System design document for Group 16

Group 16

13th October 2022

Contents

1 Introduction

This is a design document explaining the inner workings of Feyrune, a Pokémon-style RPG, but with the ability to be easily extended with new functionality to differentiate itself.

1.1 Definitions, acronyms, and abbreviations

Feyrune:	The name of the application
DoD:	Definition of Done
Player:	The user controlled element of the game used to interact with the rest of the world.
Tile:	A square of 16x16 pixels.
Tile set:	An image containing multiple different tiles.
Map:	A collection of tiles in a grid.
Libgdx:	A graphical library and game engine
Tiled:	A program for drawing maps from tile sets

2 System architecture

Feyrune was mainly created using libgdx , a java game engine, to render and run the game [1], and Tiled to create and design the maps [2].

When the application is started you as a player is thrown into a world that can be explored. When the application is closed nothing is saved and if you start the application you will be at the same staring point as the first time you started the application.
The application flow can be described by the following game loop:

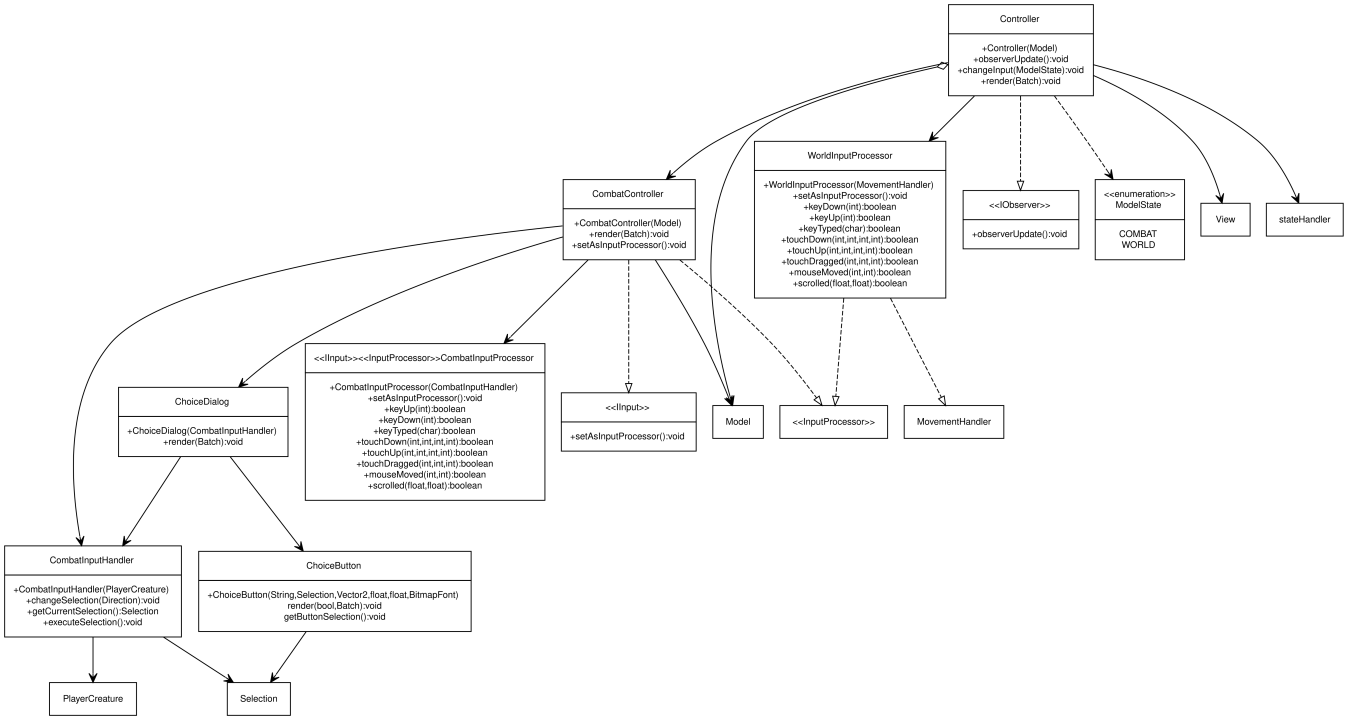
1. The player explores the map.
2. The player encounters an enemy.
3. The player defeats said enemy and gets stronger or is defeated and have to get stronger to defeat said enemy and have to fight easier monsters to get stronger before tackling the stronger monster again.
4. The player continues exploring.

3 System design

3.1 Packages

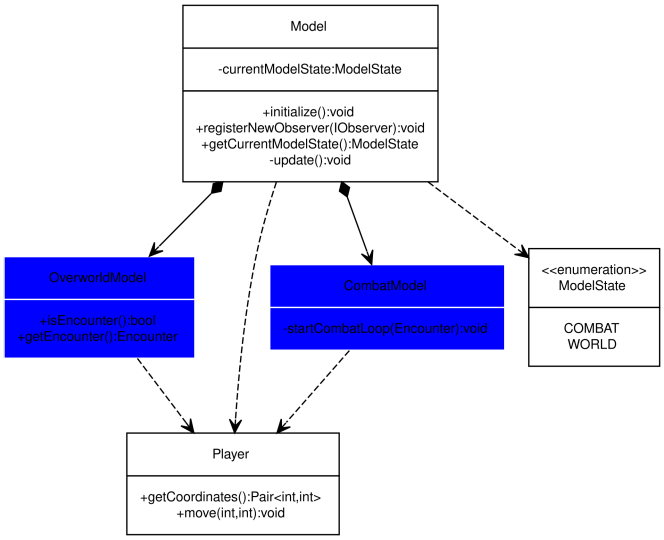
- controller
  - controller.combat
    - controller.combat.ui
  - controller.enums
- interfaces
- model
  - model.combat
    - model.combat.actions
    - model.combat.actions.abilities
    - model.combat.creatures
  - model.creature
  - model.overworld
    - model.overworld.encounter
    - model.overworld.map
  - model.player
- Util
- view
  - view.combat
  - view.components
  - view.overworld
    - view.overworld.textureMap
  - vew.player
  - view.scenes
  - view.utils

3.2 Controller



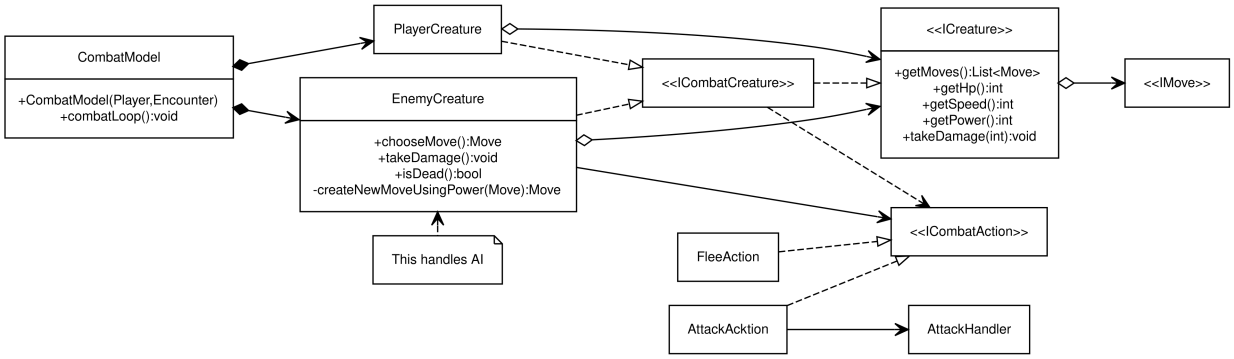
The controller is a little bit diffent from the view as how it mirrors the model in its separation of combat and overworld. It uses an iterface Iinput that extends the libgdx intrface InputProcessor. An input processor has overrides for all button events and you can only have one active at a time. so as the model changes state the corresponding inputprocessor is set as the current inoutprocessor. And then sends some processed information fron inputs to the model. The MapInputProcessor can take movement from WASD and then has a timer that controlles if it can take a new input at this moment. And if that is tru it sends a movment direction to the model. The CombatInputProcessor has four buttons that is renderd in the controller. And the events that it listens for is movment of with button is selected from WASD and if a button is actrivated from ENTER. And it is only of a button is activated that that secection is sent to the model. The controller also has a dependency on the view as it uses the same batch for rendering as the view.

3.3 Model



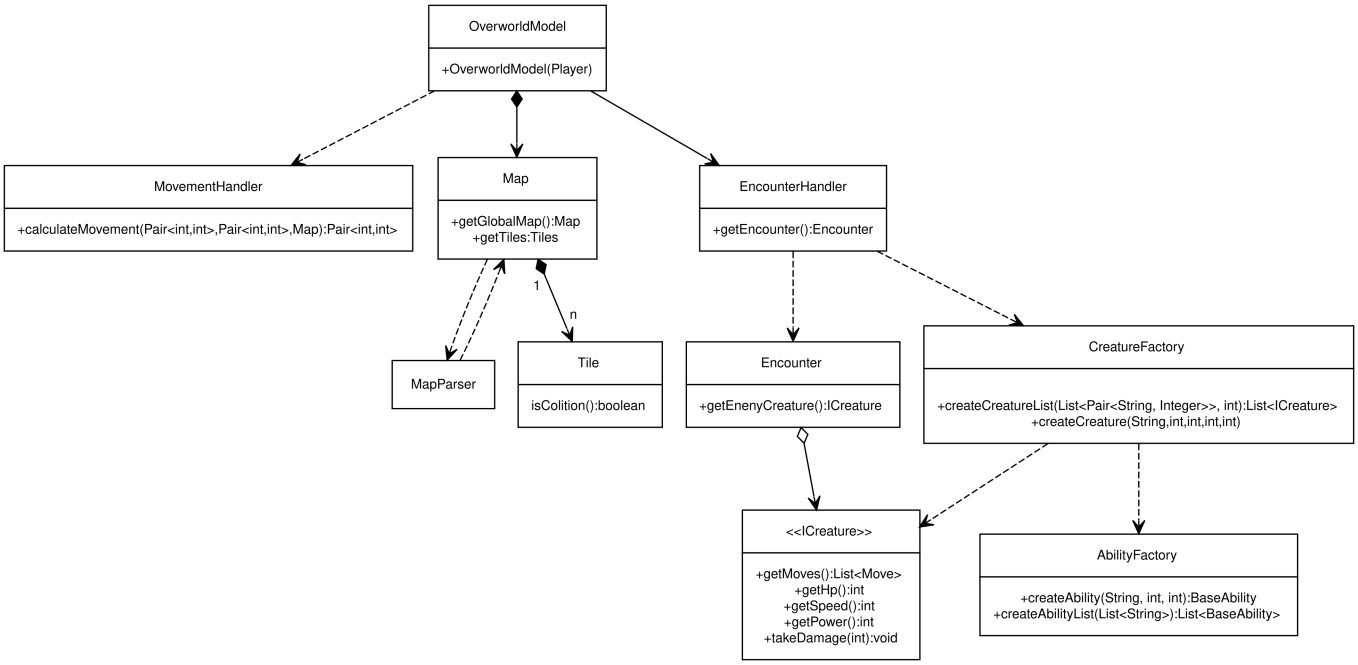
The model is separated in to two separate models, one witch models the overworld and one wich models the combat. And the main model really just controlles witch one of thees that is active and thir relation to each other. in witch both want to have information about the same player. As can be seen in the domain model there are two starting pont seen from the left, one named feyrune with has a direct relation to map and another one named combat. This is the separation we have created wit hthe two different models ans thier relation is practicly just the player and some monster, but as the monster is not constant if it isnt owned by the player the connection there is not as strong.

3.4 CombatModel



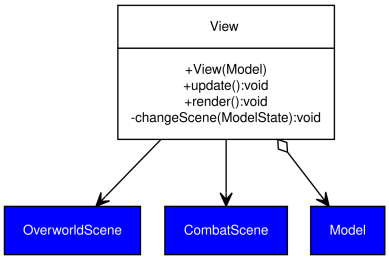
When the combat model is created it only takes the player as a referance and then instanciates new encounters or combats as the main model dictates.

3.5 OverworldModel



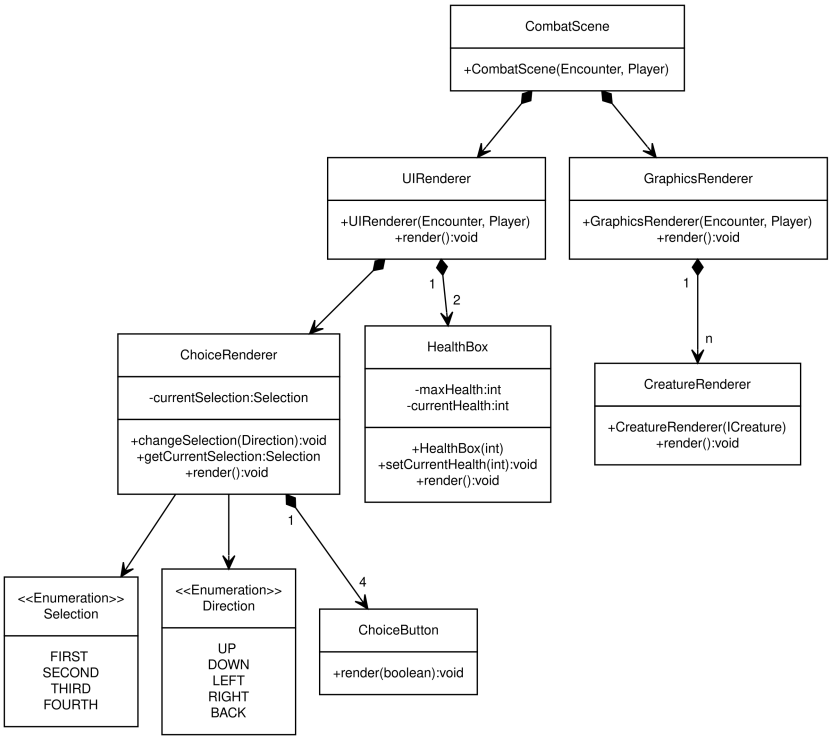
The overworld hadles the relation between the player and the map. It knows where the player is and what attributes a tile can have, if you can move there or if it can create an encounter. As a new encounter is created it is sent to the combatmodel and handled there.

3.6 View



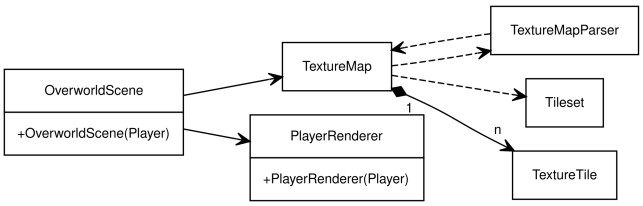
The view morrors the model in that it is separated into two different parts, overworld and combat. And mirrors the model in what tees two are responsible for. It also has a dependencie to the mnodel as it reads most of its data from there.

3.7 CombatScene



As the model state is changed from overworld to combat it the combat scene reads the same information about the encounter from the model. It then draws all relevant information that is needed in the combat, health and witch monsters are in combat. It does not draw any interactable UI elelemnts. for that is the controllers responsibility.

3.8 OverworldScene



The overworld scene is responlisble for drawing the map, it does not take the data for this from the model directly but reads from the same file that the model reads the map but extract only the graphic component. It is responsible for drawing and animating the player at the correct position and also moving the camera accordingly.

3.9 Design Patterns

Throughout the code, Group 16 has tried to implement design patterns wherever feasible. Examples of design patterns used are:

Model-View-Controller

The entire project is structured as a classic MVC pattern. The model exists in its own vacuum without any references to ether the view or the controller. The view has a reference to the model and so does the

controller. The controller also has a referance to to the view, this is to implement the sprite Batch thet is used to draw thing on the screen. This is so the controller can render its own buttons.

Factory pattern

We use a factory to create monsters, their abilities, and encounters, this is to make it easier to exchange e.g. our monster class, BaseCreature with a new one, if someone wants to extend the code with specialized monsters that maybe are able to have special abilities or different types.

State pattern (hopefully)

Observer pattern

Even if the code is running continuously and is polling, an observer is implemented in most places where we don't expect a change every frame.

Singleton

wut?

Façade pattern

wut?

4 Persistent data management

No data is stored and all images are stored using tileset .png files in the assets folder. The maps are stored in a similar vein by using .tmx files that are also stored in the assets folder. The .tmx files contain *all* relevant data for the given map.

5 Quality

5.1 Application quality

To ensure the application is of great quality, every update of it goes through several levels of testing, beginning with a personal review from the code writer. When the writer feels their contribution is good enough, the code is tested using JUnit[3] to make sure it functions logically correct, even in common corner cases.

After the JUnit testing, a pull request is created, forcing at least one other contributor to analyze it and test the code themselves before it being added to the source code.

5.2 Access control and security

6 References

References

[1] libGDX, *Libgdx*, 10th Oct. 2022. [Online]. Available: `https://libgdx.com/`.

[2] T. Lindeijer. "Tiled." (10th Oct. 2022), [Online]. Available: `https://www.mapeditor.org/`.

[3] T. J. Team, *Junit 5*, 13th Oct. 2022. [Online]. Available: `https://junit.org/junit5/`.