjälp av ”paddlarna” ska man försöka ”göra mål”, alltså få in bollen bakom motståndaren. Till sin hjälp har man ett antal ”power-ups” som vänder spelet till sin fördel. *Det förekommer även något vi kallar ”power-downs” som påverkar den egna paddeln på något negativt vis om man inte lyckas undvika dem.*

När programmet startas visas en meny där användaren får välja mellan att starta spelet, konfigurera kontroller och avsluta spelet. Väljer spelaren att konfigurera kontroller visas en skärm där man kan för vardera spelare välja mellan olika sätt att kontrollera paddeln, *samt för vissa av inmatningsmetoderna konfigurera vilka funktioner som tilldelas olika knappar och joystickaxlar*.

När spelet startar så visas vardera spelares paddlar vid skärmens kortsidor, den ena till vänster och den andra till höger. Till en början befinner sig bollen i mitten av spelplanen och ~~när en startknapp trycks ned så inleds en nedräkning~~ *en nedräkning inleds automatiskt*. Efter nedräkningen skjuts bollen iväg med slumpvald riktning och hastighet, och spelet är i full gång! Spelets gång yttrar sig genom att spelarna försöker undvika att släppa in bollen på sin kortsida. Detta undviks genom att rotera och flytta sin paddel för att slå till bollen i önskad riktning.

Enligt ett visst intervall så tillförs ett antal brickor till spelplanen. Brickorna utgör ett fast hinder för bollen som vid kontakt tvingas till studs. Efter ett visst antal kontakttillfällen mellan boll och enskild bricka så förstörs brickan, och lämnar därmed spelplanen. ~~Vissa typer av brickor kan efterlämna~~ *Då en bricka förstörs kan det skapas* en så kallad ”power-up”. Denna visas grafiskt på skärmen i den position brickan tidigare fanns, för att sedan förflytta sig i riktning mot den

kortsida där den spelare befinner sig som anslog bollen i det skede precis innan brickan förstördes. Spelaren får därmed uppgiften att med sin paddel infånga sagd power-up. Om spelaren lyckas fånga in en power-up så tillförs spelaren någon form av effekt. Det kan till exempel röra sig om att få en förstorad paddel, att få sina tillslag på bollen förstärkta eller att bollens storlek ändras. *Vissa power-ups (power-downs) har missgynnande effekter och blir därför istället en företeelse som spelarna försöker undvika.*

Kommer bollen i kontakt med skärmens långsidor studsar bollen tillbaka in på spelplanen. Vid kontakt med en kortsida lämnar bollen spelplanen, och en poäng tilldelas den spelare som befinner sig vid motstående kortsida. ~~Efter en kort paus startar en ny nedräkning~~ *Omedelbart inleds en ny nedräkning* efter vilken bollen släpps iväg igen. ~~Då en av spelarna har uppnått en viss mängd poäng avslutas matchen genom att spelarna inte längre tillåts kontrollera paddlarna, bollen plockas bort och en enkel meny visas. Denna meny tillåter spelarna att välja om de vill starta om eller återgå till huvudmenyn.~~ *Spelet fortsätter så länge spelarna önskar spela. Om spelarna önskar avsluta spelet trycker de fram pausmenyn, navigerar till menyvalet för avslut och bekräftar.*

Det går även att pausa spelet under en pågående omgång. Då spelet pausas visas en meny ~~snarlik som den som visas då en omgång är slut, utökad~~ med ett menyval för att återuppta spelet. Om spelet återupptas sker först en nedräkning innan bollen börjar röra sig.

Spelarna kan välja mellan en rad olika sätt för att kontrollera paddlarna. Dessa inkluderar tangentbord, ~~mus~~ och handkontroll.

3 Programbeskrivning C

Vi hade inte planerat att göra någon ”AI” / datormotståndare, men en enkel sådan tillkom plötsligt och fungerade så pass bra (speciellt vid speltestning) att vi behöll den. Därmed är Flongout nu ett spel man även kan spela för sig själv, även om datormotståndaren inte erbjuder någon större utmaning. ”Power-downs” kom som förslag på en utav övningarna och innebar en mycket enkel utökning av systemet med powerups.

Inte heller konfigurering av tangentbords- och knapplayouter var med i den ursprungliga planeringen. Behovet av detta blev dock snabbt påtagligt då vi under utvecklingsarbetet använde olika modeller av USB-handkontroll med vissa skillnader i beteende, exempelvis hade

de olika ID-nummer för inbördes motsvarande knappar och joystickaxlar.

Vi implementerade aldrig idén med att använda en startknapp för att få igång nedräkningen. Själva nedräkningen implementerades först, utan startknapp, och vi blev så pass nöjda med hur det fungerade att vi lät bli att införa startknappen.

Vi valde att inte införa någon poänggräns eller annan form av automatiskt avslutande av spelet. Valet gjordes antagligen främst på grund av att det inte fanns någon självklar sådan gräns. Mer om detta följer i avsnittet sammanfattning i slutet av detta dokument.

På grund av tidsbrist implementerades aldrig kontrollmetoderna mus respektive mus och tangentbord. Mer om detta följer även det i sammanfattningen senare.

4 Användarbeskrivning A

Som målgrupp för spelet ser vi personer med viss vana av dator- och tv-spel. Spelet kommer inte innehålla några egentliga förklaringar och kommer därmed att kräva en viss nivå av igenkännande hos användaren. Tanken är att spelet är självförklarande och då mängden innehåll är begränsat antas en person ur målgruppen ha sett och förstått nästan allt efter ett par provomgångar. Vid första anblick bör en person direkt kunna koppla utseendet hos bollen och paddlarna till flipperspel, och mer eller mindre direkt förstå deras syfte.

5 Användarbeskrivning B

Som målgrupp för spelet ser vi personer med viss vana av dator- och tv-spel. Spelet kommer inte innehålla några egentliga förklaringar och kommer därmed att kräva en viss nivå av igenkännande hos användaren. Tanken är att spelet är självförklarande och då mängden innehåll är begränsat antas en person ur målgruppen ha sett och förstått nästan allt efter ett par provomgångar. Vid första anblick bör en person direkt kunna koppla utseendet hos bollen och paddlarna till flipperspel, och mer eller mindre direkt förstå deras syfte.

6 Användarbeskrivning C

Inga förändringar skedde vad gäller målgruppsanpassning. Vid de användartester som gjordes verkade samtliga testpersoner förstå vad

spelet gick ut på utan att få någon inledande förklaring, vilken tyder på att våran målgruppsanpassning har fungerat bra.

7 Användarscenarier A

7.1 Scenario #1 A

Bob och Alice har tänkt spela en omgång Flongout. Bob tar fram sin laptop och sin USB-handkontroll. De två inser dock att de bara har en handkontroll, och bestämmer sig för att Alice får spela med tangentbordet. Alice startar Flongout och navigerar sig från huvudmenyn till spelets kontrollinställningar. Alice ska vara spelare nummer ett, så hon väljer ”tangentbord” som kontrollmetod för spelare ett. För Bob, spelare två, väljer hon istället ”handkontroll #1”, och ber Bob att trycka på några knappar för att testa handkontrollen. Då Bob testar några knappar blinkar en liten lampa vid spelare två på skärmen för att visa vilken handkontroll som har valts. Alice lämnar sedan kontrollinställningarna och startar spelet.

7.2 Scenario #2 A

Bob och Alice står nu vid spelets startskärm och vid given input av både tangentbord och handkontroll rör sig respektive paddel. Alice frågar Bob om han är redo att starta och Bob nickar.

Alice trycker på startknappen och en nedräkning börjar för att visa när bollen ska skjutas iväg, bollen skjuts iväg i slumpvald riktning, Alice har otur den här gången och bollen kommer mot hennes kortsida. Eftersom det är första gången Alice spelar Flongout så är hon inte beredd på i vilken hastighet paddeln rör sig och missar därför att skydda sin kortsida. Ställningen är nu Bob 1, Alice 0, nedräkningen börjar igen. Denna gång färdas bollen mot Bobs kortsida och eftersom Bob använder handkontrollen så har han mycket lättare att styra sin paddel, det gör han genom att ena analoga styrspaken anger position av paddeln och den andra analoga styrspaken anger paddelns vinkel.

Bob returnerar bollen och den studsar i den nedre långsidan av spelplanen och reflekteras mot Alices kortsida. Även Alice lyckas denna gång returnera bollen i Bobs riktning och vid kontakt med Bobs paddel dyker ett antal brickor upp på spelplanen. Bollen färdas i rikning mot en av de nytillkomna brickorn och vid kontakt med brickan så förstörs brickan och försvinner, bollen returneras till Bob. Bob returnerar bollen ännu en gång och denna gång når bollen Alices kortsida, Alice siktar på en bricka med bokstaven P på och träffar den. Brickan går sönder och ett P ”flyter” mot Alices kortsida, Alice fångar

in det flytande P:et med sin paddel. Alice märker ingen skillnad, men när Bob slår till bollen mot Alices del av spelplanen så märker hon genast vad som hänt.

När bollen når Alices halva av spelplanen så saktar bollen och och färdas med halva hastigheten, Alice returnerar nu lätt varje boll som Bob skjuter mot henne. Bob hinner inte skydda sin kortsida och resultatet är nu 1 - 1, spelet avbryts då det ringer på dörren och Alice måste öppna, så Alice trycker på ESC och hamnar i spelmenyn med pågående spel pausat medans hon öppnar dörren. Spelet kan inte fortsätta vid detta tillfälle då Alice har viktigare saker för sig, så Bob väljer att avsluta spelet via spelmenyn och återgår till huvudmenyn.

8 Användarscenarier B

8.1 Scenario #1 B

Bob och Alice har tänkt spela en omgång Flongout. Bob tar fram sin laptop och sin USB-handkontroll. De två inser dock att de bara har en handkontroll, och bestämmer sig för att Alice får spela med tangentbordet. Alice startar Flongout och navigerar sig från huvudmenyn till spelets kontrollinställningar. Alice ska vara spelare nummer ett, så hon väljer ”tangentbord” som kontrollmetod för spelare ett. För Bob, spelare två, väljer hon istället ”handkontroll #1”, och ber Bob att trycka på några knappar för att testa handkontrollen. ~~Då Bob testar några knappar blinkar en liten lampa vid spelare två på skärmen för att visa vilken handkontroll som har valts.~~ Alice lämnar sedan kontrollinställningarna och startar spelet.

8.2 Scenario #2 B

Bob och Alice står nu vid spelets startskärm och vid given input av både tangentbord och handkontroll rör sig respektive paddel. Alice frågar Bob om han är redo att starta och Bob nickar.

~~Alice trycker på startknappen och en~~ En nedräkning börjar för att visa när bollen ska skjutas iväg, bollen skjuts iväg i slumpvald riktning, Alice har otur den här gången och bollen kommer mot hennes kortsida. Eftersom det är första gången Alice spelar Flongout så är hon inte beredd på i vilken hastighet paddeln rör sig och missar därför att skydda sin kortsida. Ställningen är nu Bob 1, Alice 0, nedräkningen börjar igen. Denna gång färdas bollen mot Bobs kortsida och eftersom Bob använder handkontrollen så har han mycket lättare att styra sin paddel, det gör han genom att ena analoga styrspaken

anger position av paddeln och den andra analoga styrspaken anger paddelns vinkel.

Bob returnerar bollen och den studsar i den nedre långsidan av spelplanen och reflekteras mot Alices kortsida. Även Alice lyckas denna gång returnera bollen i Bobs riktning och vid kontakt med Bobs paddel dyker ett antal brickor upp på spelplanen. Bollen färdas i riknning mot en av de nytillkomna brickorna och vid kontakt med brickan så förstörs brickan och försvinner, bollen returneras till Bob. Bob returnerar bollen ännu en gång och denna gång når bollen Alices kortsida, Alice siktar på en bricka ~~med bokstaven P på~~ och träffar den. Brickan går sönder och ett P ”flyter” mot Alices kortsida, Alice fångar in det flytande P:et med sin paddel *som genast blir nästan dubbelt så stor!* ~~Alice märker ingen skillnad, men när Bob slår till bollen mot Alices del av spelplanen så märker hon genast vad som hänt.~~

~~När bollen når Alices halva av spelplanen så saktar bollen in och och färdas med halva hastigheten.~~ Alice returnerar nu lätt varje boll som Bob skjuter mot henne. Bob hinner inte skydda sin kortsida och resultatet är nu 1 - 1, spelet avbryts då det ringer på dörren och Alice måste öppna, så Alice trycker på ESC och hamnar i spelmenyn med pågående spel pausat medans hon öppnar dörren. Spelet kan inte fortsätta vid detta tillfälle då Alice har viktigare saker för sig, så Bob väljer att avsluta spelet via spelmenyn och återgår till huvudmenyn.

9 Användarscenarier C

Scenariotexterna i B-delen ovan har justerats för att enbart spegla sådant som faktiskt implementerats i spelet.

Vad gäller indikatorlampor för handkontroller på kontrollinställningsskärmen har vi helt enkelt glömt bort att detta fanns planerat. Gällande den power-up som skulle ha sänkt hastigheten på bollen så valdes istället andra varianter som var lättare att implementera. Mer om båda dessa områden följer i sammanfattningen.

10 Testplan A

Förutom unit-testning av programkodens olika delar kommer följande moment testas med riktiga användare:

- Val av kontrollmetoder (se scenario #1)
- Start av spelet (se scenario #2)

- Spelande av en omgång för att sedan återgå till huvudmenyn
- Spelande av flera på varandra följande omgångar för att sedan återgå till huvudmenyn
- Spelande av en eller flera omgångar för att sedan avsluta spelet
- Avbryta en omgång och starta en ny
- Avbryta en omgång och avsluta spelet (se scenario #2)

11 Testplan B

Förutom unit-testning av programkodens olika delar kommer följande moment testas med riktiga användare:

- Val av kontrollmetoder (se scenario #1)
- Start av spelet (se scenario #2)
- Spelande av en omgång för att sedan återgå till huvudmenyn
- Spelande av flera på varandra följande omgångar för att sedan återgå till huvudmenyn
- Spelande av en eller flera omgångar för att sedan avsluta spelet
- Avbryta en omgång och starta en ny
- Avbryta en omgång och avsluta spelet (se scenario #2)

12 Testplan C

Testplanen följdes som planerat. Vissa delar av programkoden var för svåra att unit-testa och saknar därmed sådana test. Mer om dessa unit-test i sammanfattningen.

12.1 Feedback från användartester

Användare #1:

> *Hmm. Ja, det var lite småsvårt på tangentbordet. Just med att slå paddeln hela vägen fram Det hade kanske varit nice om man slog hela vägen fram automatiskt när man tryckte upp/ner. Jo det var min andra feedback nu. Jag kollade inte på menyalternativen iofs. men visa vilka knappar som gör vad.*

Användare #2:

Konstigt när man ställer in gamepaden och man måste röra analogstickorna 2 gånger för att det ska gå in, sen helt plötsligt behöver man inte göra det med knapparna. Om man väljer keyboard så borde man få se vilka knappar som gör vad.

Användare #3:

Hur länge kör man? Slutar spelet aldrig? Konstigt när man väljer kontroll, cpu1 och cpu2. Vad är skillnaden?

Användare #4:

Jag förväntade att man skulle få se vilka knappar som gjorde vad i controller setup men istället så råkade jag ställa om hela keyboardet fel. Skithäftigt med ai, även fast den gör självmål ganska ofta. Varför är quick spin up ok? alltså i menyn. Det står ingenstans.

Användare #5:

Kul spel men väldigt svårt. Jag fattar ingenting av controllersetup. Bra att man kunde ta bort bakgrunden. Borde finnas flera svårighetsgrader på datorn.

12.2 Följder av användartester

Eftersom flera av testpersonerna efterfrågade information om vilka knappar som gör vad lade vi till en extra bild som visas innan spelet börjar. Bilden visar vilka knappar som styr paddeln enligt den förinställda tangentbordskonfigurationen, tillsammans med en text som påpekar att konfigurationen går att ändra i menyn.

Det gjordes även ett försök att förbättra detektionen av axlarna då man konfigurerar en handkontroll. Idén var att införa en exponentialkurva för axlarna, som skulle ge högre detektionsvärden då man rört axeln långt från utgångsläget jämfört med den linjära kurva som användes innan. Tyvärr fungerade detta inte särskilt bra i praktiken och vi gick således tillbaka till den linjära kurvan.

13 Programdesign A

13.1 Ramverk

Programmet kommer att baseras på ramverket Slick2D[1].

[1] http://slick.ninjacave.com/

13.2 Centrala klasser A

13.2.1 `Main` A

Programmets huvudklass `Main` kommer att ärva från Slick2D:s `BasicGame` och kommer att ansvara för att starta och byta mellan programmets olika ”scener” och att vidarebefodra Slick2D:s standardhändelser till dessa.

13.2.2 `Scene` A

De olika ”scener” som finns i programmet (meny, inställningar, själva spelandet) kommer att var för sig skrivas som en egen klass som implementerar interfacet `Scene`.

Exempel på viktiga metoder:

- `void Scene::init()`
- `void Scene::update(Controls[] input)`
- `void Scene::render(Graphics g)`

13.2.3 `MainMenu implements Scene` A

Klass för huvudmenyn.

Exempel på viktiga metoder: se `Scene`

13.2.4 `ConfigurationScreen implements Scene` A

Klass för kontrollinställningsskärmen.

Exempel på viktiga metoder: se `Scene`

13.2.5 `Game implements Scene` A

Klass för själva spelandet.

Exempel på viktiga metoder: se `Scene`

13.2.6 `InGameMenu implements Scene` A

Klass för själva spelandet.

Exempel på viktiga metoder: se `Scene`, samt

- `void InGameMenu::setBackground(Game game)`

13.2.7 `Controls` **A**

Klassen `Controls` kommer att modellera en för spelet idealisk handkontroll. Separata klasser för olika inmatningsmetoder kommer sedan att fylla instanser av `Controls` med lämplig data som sedan kan läsas på ett generellt sätt av spelets olika delar.

Exempel på viktiga metoder:

- `Controls::setLeftStickAngle(float angle)`
- `float Controls::getLeftStickAngle()`
- `boolean Controls::isButtonOneDown()`

13.2.8 `Physics` **A**

Klassen `Physics` kommer att ansvara för de fysikberäkningar som behövs.

Exempel på viktiga metoder:

- `void Physics::resolveCollision(Ball b, Paddle p)`
- `void Physics::resolveCollision(Ball b, Brick br)`
- `void Physics::resolveWallCollision(Ball b)`

13.2.9 `Brick` **A**

Brickorna skapas i denna klass. Det ska finnas möjlighet att välja färg, hårdhet och om brickan har en powerup eller inte.

Exempel på viktiga metoder:

- `float Brick::getHardness()`

13.2.10 `Paddle` **A**

Klassen för ”paddlarna” då både player 1 och player 2 behöver en paddel, samt vissa metoder för att ändra utseende samt attribut av paddeln.

Exempel på viktiga metoder:

- `float Paddle::getAngularMomentum()`

13.2.11 `Ball` **A**

Klassen för bollen.

Exempel på viktiga metoder:

- `Vec2 Ball::getVelocityVector()`
- `void Ball::accelerate(Vec2 v)`

13.2.12 `Powerup` **A**

Interface för powerups.

Exempel på viktiga metoder:

- `Powerup::applyTo(Ball b)`
- `Powerup::applyTo(Paddle p)`
- `Powerup::applyTo(Physics ph)`

14 Programdesign B

14.1 Ramverk

Programmet kommer att baseras på ramverket Slick2D[2].

14.2 Centrala klasser B

14.2.1 `Main` **B**

Programmets huvudklass `Main` kommer att ärva från Slick2D:s `BasicGame` och kommer att ansvara för att starta och byta mellan programmets olika ”scener” och att vidarebefodra Slick2D:s standardhändelser till dessa.

14.2.2 `Scene` **B**

De olika ”scener” som finns i programmet (meny, inställningar, själva spelandet) kommer att var för sig skrivas som en egen klass som implementerar interfacet `Scene`.

Exempel på viktiga metoder:

[2] http://slick.ninjacave.com/

- `void Scene::init`
- `void Scene::update`
- `void Scene::render`

14.2.3 `MainMenuScene implements Scene` **B**

Klass för huvudmenyn.

Exempel på viktiga metoder: se `Scene`

14.2.4 `ControllerSetupScene implements Scene` **B**

Klass för kontrollinställningsskärmen.

Exempel på viktiga metoder: se `Scene`

14.2.5 `GameScene implements Scene` **B**

Klass för själva spelandet.

Exempel på viktiga metoder: se `Scene`

14.2.6 `IngameMenuScene implements Scene` **B**

Klass för själva spelandet.

Exempel på viktiga metoder: se `Scene`, samt

- ~~`void InGameMenu::setBackground(Game game)`~~

14.2.7 `Controller` **B**

Klassen `Controller` kommer att modellera en för spelet idealisk handkontroll. Separata klasser för olika inmatningsmetoder kommer sedan att fylla instanser av `Controller` med lämplig data som sedan kan läsas på ett generellt sätt av spelets olika delar.

Exempel på viktiga metoder:

- ~~`Controls::setLeftStickAngle(float angle)`~~
- ~~`float Controls::getLeftStickAngle()`~~
- ~~`boolean Controls::isButtonOneDown()`~~

- `ctrlObj.leftAnalog.setAngle(double angle)`
- `double ctrlObj.leftAnalog.getAngle()`
- `boolean ctrlObj.buttonOne.isPressed()`

14.2.8 `Physics` **B**

Klassen `Physics` kommer att ansvara för de fysikberäkningar som behövs.

Exempel på viktiga metoder:

- ~~`void Physics::resolveCollision(Ball b, Paddle p)`~~
- ~~`void Physics::resolveCollision(Ball b, Brick br)`~~
- ~~`void Physics::resolveWallCollision(Ball b)`~~
- `void Physics::step(double time, double divisor)`

14.2.9 `Brick` **B**

Brickorna skapas i denna klass. ~~Det ska finnas möjlighet att välja färg, hårdhet och om brickan har en powerup eller inte.~~

Exempel på viktiga metoder:

- ~~`float Brick::getHardness()`~~
- `Brick::setHp(int hp)`
- `int Brick::getHp`

14.2.10 `Paddle` **B**

Klassen för ”paddlarna” då både player 1 och player 2 behöver en paddel, samt vissa metoder för att ändra ~~utseende samt~~ attribut av paddeln.

Exempel på viktiga metoder:

- ~~`float Paddle::getAngularMomentum()`~~
- `float Paddle::getAngularVelocity`

14.2.11 `Ball` **B**

Klassen för bollen.

Exempel på viktiga metoder:

- `Vec2 Ball::getVelocity~~Vector~~()`
- `void Ball::accelerate(Vec2 v)`

14.2.12 `Powerup` **B**

~~Interface~~ Abstrakt basklass för powerups.

Exempel på viktiga metoder:

- ~~`Powerup::applyTo(Ball b)`~~
- ~~`Powerup::applyTo(Paddle p)`~~
- ~~`Powerup::applyTo(Physics ph)`~~
- `Powerup::applyStaticEffects(Paddle collector, Paddle other, GameScene game))`
- `Powerup::removeStaticEffects(Paddle collector, Paddle other, GameScene game))`

15 Programdesign C

Från ett översiktligt perspektiv så följdes designplanen riktigt bra. Naturligtvis har det blivit många små förändringar vad gäller namngivning, parametrar och sammanslagning / uppdelning av metoder.

16 Tekniska frågor A

- Internt koordinatsystem. Skall origo vara i skärmens centrum? Skall y-koordinaten växa eller avta i riktning uppåt från origo?
- Egen fysikmotor eller en existerande? (exempelvis Box2d). Frågan är om det är lättare att bygga en egen motor, eller att sätta sig in i en färdigbyggd. Oftast hanterar inte en färdig fysikmotor enbart kollisioner som är det vi behöver, så i detta fall är det nog lättare att bygga en egen. Problemet med en egen fysikmotor är att få alla småproblem att fungera, det är lätt hänt att bollar ”buggar” sig igenom väggar och studsar fel etc.

- Rita egen vektorgrafik eller använda bilder? Ska vi rita grafiken i spelet programmatiskt eller vektorgrafik av typ SVG eller bara rastergrafik, t.ex PNG? Att rita grafiken programmatiskt är väldigt tidskrävande men det skalar bättre och kvalitén blir bättre. Kan slick hantera vektorgrafik av typ SVG? Är vissa grafiska objekt värda att programmera?
- Nätverksspel eller inte? Nätverksspel hade ju självklart varit roligt, men är tidsramen för detta projekt för kort? Om det finns tid över kanske nätverksspel kan programmeras. Det förhållandevis stora antalet fysikberäkningar som behövs i spelet gör nätverksspel särskilt svårt att implementera på ett bra sätt på kort tid.

17 Tekniska frågor B

- Internt koordinatsystem. Skall origo vara i skärmens centrum? Skall y-koordinaten växa eller avta i riktning uppåt från origo?
- Egen fysikmotor eller en existerande? (exempelvis Box2d). Frågan är om det är lättare att bygga en egen motor, eller att sätta sig in i en färdigbyggd. Oftast hanterar inte en färdig fysikmotor enbart kollisioner som är det vi behöver, så i detta fall är det nog lättare att bygga en egen. Problemet med en egen fysikmotor är att få alla småproblem att fungera, det är lätt hänt att bollar ”buggar” sig igenom väggar och studsar fel etc.
- Rita egen vektorgrafik eller använda bilder? Ska vi rita grafiken i spelet programmatiskt eller vektorgrafik av typ SVG eller bara rastergrafik, t.ex PNG? Att rita grafiken programmatiskt är väldigt tidskrävande men det skalar bättre och kvalitén blir bättre. Kan slick hantera vektorgrafik av typ SVG? Är vissa grafiska objekt värda att programmera?
- Nätverksspel eller inte? Nätverksspel hade ju självklart varit roligt, men är tidsramen för detta projekt för kort? Om det finns tid över kanske nätverksspel kan programmeras. Det förhållandevis stora antalet fysikberäkningar som behövs i spelet gör nätverksspel särskilt svårt att implementera på ett bra sätt på kort tid.

18 Tekniska frågor C

Resultat / svar på tekniska frågor:

- Internt koordinatsystem.
  Ja!
  - Skall origo vara i skärmens centrum?
    Ja!
  - Skall y-koordinaten växa eller avta i riktning uppåt?
    Växa, så att trigonometriska funktioner fungerar som man är van vid dem.
- Egen fysikmotor eller en existerande?
  Egen. Att sätta sig in i Box2D visade sig vara icke-trivialt.
- Rita egen vektorgrafik eller använda bilder?
  Då vi inte lyckades få Slick2D att ladda SVG fick vi nöja oss med PNG-bilder.
- Nätverksspel eller inte?
  Det fanns ingen chans att hinna med detta.

Inga nya tekniska frågor tillkom under utvecklingsarbetet.

19 Arbetsplan A

Planen till den 8/5, eller så fort som möjligt, är att först ha en förenklad prototyp där vi skriver en enklare variant av spelet (Pong) samtidigt som vi jobbar på klasserna som ska vara med i det riktiga spelet (t.ex lägga paddlarna i paddel-klassen). Även vissa fysikelement kommer vara samma i prototypen och i det färdiga spelet, t.ex bollens kollision med långsidorna.

Till veckan därpå, den 15/5 förväntas fysiken till spelet vara klar och nya element såsom brickor och powerups läggs till i prototypen efterhand.

Till den 19/5 är planen att spelet ska vara spelbart och inkludera samtliga planerade inslag (brickor, powerups) för att kunna inleda användartester. Spelet får sedan finslipas efter användarnas input.

Den 22/5 är nya användartester planerade. Planen är att få någon annan grupp i indagruppen att användartesta vårat spel efter eller innan den muntliga lägesrapporten.

Ända fram till den 25/5 pågår finputsning av spelet baserade på andra omgångens användartester och spelet förväntas vara helt klart tills deadline.

En uppdelning av klasser kommer att ske mellan projektgruppens medlemmar.

20 Arbetsplan B

Planen till den 8/5, eller så fort som möjligt, är att först ha en förenklad prototyp där vi skriver en enklare variant av spelet (Pong) samtidigt som vi jobbar på klasserna som ska vara med i det riktiga spelet (t.ex lägga paddlarna i paddel-klassen). Även vissa fysikelement kommer vara samma i prototypen och i det färdiga spelet, t.ex bollens kollision med långsidorna.

Till veckan därpå, den 15/5 förväntas fysiken till spelet vara klar och nya element såsom brickor och powerups läggs till i prototypen efterhand.

Till den 19/5 är planen att spelet ska vara spelbart och inkludera samtliga planerade inslag (brickor, powerups) ~~för att kunna inleda användartester. Spelet får sedan finslipas efter användarnas input.~~

Den ~~22/5~~ 21/5 är användartester planerade. Planen är att ~~få någon annan grupp i indagruppen att användartesta vårat spel efter eller innan den muntliga lägesrapporten.~~ hugga tag i folk i korridorerna på KTH och ~~tvinga~~ låta dem testa spelet. Spelet får sedan finslipas efter användarnas input.

Ända fram till den ~~25/5~~ 22/5 pågår finputsning av spelet baserade på ~~andra omgångens~~ användartesterna och spelet förväntas vara helt klart tills deadline.

~~En uppdelning av klasser kommer att ske mellan projektgruppens medlemmar.~~

21 Arbetsplan C

Arbetsplanen följdes i stort. Användartesterna blev fördröjda, och det blev bara en runda istället för de två som var planerade. Detta beror till viss del på att vi av någon anledning rörde till det med olika felaktiga datum (övningen 15/5 skedde i själva verket 13/5, var den 19/5

kom ifrån minns vi inte och den smått viktiga deadline hade vi av någon anledning skjutit fram tre dagar).

Vi gjorde aldrig någon egentlig uppdelning av klasser sinsemellan. Istället blev det inledningvis en uppdelning i form av att vissa främst arbetade med en förenklad prototyp samtidigt som andra arbetade med de klasser som skulle ingå i den slutgiltiga versionen men som även behövdes till prototypen, exempelvis fysiken. Halvvägs genom utvecklingsarbetet upphörde denna indelning, prototypen övergavs då dess funktion — att driva på utvecklingen av de klasser som behövdes för att bygga det riktiga spelet — var uppfylld, och under resterande tid låg allt fokus på den slutgiltiga versionen.

22 Sammanfattning

22.1 Oimplementerade funktioner

22.1.1 Poäng- eller tidsgräns

Vi implementerade aldrig någon poäng- eller tidsgräns för att automatiskt avsluta en spelomgång. Hade vi valt *en* sådan gräns hade det varit oerhört trivialt att lägga till, men det bästa vore förstås om det erbjöds flera olika valbara gränser, vilket tar klart längre tid att göra samtidigt som effekten på slutprodukten inte blir särskilt stor.

Vid användartesterna så frågades det dock om när spelet tar slut, och vi kommer antagligen att vid tillfälle lägga in ett par olika valbara gränser.

22.1.2 Mus som kontrollmetod

Mycket tid lades på kontrollmetoder, och vi hade planerat att inkludera mus samt kombinationen mus och tangentbord som valbar kontrollmetod. Detta hanns dock aldrig med, men med tanke på det fina system som finns på plats kring valbara kontrollmetoder så vore det dumt att inte lägga till muskontroll vid tillfälle. Det skulle antagligen fungera riktigt bra att flytta paddeln med musen och klicka på höger och vänster musknapp för att snärta till paddeln uppåt och nedåt.

22.1.3 Indikatorlampor för inmatning

Just idén med indikatorlampor för att testa så man valt rätt index på handkontroll hade vi helt enkelt glömt bort. En annan idé som dök upp under arbetet men valdes bort på grund av tidsbrist var att kunna

rita upp en representation av den idealiserade handkontrollen och visa hur användarens inmatningar påverkade denna, tillsammans med textetiketter som visar vad de olika delarna av den idealiserade kontrollen gör i spelet. Detta är en mycket intressant idé och är något vi kommer försöka lägga till i efterhand. En sådan visualisering skulle sedan kunna användas för samma funktion som de tidigare tänkta indikatorlamporna.

22.1.4 Power-up som sänker bollens hastighet

Om vi utökar systemet för powerups med en metod `getDynamicEffects` så blir detta väldigt enkelt att lägga till, och är således något vi planerar att försöka göra i efterhand.

22.1.5 Saknade unit-tests

Vissa delar av programkoden, särskilt de grafiska delarna, är svåra att testa utan att ha tillgång till så kallade ”mock objects”. Vi gjorde en mycket kort undersökning om stödet för sådana hos ramverket JUnit och kom fram till att det verkade krävas tredjepartslösningar, och valde därför att skjuta det på framtiden. Det vore dock mycket lärorikt att titta på hur man brukar göra med ”mock objects” i JUnit och är någonting vi planerar försöka göra framöver.

22.2 Vidareutveckling av spelet

Förutom de inslag som nämnts i avsnittet ovan finns det i dagsläget inga konkreta planer på att vidareutveckla spelet.

22.3 Har vi lärt oss något?

22.3.1 Mikael Forsberg

Projektarbetet har varit lärorikt på flera olika sätt. Att skriva en vettigt fungerande fysikmotor är något av en utmaning även för luttrade programmerare. Virtuella bollar har en tendens att hamna inuti väggar och ramla genom golv, och det är inte sällan lösningen till en kollision genast leder till en ny.

Att samarbeta intensivt med Git har varit lärorikt och intressant. Jag har sedan tidigare en del erfarenhet av Git, men har aldrig samarbetat till den grad att så kallade ”branches” blir användbara.

Att använda en stor IDE, Eclipse, har även det varit lärorikt. Dock kanske inte främst på det sätt som kursledaren hade tänkt. Visst är det effektivt att kunna byta namn på klasser och variabler med ett

högerklick och några tangenttryckningar, men priset man får betala verkar vara ganska högt. Eclipse är nämligen så pass komplext att det behövs många olika konfigurationsfiler och stora mängder cachad metadata för att hålla reda på alltsammans. Ibland verkar dessa filer bli korrupta. Sådana tillfällen verkar i regel leda till att Eclipse helt enkelt inte startar längre, tills dess man raderat samtliga konfigurationsfiler och all metadata. Att man sedan får göra om alla sina personliga inställningar och på nytt importera sina projektfiler för att börja jobba igen är inte direkt någon effektivitetshöjande företeelse.

22.3.2 Robin Gunning

Uppskatta fin och effektiv kod är nåt jag lärt mig, och det får mig att se på min egen kod på ett annat sätt. Jag är inte längre nöjd med att bara slänga ihop nånting som funkar, det ska även vara elegant. Att projektet skulle kännas mer som ett ”riktigt jobb” har varit bra, och saker har inte blivit riktigt som jag tänkt mig då jag enbart skrivit en massa småprogram för övningarna innan.

Projektet har gjort det lättare för mig att sätta mig in i nån annans kod och snabbt kunna utgöra vad given kod gör.

Jag trodde att man skulle behöva diskutera varenda liten detalj av programmet med de andra för att komma fram till hur man ville ha det, men det behövs inte. Gillar jag inte en del av programmet så får jag programmera dit ett val så att man kan stänga av det (bakgrunden). Det verkar som om det för det mesta löser sig genom koden och man behöver inte diskutera varenda liten detalj.

22.3.3 Jonathan Håkansson

Jag lärt mig att man kan bygga upp ett spel på många olika klasser. T.e.x att man hade counter som en sorts räknare och power kunde inneha många flera olika sorters klasser. La till en extra powerup som gjorde att bollen gick snabbare.