**Slutrapport – Inda11**

**Projektplan – Inda11**

**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.

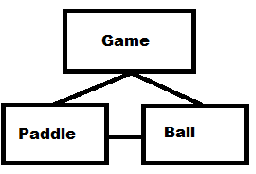
**Testplan**.   
Testklasser till metoder som inte har med GUIn att göra – i den mån det är meningsfullt och går. Annars är det användartestning som gäller. Användartestare får prova att spela mot varann eller mot datorn – eventuellt testa menyalternativ osv.

**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()



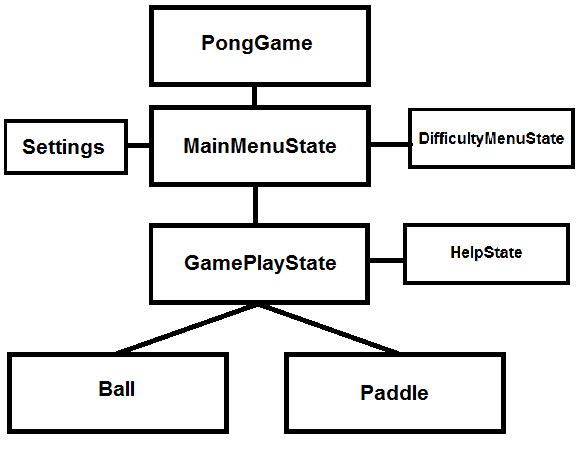
**Tekniska frågor**. Angående formen för projektet så tänkte vi använda github. Den underlättar source-control och bug-databas. Vi kollar kontinuerligt på egna maskiner att ändringar funkar. Om det inte funkar mellan maskiner så finns alltid telefon och mail. Vi ser till att dokumentera grundligt på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

DELB)))

**Programbeskrivning.**Grundutförandet och spelkonceptet är öförändrad. Spelet startar med en meny där användaren kan välja att spela mot en annan, spela mot datorn eller avsluta spelet.  
Ytterligare en meny startas när användaren väljer att spela mot datorn, där datorns svårighetsgrad kan väljas. Båda 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.**  
**Användarscenarier.**Scenarie 1 och 2 har skiljer sig endast i ett par detaljer. 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). Sista detaljen är spelarens och datorns position i spelbanan. Spelaren kan ändra position och andra inställningar vid startmenyn.  
  
**Testplan.  
  
Programdesign.  
**

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.