LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

Josefin Bodin
Alicia Bergman
Ahmed Sikh

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2019
Version 1.1

17 december 2020

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.1	Färdig designspecifikation	201217
1.0	Första utkast	201124

2 Detaljbeskrivning player

Syftet med Player-klassen är att representera den karaktär som användaren kontrollerar på spelplanen. Den har utseendet av ett "kakmonster" och styrs genom den utritade labyrinten.

Player-klassen ärver funktionalitet från entity-klassen som i sin tur ärver från objekt-klassen.

Konstruktorn som finns i Player tar inte in några parametrar. När ett objekt av typen player skapas sätts texturen för spriten till den bild som ser ut som karaktären. Förutom texturen sätts även skalan på spriten, den har även en default position som spelaren får. Förutom det får variablerna som representerar liv ett värde av 3 och hastigheten får värdet 150. I konstruktorn anropas även Entitys konstruktor, till den skickas bilden, skalan och positionen.

De medlemsvariabler som player har är heltal variablerna points och lives. Sedan har den också en float som heter speed, den har spelarens hastighet lagrad. Den innehåller även bool variabler som säger om spelaren kan flytta i en viss riktning(canMoveUp, canMoveDown, canMoveRight och canMoveLeft).

Metoder som finns i klassen är bland annat input som sätter medlemsvariabler till rätt värden beroende på den inmatning som användaren har gjort. I Game-klassen skickas olika riktningar med beroende på vad användaren har matat in.Input kallar på och kör move-funktionen.

I move så sker själva förflyttningen av spriten. Det kollas även så att spelaren kan flytta på sig. Det finns även funktionerna get_lives, set_lives, get_points och set_points för att de andra objekten skall kunna uppdateras vid kollision. set_speed är också en funktion som tillåter ändringa av spelarens hastighet. Det finns även en metod som säger hur spelaren ska agera i de tillfällen när den kolliderar med en vägg, den heter collision_with_wall. Update gör ingenting i player.

2.1 Entity

Det som player ärver från entity är följande:

- int move_direction - En variabel som har värdet av den riktningen som spelaren rör sig i. De olika riktningar har fått nummer som representerar dem.
- sf::Time movement - En variabel till för att spelaren skall ha samma hastighet på olika datorer.
- virtual void move(sf::Time) = 0 - Funktion som inte har någon implementation i entity-klassen.
- virtual void update(sf::Time) = 0 - Funktion som inte har någon implementation i entity-klassen.

2.2 Object

Det som player ärver från objekt är följande:

- sf::Sprite sprite - Variabel som får en textur som representerar vår spelare.
- sf::Texture texture - Variable som håller våran texture.
- void draw(sf::RenderWindow&) - Funktion som gör det möjligt att rita ut objekten.
- virtual void update(Player&, sf::Time) = 0 - Funktion för att uppdatera spelaren.

- bool collision(Object const&) const - För att kolla om två objekt har kolliderat.
- sf::Sprite getsprite() const - Ger tillgång till spriten för alla objekt.

3 Game

I denna klassen körs hela spelet, den uppdaterar spelplanen och spelaren samt tar hand om användarens inmatning.

Game-kassen samarbetar mycket med world-klassen då det är den som håller all objekten.

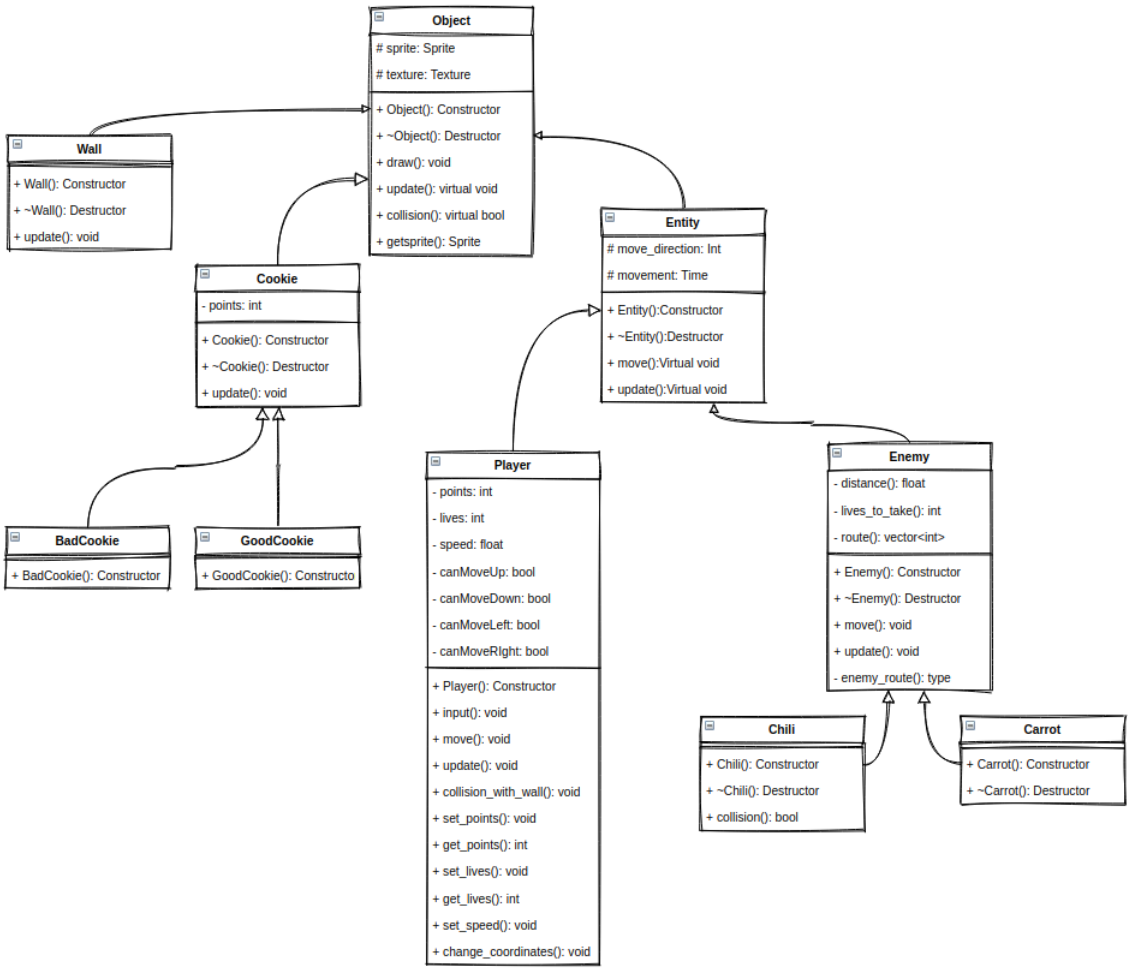
Det skickas in en parameter i konstruktorn som är namnet på den banan som ska köras. Med den initieras ett world objekt som ligger i game. I klassens konstruktor sätts även texten som kommer synas på skärmen som visar hur mycket poäng och liv spelaren har i början av spelet. Den texten uppdateras under spelets gång.

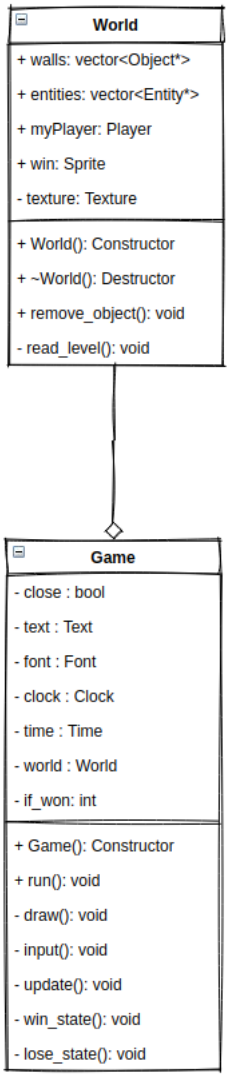
De medlems variabler som Game har är en bool som säger om spelet ska fortsätta köras eller om det ska stängas ner. Det finns även variablerna sf::Text och sf::Font har hand om hur texten som visas på skärmen ser ut och en sf::Clock och sf::Time som användes för att spelet ska köras likadant på alla datorer. Det finns även ett objekt av typen world som håller alla objekten som game ritar ut och uppdaterar.

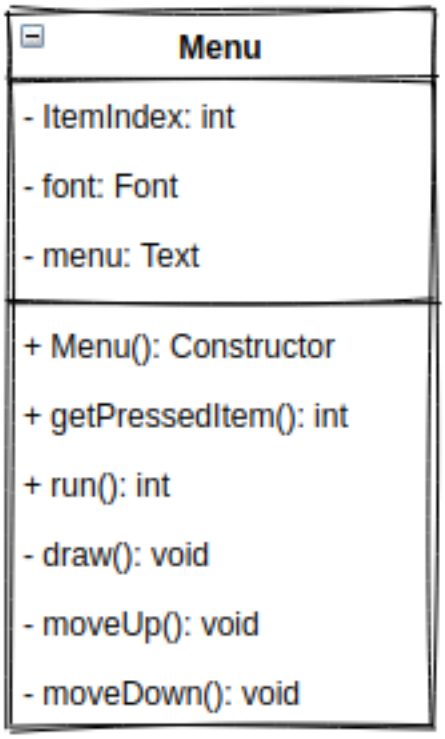
Bland de metoder som Game har finns bland annat en funktion som heter run. Den kör hela spelet. I run startas klockan för spelet om och de funktioner som ligger i game kallas på. Input är en av dem, den tar den informationen som ges när användaren trycker på knappar på tangentbordet. Beroende på inputen som ges så skickas olika parametrar in till player för att den sedan ska röra sig. I update-funktion så uppdateras spelet och uppdateringen ser annorlunda ut beroende på om någon kollision sker. Det finns även en draw funktion som ritar ut alla objekten som finns på spelplanen och funktioner som kollar om spelaren har vunnit eller om det är game over. Detta kontrolleras i update-funktionen.

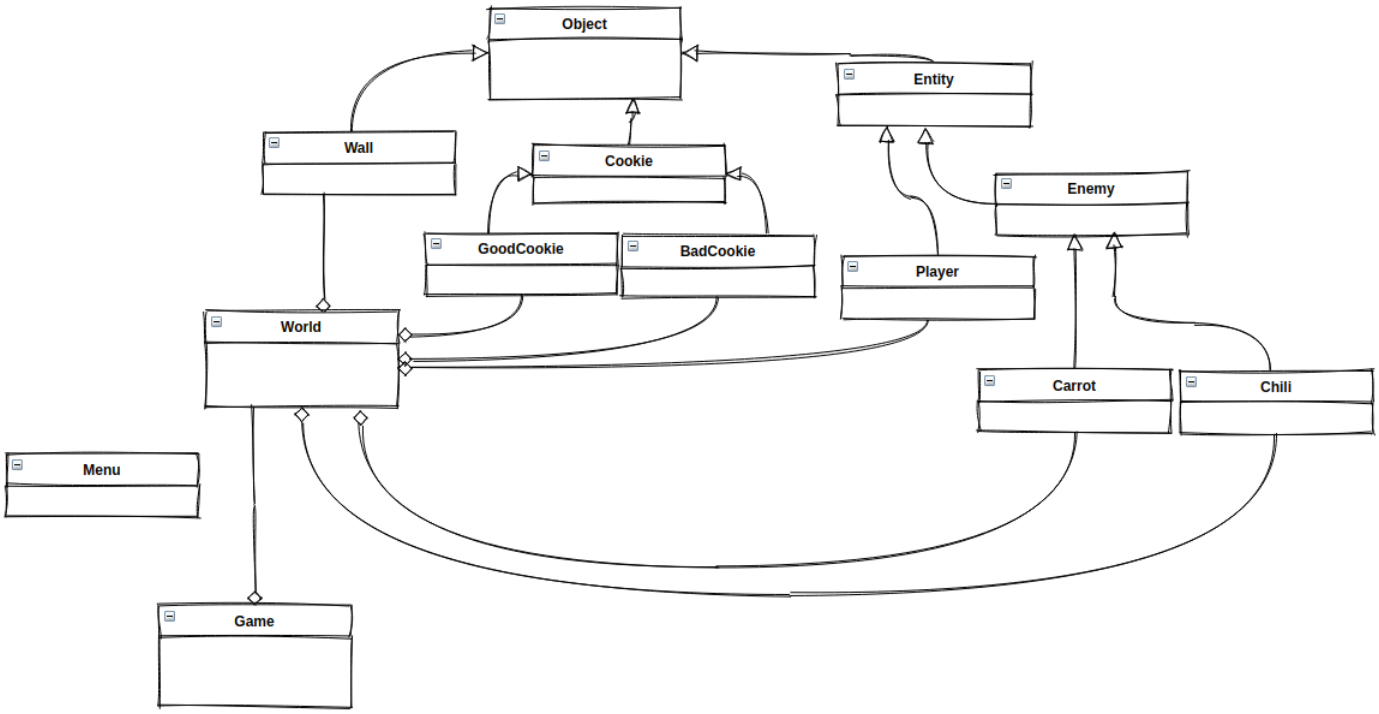
4 Extern filhantering

För att läsa in olika banor så skrivs de i en fil som läses in av world-klassen. Den informationen som läses in från filen initierar nya objekt som läggs på heapen och i vektorn finns pekare till de objekten.


Object
sprite: Sprite
texture: Texture
+ Object(): Constructor
+ ~Object(): Destructor
+ draw(): void
+ update(): virtual void
+ collision(): virtual bool
+ getsprite(): Sprite
Wall
+ Wall(): Constructor
+ ~Wall(): Destructor
+ update(): void
Cookie
- points: int
+ Cookie(): Constructor
+ ~Cookie(): Destructor
+ update(): void
Entity
move_direction: Int
movement: Time
+ Entity():Constructor
+ ~Entity():Destructor
+ move():Virtual void
+ update():Virtual void
BadCookie
+ BadCookie(): Constructor
GoodCookie
+ GoodCookie(): Constructo
Player
- points: int
- lives: int
- speed: float
- canMoveUp: bool
- canMoveDown: bool
- canMoveLeft: bool
- canMoveRIght: bool
+ Player(): Constructor
+ input(): void
+ move(): void
+ update(): void
+ collision_with_wall(): void
+ set_points(): void
+ get_points(): int
+ set_lives(): void
+ get_lives(): int
+ set_speed(): void
+ change_coordinates(): void
Enemy
- distance(): float
- lives_to_take(): int
- route(): vector<int>
+ Enemy(): Constructor
+ ~Enemy(): Destructor
+ move(): void
+ update(): void
- enemy_route(): type
Chili
+ Chili(): Constructor
+ ~Chili(): Destructor
+ collision(): bool
Carrot
+ Carrot(): Constructor
+ ~Carrot(): Destructor


World
+ walls: vector<Object*>
+ entities: vector<Entity*>
+ myPlayer: Player
+ win: Sprite
- texture: Texture
+ World(): Constructor
+ ~World(): Destructor
+ remove_object(): void
- read_level(): void
Game
- close : bool
- text : Text
- font : Font
- clock : Clock
- time : Time
- world : World
- if_won: int
+ Game(): Constructor
+ run(): void
- draw(): void
- input(): void
- update(): void
- win_state(): void
- lose_state(): void


Menu
- ItemIndex: int
- font: Font
- menu: Text
+ Menu(): Constructor
+ getPressedItem(): int
+ run(): int
- draw(): void
- moveUp(): void
- moveDown(): void


Object
Entity
Wall
Cookie
Enemy
GoodCookie
BadCookie
Player
World
Carrot
Chili
Menu
Game