Requirements and Analysis Document for Group 29

Sally Chung, Amanda Dyrell Papacosta,
Eric Erlandsson Hollgren, Jonatan Jageklint, Zoe Opendries

Oktober
version 2

1 Inledning

Strategispel har för användaren varit något som under en lång tid tillfredsställt dennes kognitiva förmåga. Det är den kreativa problem lösningen som ger användaren möjligheten till att fritt tänka hur den ska bemöta sina problem. Genren belönar spelaren genom sitt väl planerande och sina taktiska val som genomförs under spelets gång. Spelet har ett genomgående ”enkelhets”-tema där förståelsen för hur spelet fungerar inte är komplicerat. Av den anledningen är spelet nybörjarvänligt och ger möjligthet för spelaren att utveckla sina kunskaper om både spel likväl strategitänk.

Syftet med applikationen är att kunna erbjuda användaren ett enkelt tvådimensionellt ”Age of War” efterliknande spel, med inslag av tower defense. Den största skillnaden är att, förutom att planera hur exempelvis sitt torn ska uppgraderas, får användaren genom sin 2D karaktär slåss mot fiender. Projektet är riktat till personer som vill utmana sitt strategitänk, men också personer som vill ha ett lättsamt spel.

1.1 Definitions, acronyms, and abbreviations

- DoD - Definition of Done
- MVC - Model View Controller
- SOLID - Single responsibility principle, open closed principle, Liskovs substitution principle, interface segregation principle, dependency inversion principle

2 Requirements

I detta kapitel beskrivs projektets user stories, DoD och user interface. För DoD:en och user stories som har skrivits, har vi använt av oss av trello. Dessa har sedan skrivits på engelska för att utforma uppgifter till varje story.

2.1 User Stories

User story: Player

Story Identifier: 1.1

Story Name: player

Description

As a player I would like to have a main character which is able to move sideways.

Confirmation

Confirmation: A player character can move across the game-world.

Functional:

- The player is able to move sideways
- The player is shown on screen
- The character is moved by using 'a' and 'd'

Non-functional

- Connect keybindings

User story: Enemy

Story Identifier: 1.2

Story Name: Enemy

Description

As a player I would like to have an enemy show up and move across the screen.

Confirmation

Confirmation: An enemy can be created which moves across the game world.

Functional:

- The enemy is shown on screen
- The enemy's position is updated every second

Non-functional

User story: Base and Turrets

Story Identifier: 1.3

Story Name: Base and Turrets

Description

As a player I would like to have a base with turrets to protect.

Confirmation

Confirmation: A base exists on a fix position, has health and can contain turrets.

Functional:

- The base exists on a position.
- The base has health.
- The base can hold turrets.

Non-functional

- Passes unit tests.
- Adds turrets to an ArrayList that can only hold turrets.

User story: UI

Story Identifier: 1.4

Story Name: UI

Description

As a player I would like to have a UI for HP, gold and points so that I am aware of how much I have left.

Confirmation

Confirmation: A player can see the UI during the game and see it update.

Functional:

- The UI is displayed on screen.
- The score, hp and gold updates during the gameplay.

Non-functional

Timer and multiplier

Story Identifier: 1.5

Story Name: GameTimer and RoundHandler

Description

As a player I would like to have enemies provide a challenge throughout the game

Confirmation

Confirmation: GameTimer and RoundHandler classes are implemented with specified functionality.

Functional:

Non-functional

- Is able to pass the unit tests
- A good function is used for calculating the multiplier
- Can only be one timer in the game
- The in game timer can be used by other objects
- The roundHandler can return the multiplier to client code

User story: Projectile

Story Identifier: 2.1

Story Name: Projectile

Description

As a player I would like the projectiles to deal damage when health functionality is implemented.

Confirmation

Confirmation: A projectile moves across the screen that collides with enemies and/or the ground.

Functional:

- Projectile collides with enemies
- Projectile collides with ground
- Projectile moves

Non-functional

- Add projectile
- Add projectile collision
- Add damage attribute to projectile
- Passes unit tests

User story: Wall & ground

Story Identifier: 2.2

Story Name: worldBoundaries

Description

As game objects I would like to have a world to stand on. reason

Confirmation

Confirmation: Units stand on a world and can move within its boundaries.

Functional:

- There are walls within the sides of the desktop-launcher
- There is a ground which the entities stands on

Non-functional

- Collision between entities and these are not fixed

User story: CollisionDetection

Story Identifier: 2.3

Story Name: collisionDetection

Description

As a unit I would like to know where the world ends and to achieve damage when colliding with an enemy.

Confirmation

Confirmation: Attacks of units collide and units don't move through walls.

Functional:

- Collision method should be working for a player
- The player should collide with the walls (not go through them)
- Collision between enemy and player

Non-functional

- A check for player's width and x-position and block's width, height, x-position and y-position
- Same as above but instead compare it with enemy's width and x-position

User story: Enemy

Story Identifier: 2.4

Story Name: Enemy

Description

As a player I want enemies to enter the game in waves

Confirmation

Confirmation: A 'Wave' (or group) of enemies are sent out into the playing field in intervals of 30 seconds.

Functional:

- The enemies in the wavelist are spawned every 30 seconds to then be rendered on the screen one after the other.

Non-functional

- Is able to pass the unit tests.
- Adds a set number of enemies to a list
- Renders each enemy in said list

User story: Upgrade Tower and Build Turrets

Story Identifier: 2.5

Story Name: Upgrade Tower and Build Turrets

Description

As a game-player I would like to upgrade my tower to look and be stronger and upgrade it with turrets, so that I can better protect my tower.

Confirmation

Confirmation: Tower can be upgraded, add Turrets to it and see these updates visually.

Functional:

- Tower can be upgraded.
- Turrets can be built on the Tower.
- Tower represents visually and updates its texture with upgrades.

Non-functional

- Passes unit tests.
- Listens to actions and tells Tower to upgrade and add turrets.
- TowerView updates its textures accordingly by looking at Tower.

User story: Healthbar

Story Identifier: 2.6

Story Name: Healthbar

Description

As a player I would like to see how much health me and my enemies have left.

Confirmation

Confirmation: The player and enemies have healthbars that follow them.

Functional:

- The healthbar displays the current health.
- The healthbar follows the enemies and player.

Non-functional

- Healthbar should update when player takes damage

User story: Enemy doing damage

Story Identifier: 3.1

Story Name: EnemyAttack

Description

As an enemy I am able to do damage to the main character so the character has a goal.

Confirmation

Confirmation: Enemy is able to do damage to Tower and Player.

Functional:

- Enemy is able to do damage
- Player is taking damage from this
- Tower is taking damage from this

Non-functional

- Enemy is able to do damage through collision
- Should connect with doing damage to the main character
- Player's health should decrement when taking damage from enemy
- Tower's health should decrement taking damage from enemy

User story: Player doing damage

Story Identifier: 3.2

Story Name: PlayerAttack

Description

As a player I would like to be able to deal damage to enemies.

Confirmation

Confirmation: Enemy takes damage from the player, and is removed when its health is less than or equal to zero.

Functional:

- Player is able to do damage
- Enemy is taking damage
- Enemy's health should decrement
- When killing an enemy it should be removed, points should be added and gold too
- When an enemy dies, points and gold are gained

Non-functional

- Call the mainhandler which sends request to goldhandler and pointhandler
- Enemy is being deleted after when health is less than or equal to zero

User story: Player's movements are animated

Story Identifier: 3.3

Story Name: Player animation

Description

As a player I would like the body of the main character to be animated.

Confirmation

Confirmation: Player has animations for when it is idle, aswell as its movements, attack and death.

Functional:

- Player's movements are animated, such as idling, running, dying and attacking.

Non-functional

- When rendering it the player's movements state should update depending on which button is pressed or not pressed.

User story: Projectile

Story Identifier: 3.4

Story Name: Projectile

Description

As a player i would like the projectiles from my tower to deal damage to the enemies after refactoring of the enemy code. This is user story is the same as 2.1, it needed an update when more abstractions were implmented.

Confirmation

Confirmation: Projectiles move across the screen and collides with either an enemy or the ground.

Functional:

- Projectile collides with enemies
- Projectile collides with ground
- Projectile moves

Non-functional

- Add projectile
- Add projectile collision
- Add damage attribute to projectile
- Passes unit tests

User story: ViewHolder

Story Identifier: 3.5

Story Name: ViewHolder

Description

As a developer I would like a class that holds all the objects created

Confirmation

Confirmation: All objects instantiated at the start of the application is created and held in ViewHolder.

Functional:

No-functional:

- Refactor abstract and views even more

User story: Entity

Story Identifier: 3.6

Story Name: ViewHolder

Description

As a developer I would like the enemy and player to be an entity

Confirmation

Confirmation: Enemy and Player extends the entity class.

Functional:

- Enemies and player are entities

No-functional:

- Refactor enemy and player class to extend an entity class

User story: Buttons

Story Identifier: 3.7

Story Name: Buttons

Description

As a player I would like to press on a button to upgrade my gadgets, and be able to upgrade turrets. reason

Confirmation

Confirmation: Buttons are viewable and clickable and appears as specified when running the application, and when clicked, expected functionality is executed.

Functional:

- Buttons upgrade turret and tower
- Buttons are viewable and clickable in running of application.
- Tower and turrets upgrade when clicking upgrade buttons.

Non-functional

- Refactor instantiations of views to app. (Required to assert correct dependency.)
- Buttons are instantiated in ButtonView and acts as listener for clicks
- When an action occurs it notifies Controller that actions happened, and then Controller calls to Model to update itself.
- Implement turret upgrade.
- Make tests that check that a turret upgrades.

User story: Finishing game

Story Identifier: 3.8

Story Name: GameOver

Description

As a player I would like to know when my game is finished

Confirmation

Confirmation: The player can die, and informs the player in the event of that happening.

Functional:

- Player can die
- A pop-up screen shows up when player is dying

Non-functional

- A playerDead() method should be implemented, and check at time when player dies.
- Player should be removed from screen when dying

User story: Respawning

Story Identifier: 4.1

Story Name: Player respawn

Description

As a player I would like to be able to respawn

Confirmation

Confirmation: The Player can respawn by pressing a Respawn button.

Functional:

- When player has died, the player should be able to respawn

Non-functional

- The health should start from its startin value again after being respawned.
- The player should know when it is able to respawn again

User story: Menu

Story Identifier: 4.2

Story Name: Menu

Description

As a player I would like to have a menu to guide me to the game

Confirmation

Confirmation: There is a Play button which starts the game, a Quit button that exits the game and a Pause button which pauses the game.

Functional:

- There is a play button
- There is a quit button
- There is a way to pause the game

Non-functional

- Pause button is connected with escape
- First thing to show is the play and quit button
- When clicking on the play-button, the game should start
- When clickin on the quit-button, the game should close

2.2 Definition of Done

I projektet har vår ”Definition of Done” bestått av:

- Is able to pass the unit test
- User story should complete its acceptance criteria
- Code is reviewed by at least one person
- Code should be documented

- The code should be motivated from an OOP-perspective
- The code should follow/be SOLID
- The code should follow the MVC-pattern

Det som kan vara viktigt att notera är att all kod bör ha kunnat reflekterats kring SOLID-principerna och MVC-pattern. Detta för att erhålla ett sådant objektorienterat tankesätt som möjligt, för att uppfylla kursens mål. Våra DoD var under projektets gång till för att uppfyllas innan en merge till main skulle ske.

2.3 User interface

I detta avsnitt kommer applikationens första skisser att visas samt slutresultatet av det hela. Dessutom kommer förklaringar för hur det fungerar att beskrivas.

2.4 Tidiga iterationer

Figure 1: Första iteration av spel

Första iterationen som visas i fig. 1 är en tidig skiss gjord i figma för att på ett ungefärligt sätt visa hur applikationen skulle formas. Bland annat syns två karaktärer, ett torn, en user interface som indikerar antalet guld, knappar vid vänstra hörnet och även att båda karaktärer har varsina healthbars. Att ena karaktären skulle ha en indikation på att skada har tagits, syns genom att låta den gråa färgen i healthbaren vara synlig. Medan en healthbar som är helt grön istället innebär att karaktärens liv är fullt; den har inte tagit skada.

Knapparna i det vänstra hörnet visar tidiga skisser på hur tornet kan uppgraderas med att tillägga turrets (mindre torn till det större tornet). Tornen skulle isåfall uppgraderas efter att ha ihoptjänat x-antal guld som spelaren skulle få genom att ha dödat fienden.

2.5 Slutprodukt

Den slutliga produkten och även det som kom att bli projektet är det som visas nedan.

Figure 2: Start menyn av spel

För att påbörja spelet, eller välja att gå ur spelet, genom att klicka på knapparna som visas i fig. 2 så går den vidare till nästa vy eller stänger ner spelet helt och hållet.

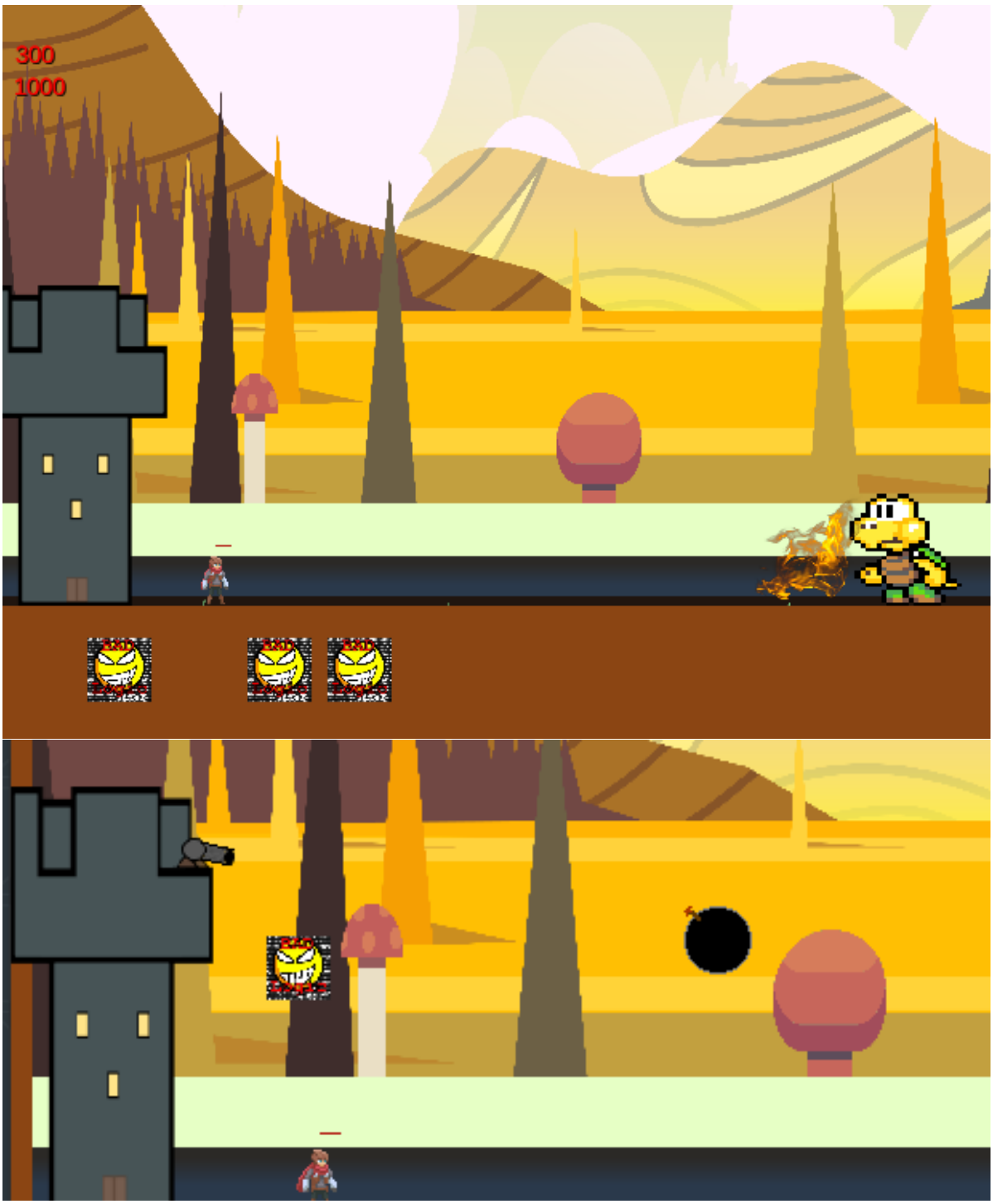

Figure 3: Början av själva spelet

Skulle personen som spelar spelet välja att klicka på ”play” kommer den till nästa vy, som visar själva spelplanen. Det fig. 3 är en ”main character” som spelaren är,

och kan styra. Genom att använda knapparna ’a’, ’d’ eller piltangenterna, förflyttas karaktären i spelet varpå det röda strecket (spelaren healthbar) kommer följa efter.

I vänstra hörnet kan poängen och guldet ses som spelaren börjar med. När fiender har dödats kommer dessa att uppdateras. När spelaren har insamlat tillräckligt med guld, kan en turret köpas genom att klicka på mittenknappen. Från det tillagda turret kommer kanonkulor att skjutas, vilket den nedre bilden av de två, som visas i fig. 3,visar.

Figure 4: Rendering av enemies.

Under en runda kommer det komma 10 fiender i olika omgångar. I fig. 4 visar även att poängen och guldet har uppdaterats när siffrorna jämförs med fig. 3

Figure 5: Meddelanden från tredje knappen

När den tredje knappen klickas på, som visas i till exempel fig. 4 kommer någon av de meddelanden som visas i fig. 5 att synas. Spelaren kan antingen återuppstå igen i spelet för att samla poäng. Skulle spelaren dö så måste spelaren vänta 5 sekunder innan den kan återuppstå i spelet. Det tredje alternativet visar meddelar att spelaren inte är död och inget kommer då att hända än att ett meddelande visas.

Figure 6: Meddelande från den första knappen

Om spelaren inte skulle ha tillräckligt med guld för att uppgradera sitt torn, kommer meddelandet i fig. 6 att visas.

3 Domain model

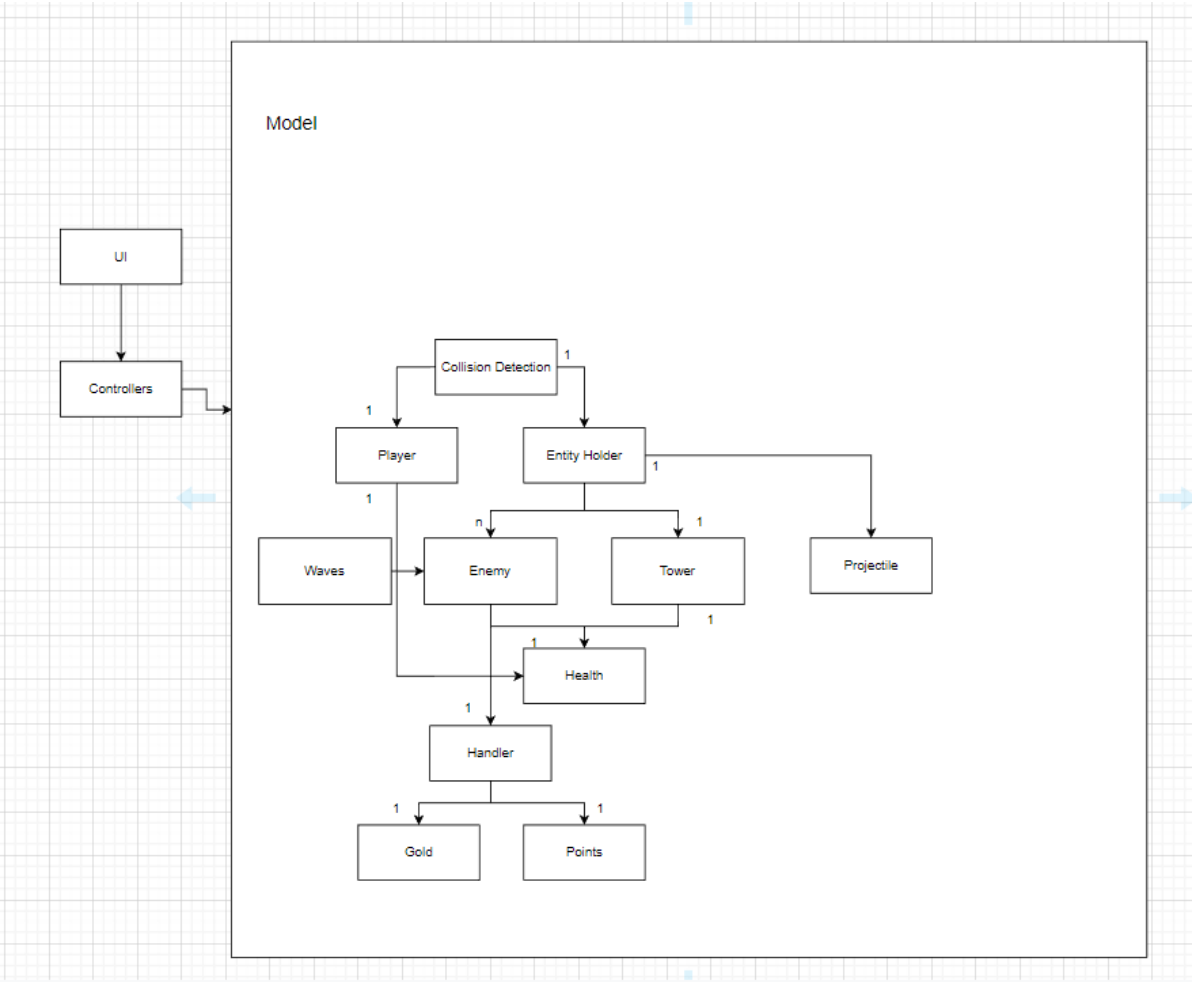

Figure 7: Domänmodell

3.1 Class responsibilities

Bilden ovan visar domän modellen för spelet så som den ser ut nu. Grund principen för strukturen är baserad på MVC.

3.1.1 Model-classes

CollisionDetection klassen hanterar kollisioner mellan spelaren och kanterna av skärmen samt kollisioner mellan olika aktörer på skärmen.

Enemy klassen hanterar skapandet av en enemy och ärver attribut från entity klassen. Klassen implementerar de metoder som kan behövas vid användandes av ett sådant objekt i spelet. Den har ansvar för att flytta fienden samt se till så att denne kan ta/ger skada vid attacker.

EntityHolder är en klass som det bara ska finnas en av. Ansvarsområdet som åligger klassen är att hålla listor med objekt som finns i modellen.

MainHandler klassen är en chain of responsibility. Detta innebär att beroende på request som skickas från specifik klass kommer en kedja att skapas i handlers successor vilka är:

- GoldHandler: som ser till att uppdatera guldet när objektet som skickar en request till Mainhandler ber om specifikt enumet 'GOLD'.
- PointHandler: som ser till att uppdatera poängen när objektet som skickar en request till MainHandler ber om specifika enumet 'POINT' och är även näst på tur efter GoldHandlern.

Healthbar klassen ansvarar över healthbaren för en entity, och positionerar sig utefter en entity och uppdaterar även dess entities hp i healthbaren.

Player klassen är till för att ha ansvar över spelaren som likt enemy ärver attribut från entity. Klassen i sig ansvarar för var vart player befinner sig i x och y led, förändringar i HP samt hastigheten som spelaren ska röra sig med. Ytterligare funktionalitet är att kolla ifall spelaren är död, om den har möjlighet att röra på sig/inte röra på sig och att ifall den är död kan den efter en tid återupplivas.

Wave klassen hanterar allt som har med vågen av enemies att göra. Den har ansvar för att skapa ett antal enemies i en queue som sedan renderas en efter en i ett satt intervall på tio sekunder.

Projectile klassen beskriver hur en projektil ska fungera såsom positionen, storleken och hur mycket skada objektet ska göra vid kollision. Klassen innehåller metoder som flyttar projektilen i x och y led.

Tower klassen har ansvar över allt som har med basen att göra som dess HP, level, x och y-positioner. Den ansvarar för att uppdatera sina egenskaper när användaren uppgraderar basen, samt ifall användaren väljer att bygga och uppgradera turrets så ansvarar basen för att lägga till en turret och uppgradera specifika turrets.

3.1.2 View-classes

UI delen består av vyer vars syfte är att visa upp delar av modellen för spelaren på skärmen.

3.1.3 Controller-classes

Controllers representerar olika controllers som spelaren använder för att kommunicera med modellen, vilket innefattar vad som ska ske vid tangentbordstryck eller musklick på en view.