**Slutrapport – Inda11**

**Del A – Projektplanen**

**Programbeskrivning**.   
Programmet är ett spel som är känt som Pong. Det är ett spel för två personer med var sin bräda/paddel som försöker studsa tillbaka en boll tillbaka mot motståndaren. Missar man bollen så att den passerar brädan får motståndaren en poäng. Konceptet är nog välkänt för de flesta. I grundutförandet tänkte vi oss ett spel för två personer eller en person mot datorn (väljs med meny) med en enkel svart bakgrund, vita väggar ovan och nedan, vit streckad mittlinje, vita paddlar, vit boll och vita siffror som visar poängställningen. Någonstans på spelskärmen kommer det att stå ”press ’h’ for help”. Vi kommer köra med fixed resolution (800x600), åtminstone till att börja med.

Om tid finns så har vi lite features att lägga till, t ex:

* High-scores
* Svårighetsgrad på AI
* Teman (annan bakgrund, boll, brädor)
* Power-ups
  + Snabbare boll för ena sidan
  + Vapen (laserkanoner)
  + Större/mindre bräda
  + Dubbla bollar
* Osv.

**Användarbeskrivning**.   
Vi antar att vem som helst kommer att använda programmet, i princip måste användaren vara läskunnig och förstå engelska, alternativt att en läskunnig engelskspråkig person förklarar för användaren.

**Användarscenarier**.   
Scenario 1: Två personer ska spela mot varandra. Då spelet startas möts de av en meny där de får välja ’1 player’ eller ’2 player’. De väljer mellan alternativen med piltangent ’upp’ resp. ’ner’ samt väljer med ’enter’. Spelet börjar med att bollen är låst på endera spelares bräda (väljs slumpmässigt). Är det spelaren till vänster som börjar serva så väljer han/hon position på brädan med ’w’ (uppåt) eller ’s’ (nedåt), därefter servar man med ’d’. Motsvarande gäller för högra spelaren, men knapparna i samma ordning är ’upp’, ’ned’ och ’vänster’. När någon har servat så rör sig bollen rakt över planen mot den andra spelaren som med knapparna som nämnts rör sig upp eller ned för att försöka få bollen att studsa mot brädan så att den studsar tillbaka. Om någon missar bollen så att den passerar brädan så får den spelare som inte missat bollen en poäng – som uppdateras på skärmen. I samma skede får den som missade bollen serva precis som när spelet startade. Och så fortsätter spelet tills man tröttnat.

Scenraio 2: Ensam person spelar mot datorn. Funkar på samma sätt med menyn. Datorn är till vänster och personen till höger (med samma knappar). Vem som börjar är slumpmässigt. Samma gäller som ovan.

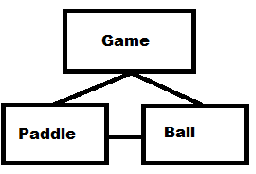
**Testplan**.   
Testklasser till metoder som inte har med GUIn att göra – i den mån det är meningsfullt och går. Annars är det användartestning som gäller. Användartestare får prova att spela mot varann eller mot datorn – eventuellt testa menyalternativ osv.

**Programdesign**.  
Vi tänkte använda Slick-2d som spelmotor. Därmed måste vi implementera dess Game interface. Vet inte om main(String[] args) hamnar i en egen klass med lite annat (meny-val och sånt) eller om den hamnar i game. Spel-loopen körs i Game genom GameContainer.

**Klass Game:** Overrides: init(), render(), update()  
init(): skapa bakgrundsbild, boll-bild, paddle-bilder, kant-bilder, skapa Ball-objekt, skapa Paddle-objekt.  
render(): ritar bilden utifrån data som finns som instansvariabler, förutom positionen av Ball och Paddle som hämtas genom metoder.  
update(): Kollar användar-input, ser till så att bollen rör sig samt paddlarna vid behov – uppdaterar alla instansvariabler med dessa metoder.

**Klass Ball:** collisionCheck(), getNewXPosition(), getNewYPosition()  
collisionCheck(): kolla kollision med kanter eller paddlar samt om bollen passerat paddlar (dvs poäng och reset-to-serve). Returnera boolean?  
getNewXPosition(), getNewYPosition() – säger sig självt.

**Klass Paddle:** getNewYPosition()



**Tekniska frågor**. Angående formen för projektet så tänkte vi använda github. Den underlättar source-control och bug-databas. Vi kollar kontinuerligt på egna maskiner att ändringar funkar. Om det inte funkar mellan maskiner så finns alltid telefon och mail. Vi ser till att dokumentera grundligt på en gång.

Angående själva det praktiska programmerandet så löser Slick en hel del åt oss verkar det ju som. Att objekt kan finnas utanför den synliga skärmen för att ta ett exempel. Att lösa:

* Kollisionskontroll, storlek på paddlar i förhållande till storlek på boll.
* Studsa snett beroende på var på paddeln bollen studsar.
* Hantera situationer där bollen endast studsar i y-led.
* Kalibrera maximala hastigheten på boll, paddel och AI.
* Fixt- eller flera skärmupplösningar?
* Skapa AI

**Arbetsplan**. I princip är formen att designa tillsammans ner till metodstubbs-nivå. Därefter programmera på egen hand. Vi kommer regelbundet reviewa varandras kod och ge feedback. Ingen riktigt klar tidsplan, försöka få en fungerande prototyp så fort som möjligt – helst vara igång innan det här lämnades in. Därefter är det bara att putsa och lägga till features. Senast på måndag den 23/4 räknar vi med att ha en spelklar grundversion av Pong.

**Del B – Uppdaterad plan**

**Programbeskrivning.**Grundutförandet och spelkonceptet är öförändrad. Spelet startar med en meny där användaren kan välja att spela mot en annan, spela mot datorn, köra ett demo-läge (dator mot dator) eller avsluta spelet. Ytterligare en meny startas när användaren väljer att spela mot datorn eller demo-läget, där datorns svårighetsgrad kan väljas. Alla tre spelalternativen leder till samma bana. Under spelets gång kan man använda ’H’-knappen som pausar spelet och visar spelaren de nödvändiga knappar samt förklaringar av alla power-ups.  
  
**Användarbeskrivning.**Inget att kommentera sen projektplan.  
**Användarscenarier.**Det enda som skiljer sig från projektplanen är att vi har valt att styra menyer med muspekaren istället för att bläddra med piltangenter (Slick-biblioteket har enkla metoder för att hantera musknappar och musrörelse).   
  
**Testplan.**Inga testklasser. Hade nog varit praktiskt vid konstruktion av AI samt för konstruktion av kollision med paddlarna, men svårt att sätta upp scenario i och med Slick?  
  
De som testat spelet är Eriks flickvän som har normal datorerfarenhet och Daniels bror som är mer erfaren. Feedback vi fått var att det inte framgick vilken sida man styrde när man kör mot datorn. Också att det inte framgick hur länge spelet pågår eller vad man ska göra när man tröttnat (med escape kommer man tillbaka till menyn men det framgår inte någonstans). Är en aning förvirrande när FPS visas i vänstra övre hörnet. Vore positivt om man fick välja vilken paddel man ska spela med. Annars var farten helt ok och spelkänslan bra.

Merparten av detta borde gå att åtgärda utan alltför mycket arbete. **Programdesign.**Det har skett många förändringar i programmets design, vilket var förväntat. Slick-biblioteket erbjuder en s.k Stateklass som förenklar uppdelningen av spelets tillstånd. Varje Stateklass (se figur nedan) har egna init, update respektive render metoder och alla initieras i PongGameklassen.

Klass PongGame  
Initierar och förbereder alla stateklasser med deras tillhörande stateID genom metoderna initStateList() och addState().

Klass MainMenuState  
Visar bilder och knappar genom init() och render() metoder, knapparna har även en buttonVisualEffect() metod som anropas när muspekaren är på en knapp. Update() metoden väntar på användarens input för att kunna gå över till DifficultyMenuState eller GamePlayState.

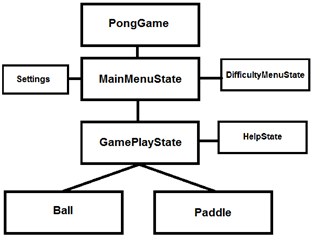
Klass Settings  
Innehåller variabler för att styra spelet, om paddlarna ska vara styrda av tangenter eller dator, svårighetsgrad, osv. Dessutom finns info om fönstrets storlek. Alla dessa variabler har publika och statiska setters och getters.

Klass DifficultyMenuState  
Samma metoder som MainMenuState ovan, annan bakgrund och andra knappar. Update() metoden väntar på att användaren väljer svårighetsgrad.

Klass GamePlayState  
Denna klass initierar två paddelobjekt och ett ballobjekt. Klassen har även koll på poängen och uppdaterar dem. Genom boolean variabler samt en String för svårighetsgrad så anropas olika AI-metoder för att styra paddeln, då paddeln inte är styrd av en användare så klart. Genom dessa kommer motsvarande getAIEasy() metod anropas för att uppdatera datorns paddel. Tanken är att easyAI ska röra på sig slumpvist med jämna intervall, mediumAI ska följa efter bollen och hardAI ska beräkna vilka koordinater bollen kommer hamna på.

Klass HelpState  
Klassen är tillgänglig under spelets gång, startas med ’H’ knappen och visar vilka knappar som paddlar styrs med samt en förklaring av power-ups.

Klass Ball  
Innehåller bollens koordinater och hastighet för varje frame. Finns metoder som anropas från GamePlayState som gör att bollen kollar krockar med väggar eller paddel, beräknar ny hastighet när det är nödvändigt, förflyttar sig, osv. I kollisionskollen finns kodat så att beroende på var på paddeln som bollen träffar så får bollen en bestämd riktning, från 30 grader till 150 grader. Hanterar även servar.  
  
Klass Paddle  
Innehåller bara info om var paddeln befinner sig samt metoder för att flytta på den.

**  
  
Tekniska frågor.**Github har funkat hittills, däremot känner vi inte till alla funktioner så vi undviker att programmera på samma klasser samtidigt. Vi har inte dokumenterat som planerat då vi programmerar bredvid varandra, men ser till att kommentera innan inlämning av källkod.   
  
Kollisionskontroll orsakade inte så mycket problem. Att få bollen känslig för var på paddeln den träffade var lite besvärligt pga hur koordinater anges (övre vänstra hörnet så att säga) – men det gick bra. I och med ovan lösning så kan inte bollen studsa mer än 30 grader-150 grader. Kalibrering gjordes med hjälp av användartestningen, det enda att ta hänsyn till var att Eriks dator är så pass slö så fick köra ganska låg FPS och lite högre speed på boll och paddel. Vi kör än så länge på fixerad upplösning (800x600). Värst av allt var hardAI som inte är helt buggfri (men nästan!). I ball klassen så räknas hastigheten med decimaler. När man då ska räkna ut var bollen hamnar så blir det trassel med avrundning. Tog ett tag att reda ut med studsar och allt. Å andra sidan kanske inte hard ska vara omöjlig. Lätt är nog lite för lätt, nästan som att spela ensam.

**Arbetsplan**Angående våra planerade features så arbetar vi med olika svårighetsgrader på AI och power-ups. Vi har inte prioriterat high-scores eller teman eftersom vi inte ser dessa som svåra implementationer. Vi har i princip haft olika ansvarsområden, t ex menyer/GUI, paddel, ball, AI osv.

**Del C - Jämförelse**

Det enda som i princip förändrats är hur spelet ser ut, men det var ju också väntat. Vi visste för lite om Slick för att veta vad som behövdes i mer detalj (som game-states), dock lärde vi ju oss det efter projektplanen. Annars har den som sagt inte ändrats så mycket.

**Sammanfattning**Att skriva till synes enkla spel är mer arbete än man tror! Nyttigt att göra något från scratch, tänka igenom vad som behövs och hur det ska implementeras, lägga upp planering och arbeta mot deadlines. Trevligt att se på ett konkret sätt resultatet av all kod man skriver – annorlunda mot att testa en LinkedList t ex. Intressant upplevelse att skriva ett program tillsammans med någon annan, det har vi inte gjort i år. Och det gick riktigt bra tycker vi båda. Slutligen så känns det som att Atari tog Pong ungefär så långt det gick redan 1976, så räkna inte med att hitta spelet i en affär nära dig.