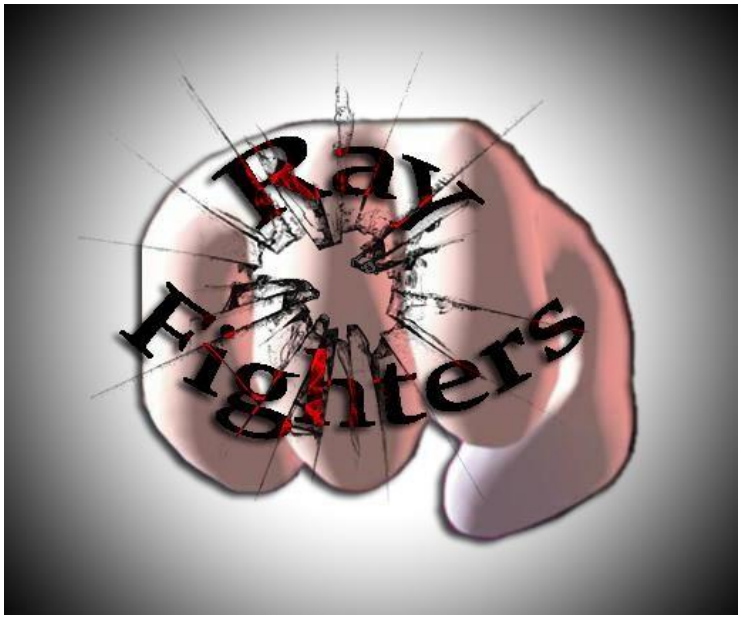
Designspecifikation

Ray fighters



Projektmedlemmar: Albin Sjöström (albsj371)
Anders Vrethem (andvr916)
Christoffer Johansson (chrjo185)
Tobias Mellberg (tobme208)

Handledare: Malin Aspelin

Innehåll

1. Inledning

Det här är ett dokument som beskriver designen för systemet, det vill säga hur modulerna för systemet är upplagt och hur gränssnittet i spelet ska se ut. Ray fighters är ett fightingspel för två eller fler personer som ska framställas av projektgruppen c4 för kursen TDDI02. En match i spelet är en fight mellan två karaktärer, där karaktärerna ska slå eller sparka för att ta ner sin motståndares hälsa till noll. När en karaktär når noll hälsa förlorar den sin runda, en match består av bäst av tre ronder. De karaktärer som spelas kommer vara valbara innan matchen startas. Flera matcher kan spelas i turneringsläget, där matcher delas upp i ett turneringssystem. Mer specifik info angående spelet finns beskrivet i kravspecifikationen.

2. Systemarkitektur

Systemets flöde som visas i figur 1 är indelat i tre olika lägen (*states*) som det kan befinna sig i där varje läge representerar ett delsystem. Dessa delsystem är meny, turnering och spelläge. Vid start ska spelet visa menyn och ska därmed vara i menyläget, från detta läget kan spelet antingen hamna i turneringsläget eller avslutas. I turneringsläget ska varje spelare välja en karaktär innan en ny match skapas. När en ny match skapas ska systemet byta till spelläget När en match är slut ska systemet byta tillbaka till turneringsläget och visa statusen för turneringens matcher, därefter ska nästa match startas. När alla matcher i turneringsläget är slut ska systemet återgå till menyläget.

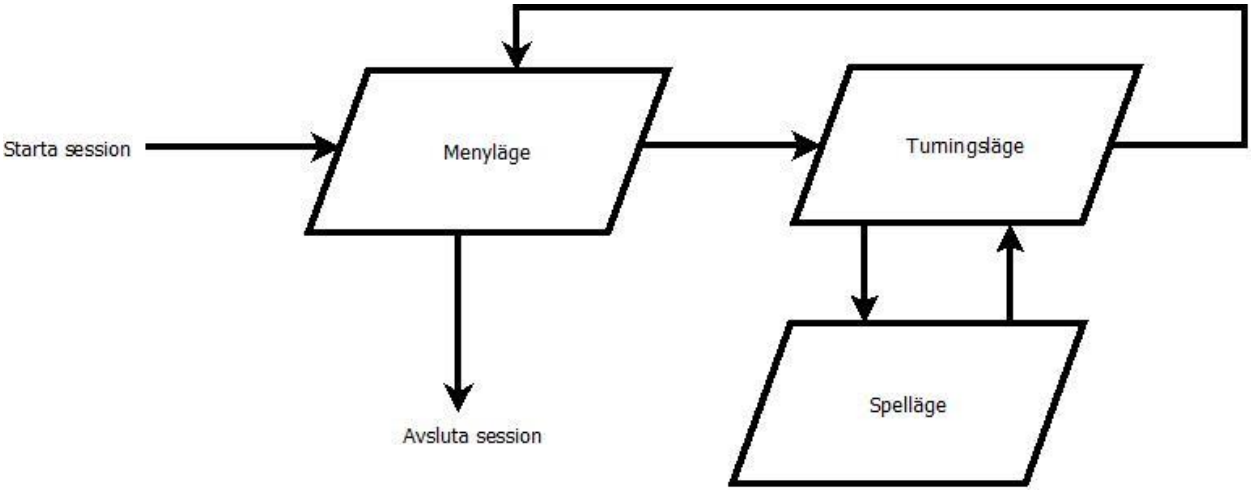


Figure 1 - *Flödesdiagram över systemet*

2.1. Menysystemet

Menysystemet har till uppgift att hantera inställningar som en användare skall kunna ändra i menyn samt att skapa en turnering. Systemet ska visa vilket alternativ i huvudmenyn som är markerat, huvudmenyn ska bestå av alternativen *quit*, *settings, tournament* och *vs-mode*. Menysystemet ska ansvara för input från tangentbordet samt vilka tangenter spelet ska använda. Alternativet *settings* ska hantera olika inställningar som användaren kan välja.

2.2. Turnerings-systemet

Systemet ska skapa en turnering efter parametrar som säger hur många spelare som ska delta. Dessa parametrar ska skickas med från menyläget. Systemet ska låta användaren välja en karaktär åt gången till varje spelare som finns i turneringen, denna karaktär ska sedan vara bunden till spelaren. Valet av karaktär ska ske i en karaktärs meny som ska hanteras inom turneringssystemet. Systemet ska hantera ett schema som visar matcherna mellan spelarna i turneringen enligt figur 2. Systemet ska avgöra om det finns en vinnare i turneringen och om det inte finns det ska nästa match i turneringen startas.

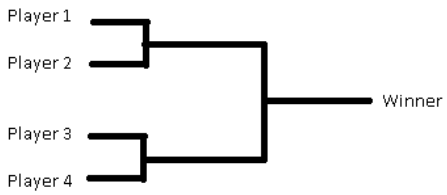


Figure 2 - Ett exempel på hur turneringssystemet för fyra spelare ser ut

2.3. Spelläge-systemet

Systemet ska hantera en match mellan två spelare bestående av 2 eller 3 ronder. Systemet ska bestämma när en rond är slut och hur den avgörs, hur många ronder som ska spelas, poängställningen som visar hur många ronder som respektive spelare har vunnit och vilken spelare som vinner matchen.

3. Teknisk specifikation

Ray fighters ska programmeras objektorienterat med ett antal genomtänkta klasser för varje delsystem. Här presenteras först en översikt av klasserna i varje delsystem som sedan kommer att beskrivas i mer detalj.

3.1. Klassöversikt

De klasser som ska användas samt deras relation till varandra visas i figur 3. Klasserna i Ray fighters kategoriseras efter vilket delsystem de tillhör. Varje klass i varje delsystem har ansvar för olika områden i programmet men det finns en huvudklass inom varje delsystem som har ett övergripande ansvar för de resterande klasserna inom delsystemet, dessa är *Menu, Tournament_mode* och *Match*. Gemensamt för huvudklasser är att de ska ha en funktion *update()* som ska hantera att logiken uppdateras samt att grafiken för det delsystem som det representerar ska ritas ut, när det beskrivs i texten att ett delsystem uppdateras är det ett anrop till denna funktion som avses. *Menu* klassen som är överst i hierarkin ska kontrollera i vilket läge som spelet befinner och göra ett anrop till rätt delsystem så att det kan uppdateras. *Tournament_mode* ska kunna bestå av exakt en *match* objekt eller inget om spelet inte är i spelläget. Klassen ska bestå av minst två *Player* objekt som representerar en spelare per objekt. En spelare ska bestå av exakt ett *Character* objekt som i sin tur ska bestå av ett antal *Bodypart* objekt.

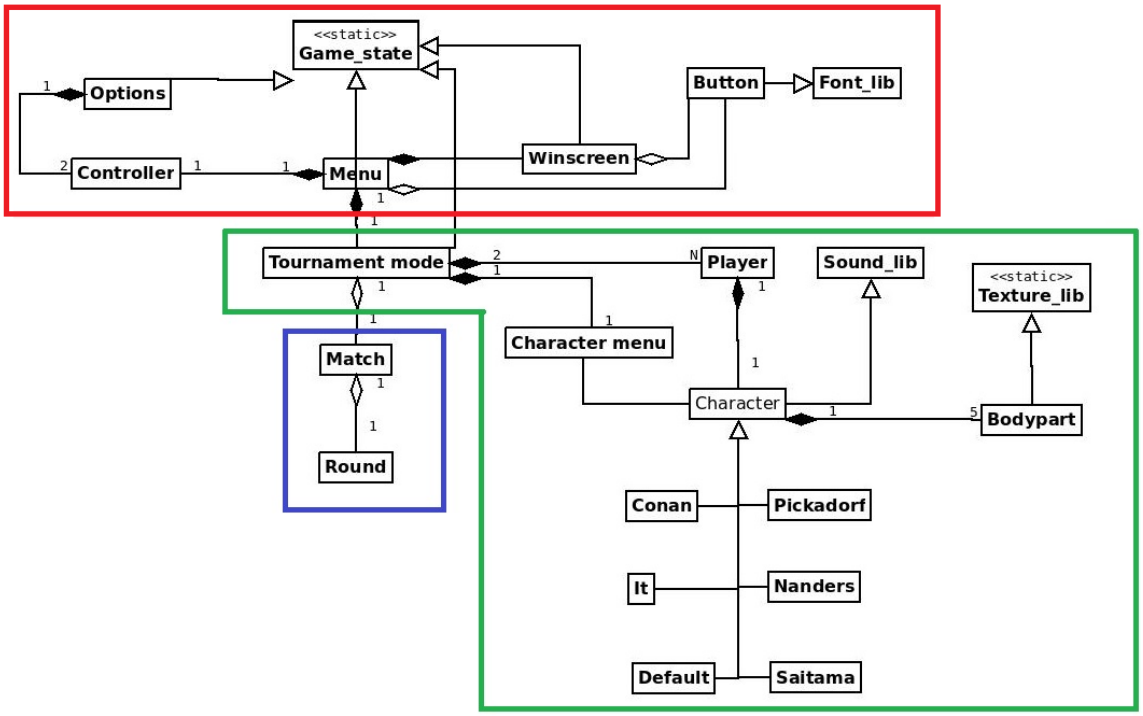


Figure 3 - Översiktligt klassdiagram över systemet och dess delsystem. Röd linje markerar Menysystemet, grön linje markerar Turneringssystemet och blå linje markerar spelläges-systemet

3.2. Menysystemets klasser

Menysystemet består av *options* klassen som ärver *Menu* klassen eftersom många attribut från *Menu* klassen behöver ändras från *options*. Det finns två stycken objekt av klassen *Controller*, varje objekt ska hantera om input från tangentbordet matchar just det objektets bestämda tangenter. Menysystemet håller reda på i vilket läge spelet befinner sig i och bestämmer när det ska byta läge.

3.2.1. Game state

Game state klassen är den översta klassen i systemets klasshierarki. Det betyder att den ska ha ansvaret för vad som ska uppdateras nedåt i hierarkin. Logiken bestäms med hjälp av en variabel som visar i vilket läge spelet är i, om läget är i *options* eller *tournament_mode* ska dessa klasser uppdateras. Om spelet däremot befinner sig i menyn ska ett nytt läge kunna startas.

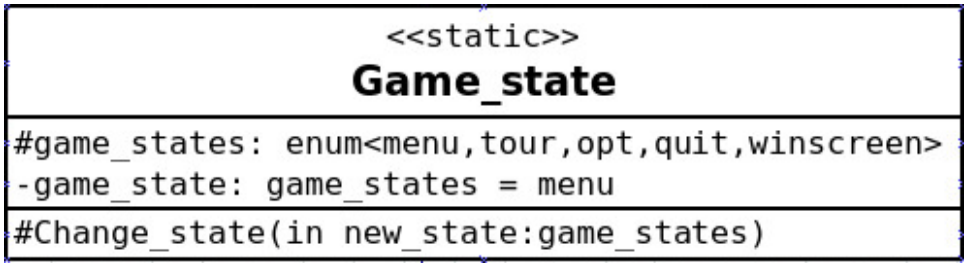


Figure 4 - Game state

enum variabeln *game_states* berättar i vilket läge spelet är i. *change_state* funktionen byter läge.

3.2.2. Menu

Hanterar huvudmenyn och undermenyer.

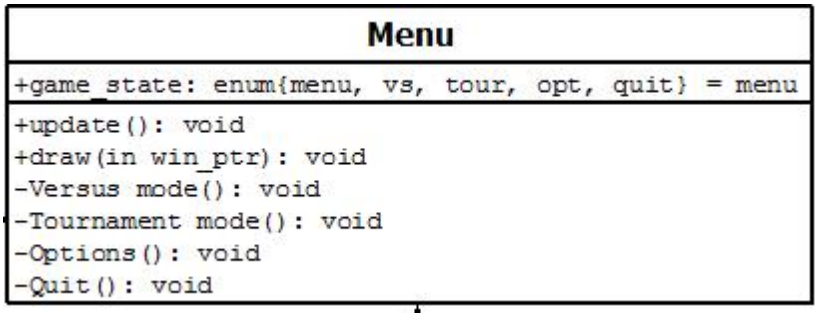


Figure 5 - Innehållet av klassen Menu.

enum variabeln *game_states* berättar i vilket läge spelet är i*, update* funktionen använder den för att bestämma om en subklass ska uppdateras. *update()* uppdaterar logiken och *draw()* ritar ut grafik. Funktionen *Versus_mode()* ska skapa ett nytt *tournament_mode* objekt med förprogrammerade parametrar samt sätta *enum* till **tour**. tournament() presentera en meny där användaren ska välja antalet spelare, därefter ska ett nytt *tournament_mode* objekt skapas med det valda antalet samt *enum* sättas till **tour**. options() ska sätta *enum* till **opt**. quit() ska avsluta spelet.

3.2.3. Options klassen

Options klassen ska visa gränssnittet för inställningsmenyn där användaren ska kunna ändra kontroller samt ljudnivå. Klassen ska innehålla två instanser av *Controller* som lagrar kontrollerna för spelarna.

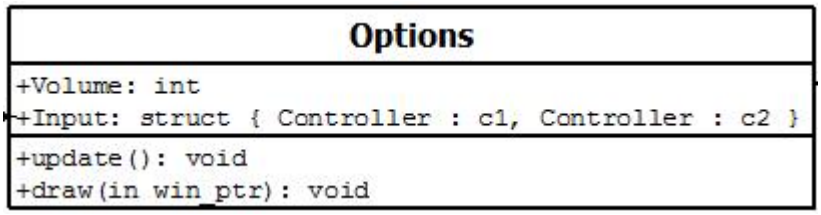


Figure 6 - Innehållet av klassen Options.

Klassen ska innehålla en *struct* som ska bestå av två *Controller* objekt. *Update()* ska hantera om användaren vill ändra kontroller, ändring av en *keybind* ska ske genom anrop till *Controller* objektets funktioner. *Draw()* ska rita ut gränssnittet för undermenyn.

3.2.4. Controller klassen

Controller ska innehålla flertalet styrvariablar, ett för varje sätt som det ska gå att styra en spelare. En styrvariabel ska endera vara sann eller falsk. Tangenter för vilka systemet ska lyssna till för att ändra styrvariablar ska finns i klassen. *Controller* ska kunna läsas av för att se vilka styrvariabler som är sanna.

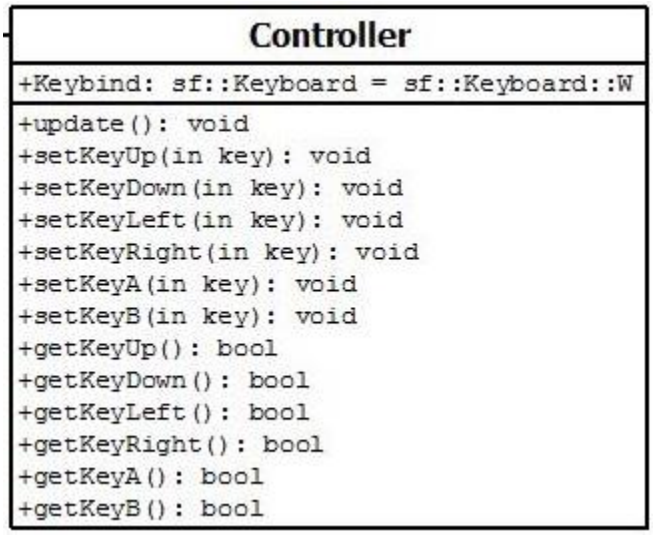


Figure 7 - Innehållet av klassen Controller.

Set funktionerna ska ändra en *keybind* med parametern som skickas med. *Get* funktionerna ska returnera sant eller falskt beroende på om den förvalda tangenten blivit nedtryckt. *Update()* uppdaterar styrvariabler i klassen.

3.2.5. Button klassen

Representerar en klick-knapp i användargränssnittet.

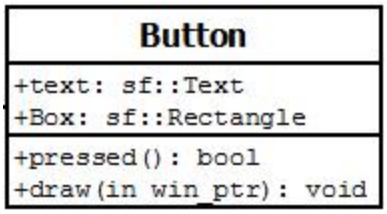


Figure 8 – Publika innehållet av klassen Button

Variabeln *Box* är en ruta som berättar hur stor knappen ska vara och var den befinner sig på skärmen. *text* innehåller den text som ska visas i rutan.

3.2.6. Font_lib

Hanterar fonten för texter.

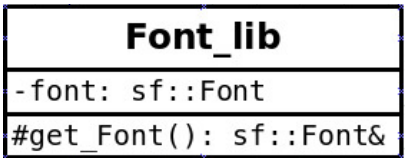
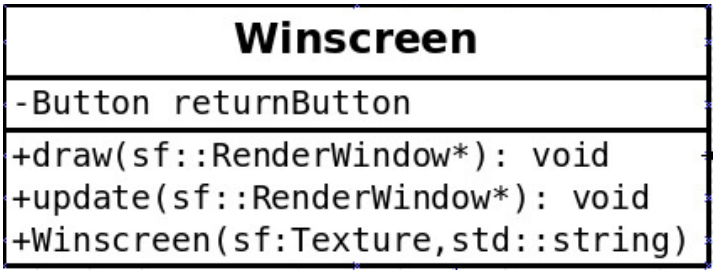


Figure 9 - innehållet av klassen Font_lib

3.2.7. Winscreen

Winscreen visas när en spelare vinner en vs-modematch eller en turnering. När klassen körs skrivs "winner" och vinnarens karaktärs namn ut med vinnarens spelares textur. Därefter skrivs skaparnas namn ut. När som helst kan spelaren återvända till startskärmen med knappen **returnButton.**



Figur 10 – innehållet av Winscreen

Winscreen ska innehålla en Button som kan göra så spelaren kan återvända till startskärmen samt en metod update() som ritar ut namen som nämns ovan. Winscreen har även visar dessa namn grafiskt. Winscreens konstruktor tar emot en sf::Texture för att bestämma texturen som ska ritas ut samt en std::string för namnet på karaktären som vunnit.

3.3. Turneringssystemets klasser

Turneringssystemet är den grupp klasser som ansvarar för antal spelare som ska vara med i en turnering och vilka karaktärer som varje spelare har valt. Delsystemet ansvarar för vilka spelare som ska mötas i en match och vilka som redan har blivit utslagna, detta ska visas grafiskt med ett turneringsschema som finns beskrivet under användargränssnitt.

3.3.1. Tournament mode

Klassen ska se till så att varje spelar objekt har en vald karaktär vid konstruktionen, det ska klassen utföra med hjälp av *character_meny*. Ansvarar för när en ny match ska startas och vilka spelare som ska delta. När klassen uppdateras ska den kontrollera om det finns en pågående match och i så fall uppdatera denna match.

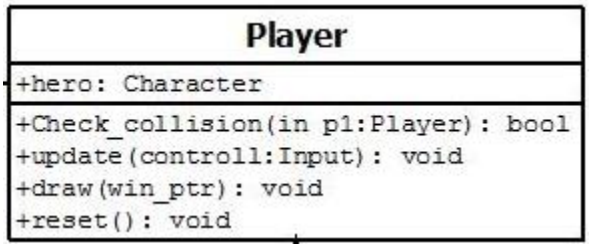
Figur 11 - Innehållet av klassen Tournament mode.

Tournament_mode ska innehålla en lista med alla spelare. *Update()* ska ta emot *stucten Input* som hanterar kontrollerna. *Draw()* ska rita ut grafik. *Done()* ska returnera falskt så länge turneringen pågår. *Player_initialization()* funktionen ska anropas när ett nytt *Tournament_mode* objekt skapas, funktionen ska initiera en karaktär till alla *Player* objekt.

3.3.2. Player

Player klassen ska innehålla och styra en karaktär. Objektet kommer tilldelas en *Controller* i funktionen *update*, funktionen ska hantera styrningen av karaktären med hjälp av *Controller.* Klassen ska hantera

kollision mellan två spelare med funktionen *Check_collision* samt återställa karaktärens hälsa med funktionen *reset*.

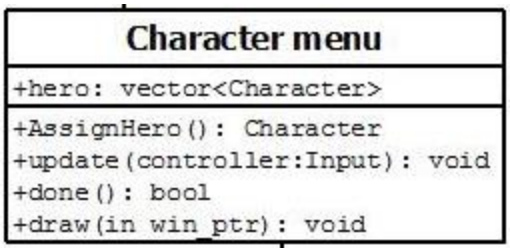


Figur 12 - Innehållet av klassen Player.

Update() ska ta emot kontroller genom *structen Input*, funktionen ska uppdatera en spelares karaktär, anrop till funktionen sker endast om spelaren är i en pågående runda i en match. *draw()* ska rita ut en spelarens karaktär och gör det med *win_ptr* pekaren. *reset()* ska återställa hälsomätaren i spelarens karaktär.

3.3.3. Character menu

Character_menu ska visa ett gränssnitt där en karaktär ska väljas av användaren. Den markerade karaktären ska lysas upp. När en användare väljer en karaktär ska klassen returnera vald karaktär till anropet. *Character_menu* ska innehålla en vektor med alla karaktärer.

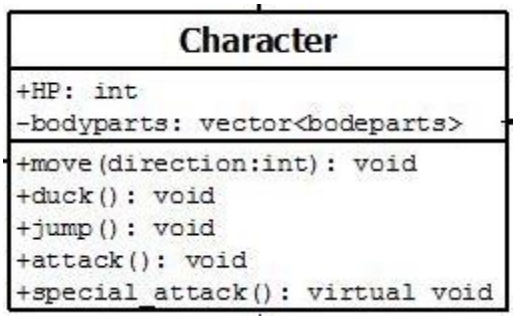


Figur 13- Innehållet av klassen Character menu.

hero ska vara en *vector* med alla *Character* objekt. Funktionen *draw(in win_ptr)* i *Character_menu* ska rita ut miniatyrer av innehållet i *hero* till ett rutsystem för varje valbar figur och visa vilken karaktär som är markerad i gränsnittet, se figur 6. I klassen finns funktionen *AssignHero()* som returnerar den valda karaktären till den aktuella spelaren.

3.3.4. Character

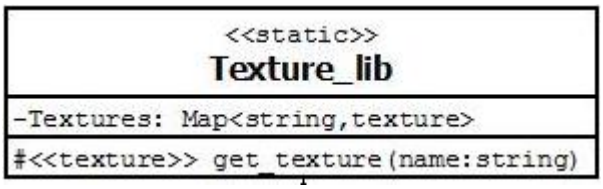
Character ska vara en basklass som varje unik karaktär ärver av. Basklassen ska innehålla hälsa och fyra *bodyparts*. Character ska fixa förflyttning, attacker och duck med följande funktioner: *move(), duck(), jump()* och *attack().* Klassen ska innehålla en virtuell funktion *special_attack()* som hanterar special attacker.



Figur 14 - Innehållet av klassen Character.

3.3.5. Texture_lib

Hanterar texturer.

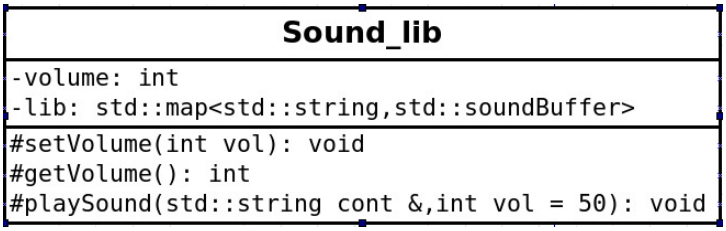


Figur 15 - Innehållet av Texture_lib klassen

Innehåller en map med alla texturer som ska finnas i spelet. En textur hämtas med funktionen *get_texture*.

3.3.6. Sound_lib

Sound lib hanterar ljud

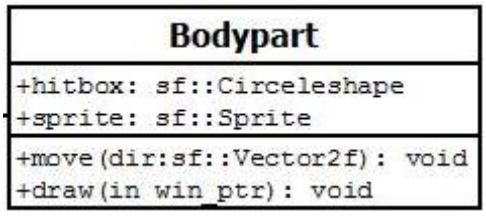


Figur 16 - Innehållet av klassen Sound_lib

Sound_lib innehåller en privat volume för att ta emot ändringen som sker i Options. Sound_lib innehåller även en std::map som används som behållare för de ljud som laddats in i klassen. Med funktionerna setVolume kan volymen ändras, getVolume hämtar volymen och playSound spelar upp det valda ljudet.

3.3.7. Bodyparts

Klassen *Bodypart* hanterar en kroppsdel till en karaktär. Kroppsdelen ska bestå av en *sprite* och en *circleshape* som ska användas för bilden respektive kontrollera kollision. Funktionen *move()* ska användas för att röra hitboxen och spriten åt ett håll givet av inparametern. Funktionen *draw()* ritar ut spriten i klassen.



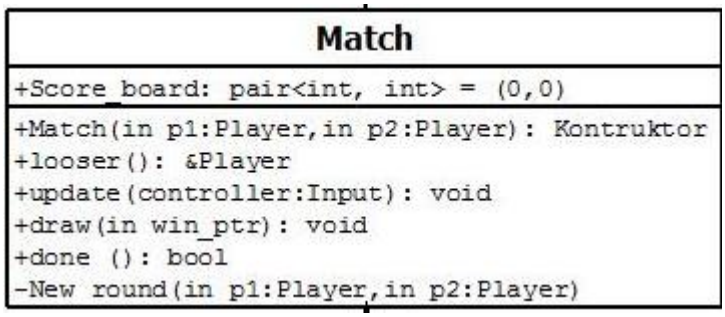
Figur 17 - innehållet av klassen bodypart.

3.4. Spellägets klasser

Delsystemet innehåller de klasser där matcherna utförs. Kan endast anropas från turneringssystemet för att uppdateras. Hanterar en spelplan och två spelare.

3.4.1. Match

Klassen kan endast skapas av två olika spelare eftersom det är vad som krävs för att spela en match. Kontrollerar om en spelare har vunnit matchen genom att se om spelaren har vunnit två matcher. Hanterar kontroller för spelare 1 och 2.

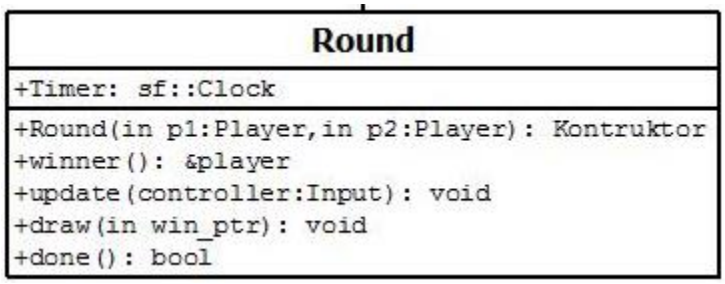


Figur 18: Innehållet av klassen Match.

Attributen är en poängställning som avgör om funktionen *New_round()* ska anropas. Funktionen anropar konstruktorn till klassen Round. När en match är avgjord returnerar funktionen *looser()* den spelare som förlorade så att denna kan tas bort.

3.4.2. Round

Konstruktorn till klassen round skapas med de två spelare som ska spela en rond. När en round skapas sätts variabeln för tiden till max. Denna variabel räknar ner tiden som är kvar på ronden tills den når noll eller tills en av spelarnas karaktärers hälsa når noll. Den spelare som har minst hälsa kvar när tiden är slut eller den spelare som har noll hälsa kvar förlorar ronden.



Figur 19: Innehållet av klassen Round.

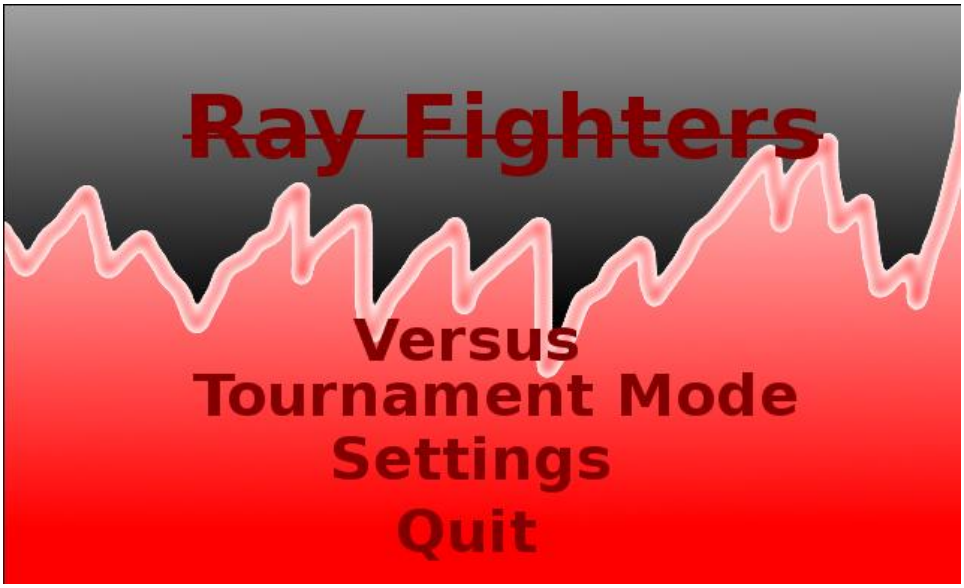
En Round ska skapas med de två parametrar *p1* och *p2,* dessa sparas privat i klassen. Funktionen *winner()* ska returnera den spelare som har vunnit matchen, *done()* ska returnera falskt fram till dess att en runda är avklarad. *update()* funktionen ska uppdatera rörelserna för båda spelarnas karaktärer med de sparade kontrollerna från controller klassen. *draw()* funktionen ska att rita ut bakgrunden, karaktärerna och spelarnas UI till fönstret som fås genom *win_ptr*.

4. Användargränssnitt

I Ray Fighters ska användargränssnittet vara designat så att en spelare slipper använda sig av en datormus och istället kan luta sig lugnt tillbaka med bara tangentbordet eller en extern joystick. Alternativval i menyer ska vara i ett tydligt format och visa spelaren vilket alternativ som är markerat genom att låta texten lysa.

4.1. Startmeny

Startmenyn ska ha en animerad bakgrundsbild, en titel samt alternativ som en användare kan välja. Temat kring bakgrunden ska vara något action dramatiskt som eld eller blixtar. I figur 20 visas hur menyn med 4 alternativ ska se ut. När användaren väljer ett av alternativen ska motsvarande meny öppnas.



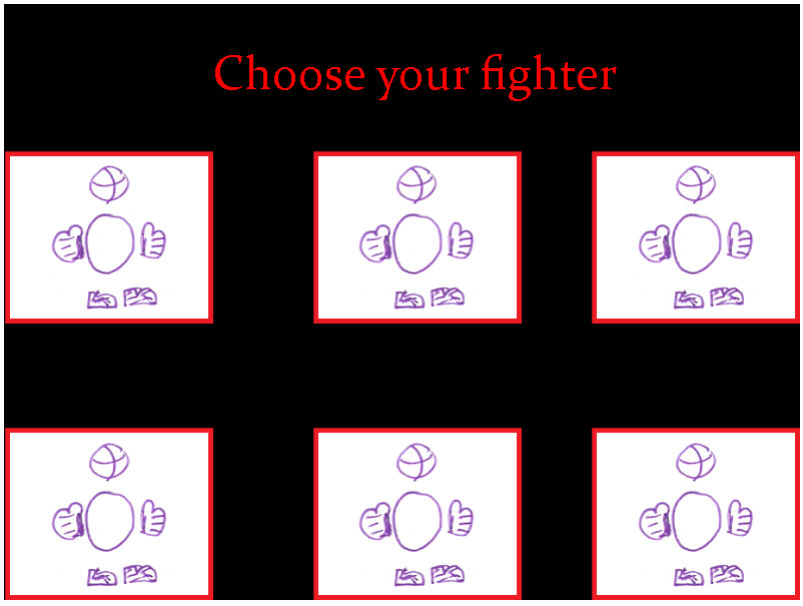
Figur 20: En stillbild av hur menyn ska se ut. Alternativen kommer att vara bättre centrerade i den färdiga produkten

4.2. Tournament mode

Vid val av alternativet Tournament mode i startmenyn ska en mellanmeny dyka upp där spelaren uppmanas välja antalet spelare som ska vara med i turneringen, detta illustreras i figur 21. Om användaren ångrar sig ska denne kunna återvända till startmenyn med alternativet *return*.

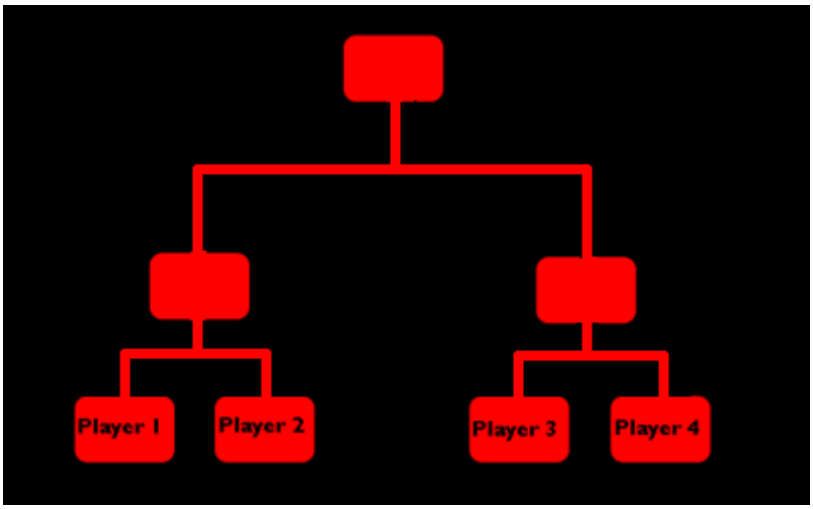
Figur 21: Menyn innan en turnering påbörjas

När antalet spelare har valts ska en karaktärmeny visas där användaren ska välja en karaktär till varje spelare som finns med i turneringen. Menyn ska fortsätta visas till dess att användaren har valt alla karaktärer. Menyn är illustrerad i figur 22.



Figur 22: bild över karaktärsmenyn

När användaren har valt karaktär till varje spelare ska spelet visa ett turneringsschema som syns i figur 23, när endast två spelare är valt ska turneringsschemat inte visas. Därefter ska matchen som är längst ner till vänster på skärmen startas. Resterande matcher till höger på samma nivå måste avgöras innan matcher på nästa nivå uppåt startas.



Figur 23: Statistik skärm för tournament mode

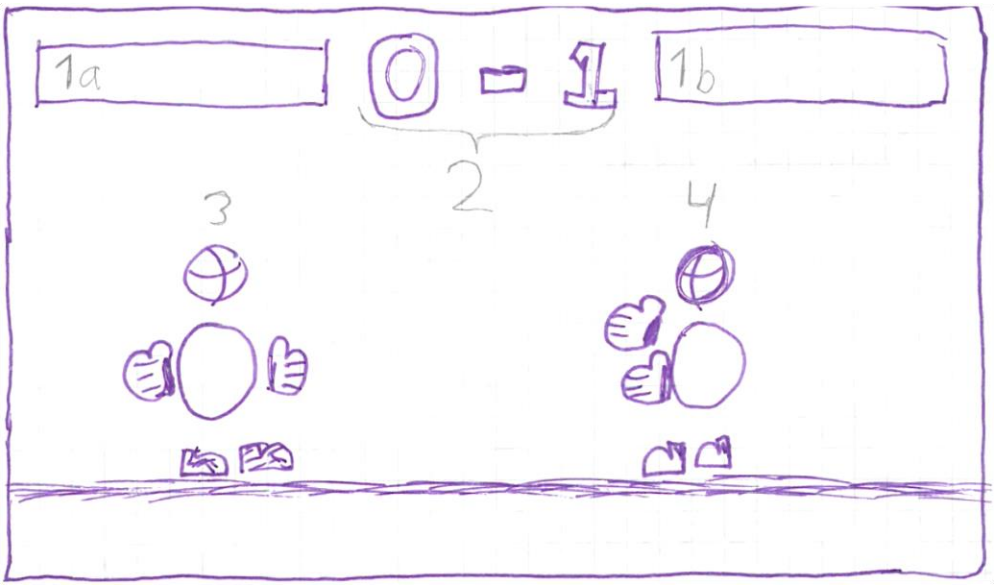
4.3. Settingsmeny

Alternativet *settings* i startmenyn ska visa inställningsmenyn enligt figur 24.
I denna meny ska användaren kunna ändra kontrollerna för båda spelare genom att navigera till en spelares keybind för ett visst alternativ, klicka enter och sedan trycka på den tangent spelaren vill binda. Det ska finnas en justerbar volymmätare och en return knapp som returnerar användaren till startmenyn.

Figur 24: Bild över settings menyn

4.4. Spelets GUI

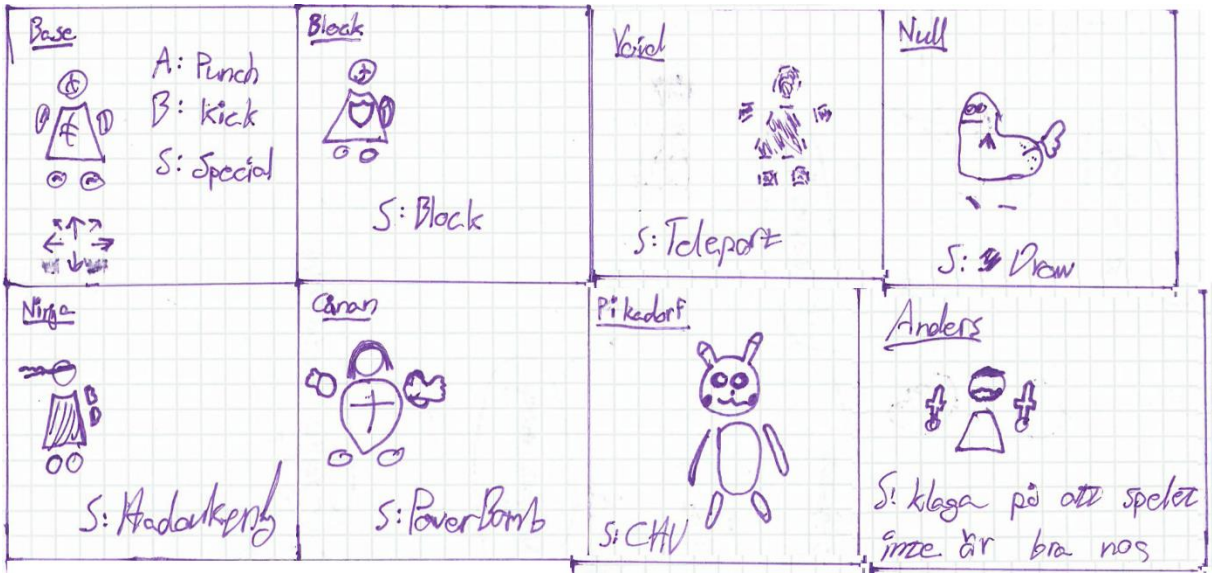
Spelets användargränssnitt är inspirerat av olika kända fighting spel som Tekken, Street Fighter och Soul Caliber. GUI:n ska bestå av en hälsomätare (1a och 1b). en VS indikator med poängräknare (2) och eventuellt en mätare för special attacker. I Ray Fighters ska det finnas en hälsomätare för varje spelare och en poängräknare som visar båda spelares aktuella poäng som kan ses i figur 25.



Figur 25: Ray Fighters GUI (1a: spelare 1's hälsa, 1b spelare 2's hälsa, 2 poängställning, 3 spelare 1, 4 spelare 2)

4.5. Karaktärsdesign

Ray Fighters har dragit inspiration till karaktärsdesign från spelet Rayman och flash animation-serien Madness där karaktärerna har osynliga armar och ben. Efter denna design har projektgruppen ritat upp olika möjliga exempel på hur färdiga karaktärer skulle kunna se ut i den färdiga produkten. Dessa exempel kan ses i figur 26.



Figur 26: Karaktärs förslag till Ray Fighters, S markerar karaktärens special attack

4.6. Vinstskärm

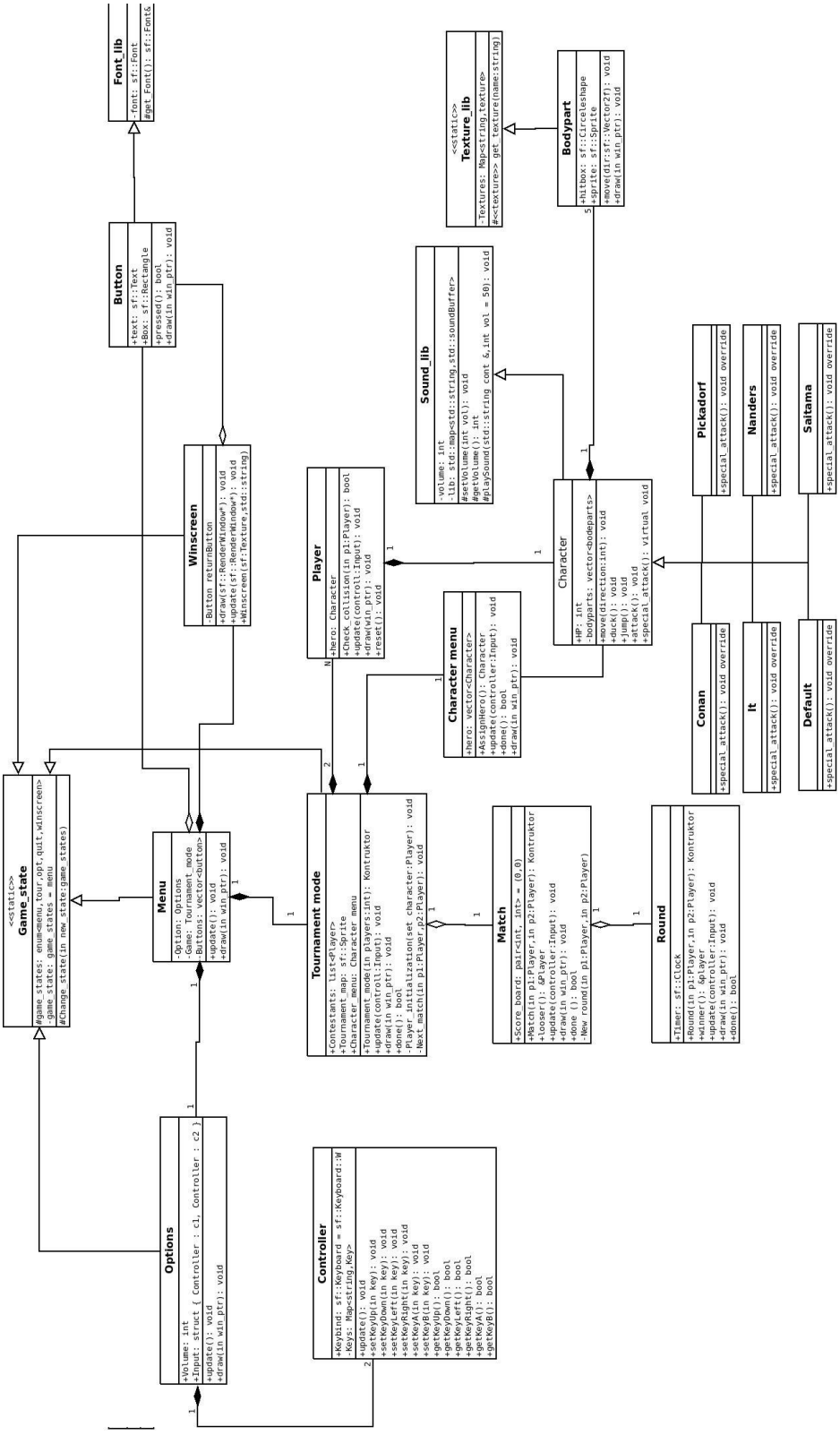
När en spelare når vinstskärmen skrivs "Winner" följt av karaktärens namn enligt figur 27. Därefter visas skaparnas namn enligt figur 28. Inspirationer kommer från en idé att göra vinstskärmen med hjälp av texturerna i spelet vilket skapar en unik design och minnesbar upplevelse.

Figur 27: Vinstskärmen ritar ut vinnaren som i detta fall är Conan

Figur 28: Vinstskärmen ritar ut namnen på skaparna (Tobias och Christoffer)

5. Bilagor

5.1. UML-klassdiagram



5.2. Revision historik

Version 1.2 2017-12-07

- Updaterat klassdiagram
- Klasserna Sound_lib, Winscreen, Font_lib, Conan, It, Saitama, Default tillagda
- Vinstskärmen tillagd i användargränssnittet

Version 1.1 2017-11-03

- Uppdaterat klassdiagram
- Klasserna Button, Texture_lib, Game_state tillagda

Version 1.0 2017-10-05

- Figurhänvisning tillagd i texten
- bildkomplettering i figur 3

Version 0.4 2017-10-04

- Komplettering

Version 0.3 2017-10-03

- Omstrukturering av klassdiagram

Version 0.2 2017-09-28

- Mer text och bilder tillagda

Version 0.1 2017-09-25, 2017-09-27

- Dokumentet skapas
- System arkitektur inlagt
- Teknisk specifikation inlagt
- Användargränssnitt inlagt