LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

Dylan Mäenpää, dylma900@student.liu.se
Albin Vedin, albve061@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2016
Version 0.2

2016-12-16

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Första utkast av designspecifikationen	161123
0.2	Designspecifikation efter att ha skrivit klart spelet	161216

2 Detaljbeskrivning av Player

Syftet med Playerklassen är att representera den karaktär spelaren styr. Spelarens karaktär styrs av spelarens tangentbordsinmatning. Detta utöver att spelaren kan skjuta projektiler är skillnaden mellan spelar och fiendekaraktären.

Player-klassen ärver av Character-klassen som ärver från Object-klassen som i sin tur ärver från Sprite-klassen

Players konstruktor tar emot 7 parametrar. Första är playerns hastighet i datatypen double, andra är boolesk som anger om spelaren är aktiv eller ej. Den tredje är hur mycket hit_points spelaren skall ha. De fyra sista är x och y koordinater samt bredd och höjd i den ordningen. Samtliga parametrar anropar Characters konstruktor.

2.1 Character

Från Character ärver Player följande metoder och variabler:

- int get_hit_points() const; - En funktion som returnerar karaktärens hit_points
- void add_hit_points(int const add_amount); - En funktion som ökar karaktärens hit_points, om hit_points blir mer än 100 så sätts den till 100.
- void take_damage(int const damage); - En funktion som sänker en karaktärs hit_points
- Direction direction; - En variabel som innehåller den nuvarande karaktärens riktning
- bool can_move(Direction const & direction, Object const & player) const; - En funktion som bestämmer om karaktären kan röra sig om objektet player inte är i vägen.
- bool can_move(Direction const & direction, std::vector<Character> const & enemies) const; - En funktion som bestämmer om karaktären kan röra sig beroende på om något objekt i vektorn är i vägen.
- sf::Clock animation_clock; - En variabel som används för att bestämmer längden på intervallet mellan skiftandet av texturer
- sf::Clock attack_clock; - En variabel som används som bestämmer ifall en karaktär kan attackera.
- int texture_counter; - En räknare som används för att identifiera var på sprite-sheetet den nuvarande texturen är.
- int hit_points; - En variabel som representerar karaktärens hit_points

Syftet med klassen är att ha en klass för alla rörliga karaktärer som har hit_points.

Characters konstruktor anropar Objects konstruktor med move_speed variabeln, aktiva boolesken, x och y koordinaterna samt bredden och höjden. I characters egna konstruktor så sätts direction variabeln till 0,0, hit_points sätts till parametern hit_points, texture_counter sätts till 0.

2.2 Object

Från Object ärver Player fölande metoder och variabler:

- bool get_active() const; - En funktion som returnerar huruvida ett objekt är aktivt eller inte
- bool interesect(Object const & object) const; - En funktion som kollar om två olika objekt kolliderar med varandra
- double get_move_speed() const; - En funktion som returnerar ett objekts move_speed
- double get_bottom() const; - En funktion som returnerar ett objekts bottom, det vill säga objektets högsta y-koordinat
- double get_top() const; - En funktion som returnerar ett objekts top, det vill säga objektets lägsta y-koordinat
- double get_left() const; - En funktion som returnerar ett objekts left, det vill säga objektets lägsta x-koordinat
- double get_right() const; - En funktion som returnerar ett objekts right, det vill säga objektets högsta x-koordinat
- void update(); - En funktion som uppdaterar ett objekts positionsvariabler
- void update_heart(); - En funktion som uppdaterar ett hjärtas positionsvariabler
- Direction generate_route(Object const & p) const; - En funktion som genererar rutten som fiender skall ta för att röra sig mot spelaren
- bool active; En variabel som representerar huruvida ett objekt är aktiva i spelets värld
- double move_speed; - En variabel som representerar ett objekts move_speed
- double bottom; - En variabel som representerar ett objekts bottom-värde, det vill säga objektets högsta y-koordinat
- double top; - En variabel som representerar ett objekts top-värde, det vill säga objektets lägsta y-koordinat
- double left; - En variabel som representerar ett objekts left-värde, det vill säga objektets lägsta x-koordinat
- double right; - En variabel som representerar ett objekts right-värde, det vill säga objektets högsta x-koordinat

Syftet med klassen är att ha en klass för alla objekt som kan kollidera med andra objekt. Objekten behöver nödvändigtvis inte ha hit_points som Character-klassen har.

Objects konstruktor anropar Sprites konstruktor med x och y koordinaterna samt bredden och höjden. I objects egna sätts move_speed variablen samt active variabeln med parametrarnas värde.

2.3 Sprite

Från Sprite ärver Player följande metoder och variabler:

- sf::Sprite object; - En variabel som representerar objektet i spelvärden
- double x; - En variabel som representerar ett objekts x-koordinat
- double y; - En variabel som representerar ett objekts y-koordinat
- int width; - En variabel som representerar ett objekts bredd
- int height; - En variabel som representerar ett objekts höjd

Spriteklassen används till för att samla alla egenskaper alla synliga objekt har på skärmen.

Sprites egna konstruktor sätter x och y samt width och height enligt parametrarna. I konstruktorn sätts också spriten object x och y värden med hjälp av sprite funktionen setPosition enligt parametrarna.

3 Detaljbeskrivning av MenuState

MenuState är den klass som tar hand om alla element när det kommer till att en meny är aktiv. Den tar hand om både start_menu och gameover_menu. Detta fungerar eftersom de två olika menyerna är identiska, det ända som skiljer är den bakgrundsbild som visas och huvudbilden som antingen är en bild med spelnamnet eller en bild som visar att du förlorat spelet.

I konstruktorn sätts playing till false då spelet ej är igång. Variabeln animation counter sätts till 0 då första animationen är på position 0 på vår rörliga karaktär vi har i menyerna. Variablen going_right sätts till true då karaktären börjar röra sig höger. Texture_counter sätts till 0 då första texturen är på position 0.

3.1 MenuState

MenuState innehåller följande metoder och variabler:

- virtual int run (sf::RenderWindow &app) override; - Huvudloopen som körs när denna klass är aktiv
- int get_highscore() const; - En funktion som hämtar information från “highscore.txt” och returnerar den nuvaranda högsta poängen
- int get_kill_count() const; - En funktion som hämtar information från “highscore.txt” och returnerar antalet dödade fiender. Denna funktion är skapad och fungerar men är inte implementerad i spelet
- void animate_character(); - En funktion som styr animeringen av en karaktärer som går fram och tillbaka på menyskärmen.
- sf::Texture play1; - En variabel som innehåller en texturvariant av spelknappen
- sf::Texture play2; - En variabel som innehåller en texturvariant av spelknappen
- sf::Texture quit1; - En variabel som innehåller en texturvariant av knappen som avslutar spelet
- sf::Texture quit2; - En variabel som innehåller en texturvariant av knappen som avslutar spelet
- sf::Texture head_texture; - En variabel som innehåller texturen för menyns huvudbild
- std::vector<sf::Texture> textures; - En vector som innehåller alla karaktärstexturer det vill säga: orc, human, skeleton
- sf::Sprite play_button; - En variabel som representerar spelknappen, det är denna som ritas ut på skärmen
- sf::Sprite quit_button; - En variabel som representerar knappen som avslutar spelet, det är denna som ritas ut på skärmen
- sf::Sprite head_sprite; - En variabel som representerar huvudbilden, det är denna som ritas ut på skärmen
- sf::Sprite character; - En variabel som representerar karaktären som går runt på menyn, det är denna som ritas ut på skärmen
- sf::Text highscore; - En variabel som representerar texten som ritas ut på skärmen som då är högsta poängen
- int menu 0; - En variabel som håller koll på vilken del av programmet som skall vara aktiv
- int sprite_counter1; - En variabel som håller koll på x-positionen i den aktuella sprite-sheeten

- int sprite_counter2; - En variabel som håller koll på y-positionen i den aktuella sprite-sheeten
- int texture_counter; - En variabel som håller koll på den nuvarandra positionen i std::vector<sf::Texture> textures och ser till så att programmet inte letar efter en textur på en plats i vectorn där det inte finns någon textur
- bool playing; - En variabel som håller koll på huruvida spelet är aktivt eller inte
- bool going_right; - En variabel som håller koll på åt vilket håll karaktären på menyskärmen rör sig
- Direction dir; - En variabel som innehåller karaktärens riktning

Den enda lokala variabeln i denna klass och dess huvudfunktion är den karaktären som rör sig runt på skärmen.

3.2 States

Från States ärver MenuState följande metoder och variabler:

- virtual int run (sf::RenderWindow &App) = 0; - En virtuell funktion som MenuState skriver över och använder för att köra sin primära menyloop.
- bool running; - En variabel som representerar huruvida den primära loopen skall vara aktiv eller inte
- sf::Sprite background; - En variabel som innehåller bakgrundsbilden som skall ritas ut på spelskärmen
- sf::Texture background_texture; - En variabel som innehåller texturen som "background" skall använda
- sf::Font font; - En variabel som representerar den aktuella fonten som skall användas när text skrivs till spelskärmen
- sf::Event event; - En variabel som innehåller och hanterar de olika händelserna som spelet skall utföra
- sf::Texture player_texture; - En variabel som innehåller ett sprite-sheet av en människa, denna används som textur för spelaren
- sf::Texture enemy_texture1; - En variabel som innehåller ett sprite-sheet av en orc, denna används som textur för fiender
- sf::Texture enemy_texture2; - En variabel som innehåller ett sprite-sheet av ett levande skelett, denna används som textur för fiender

4 Beskrivning av designen

Vår design bygger på en game klass som har kontroll över 2 spelstadier. Dessa stadier är MenuState och PlayState. Stadierna skapas och läggs i en vektor när game kör sin funktion run. De två olika stadierna ärver av klassen States som innehåller de metoder och attribut som de har gemensamt. Precis som kursens mål så är all kod uppdelad i klasser vilket gör det lättare att hitta och hantera vårt program.

Sprite är klassen alla spelobjekt ärver ifrån. Detta ger dem ett antal gemensamma metoder och attribut. Under spriteklassen finns en till abstrakt klass, Object, som innehåller allt ett spelobjekt kan tänkas behöva. Klasserna Heart och Projectile ärver från Objekt. Under Objekt finns ännu en till klass som heter Character. Character är innehåller de metoder och attribut som både Player och Enemy använder. Vår Player och Enemy klass har just nu inga extra metoder eller attribut, det som skullle kunna göras är att dela upp fiendehanteringen i PlayState till Enemy.

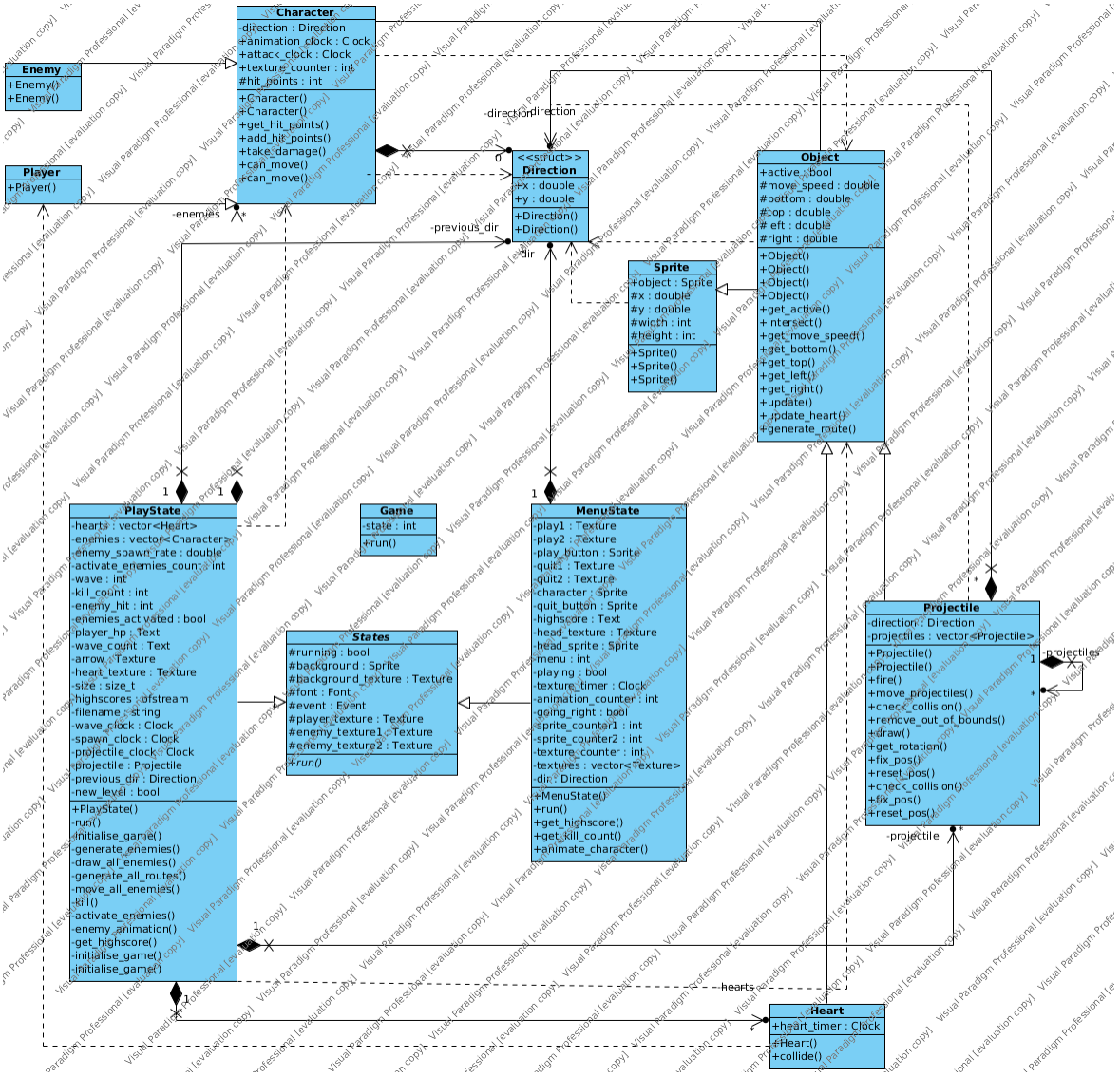
En svaghet i designen skulle vara klasserna PlayState och MenuState. Dessa har för mycket ansvar och skulle kunna delas upp för att öka abstraktionen och läsbarheten.

En annan svaghet i designen är att varje projektil som skapas innehåller en egen vektor, som inte egentligen används förutom för det första projektilobjektet vi skapar som tar hand och innehåller alla andra projektiler. Detta skulle kunna åtgärdas genom att skapa en vektor i PlayState som innehåller alla projektiler och sedan ha ett projektilobjekt och med hjälp av det kalla på de olika projektilfunktioner och skicka in vektorn som inparameter.

5 Externa filformat

De externa filformatet som används i Box Wars är en oformaterad textfil(.txt). Denna sparar dels högsta vågen som har nåtts och antalet fiender som har dödats under det aktuella spelets gång. Notera att antalet döda fiender sparas men används inte i spelets aktuella form, detta då vi kände att det blev förvirrande att både presentera högsta vågen spelaren nått och antalet fiender som dödats. Dessa två värden separeras med ett "|" tecken. Hämtningen av värdena görs med hjälp av formaterad inläsning.

Konsekvensen av att använda sig av en sådan simpel struktur blir att användaren lätt kan ändra de värden som ligger inne i textfilen och få det att se ut som att hen klarat sig längre än vad hen egentligen gjort.
Inmatningen till textfilen är dock uppbyggd på ett sådant sätt så att om användaren antingen förstör eller ändrar filen så att det inte ser ut som den borde göra så kommer spelet automatiskt rätta till filen nästa gång spelet körs.
Något man skulle kunna implementera för att uppgradera denna struktur skulle kunna vara att man gömmer högsta poängen och dess värde i en stor textfil som innehåller en mängd olika karaktärer men att man själv implementerar ett sätt för programmet att hämta det rätta värdet. På detta sätt skulle inte spelaren kunna hitta det rätta värdet och fuska sig fram till en bra högsta poäng.



Figur 1: UML-Diagram på designen