System design document for MonOOPPoly

Authors:
Hampus Rhedin Stam
Jon Emilsson
Vilhelm Hedquist
William Johnston

October 10, 2021
version 1.0

1 Introduction

The application is a game that is inspired by Monopoly and which tiles are named after places around Chalmers campus. This document explains the system design for the document.

1.1 Definitions, acronyms, and abbreviations

- MVC, Model-view-controller, is a structural design pattern used to divide and organise the code into three sections: Model, View and Controller.
- UML, Unified Modelling Language, is used to visually represent the structure of a program.

2 System architecture

Our application uses the classical MVC pattern for top level architecture. The group felt most familiar with this pattern and it's use felt perfect when dealing with very clear visual models separated from the model and a controller to deal with inputs.

The model in our application is responsible for all calculations, like for example if things are purchased or sold. It's also responsible for keeping track of the game-state, who's turn it is, if someone is bankrupt and make sure special events take place. The View is responsible for all the visual elements. However as scenebuilder has been used, a lot of the graphical components are located under a resource folder. Most of the dynamic changes should happen in the view class, although this is not fully implemented yet. The controller handles user input, it sends the information to the model which acts accordingly.

The flow of the game, simply starts with the game initialising the view and controller, there isn't much model behaviour in the first two scenes. It stores some visual info, like how many players there are and their behaviour. When the final start game button is pressed Game is created and the model begins to work. Now all three components need to work together to achieve a playable game.

3 System design

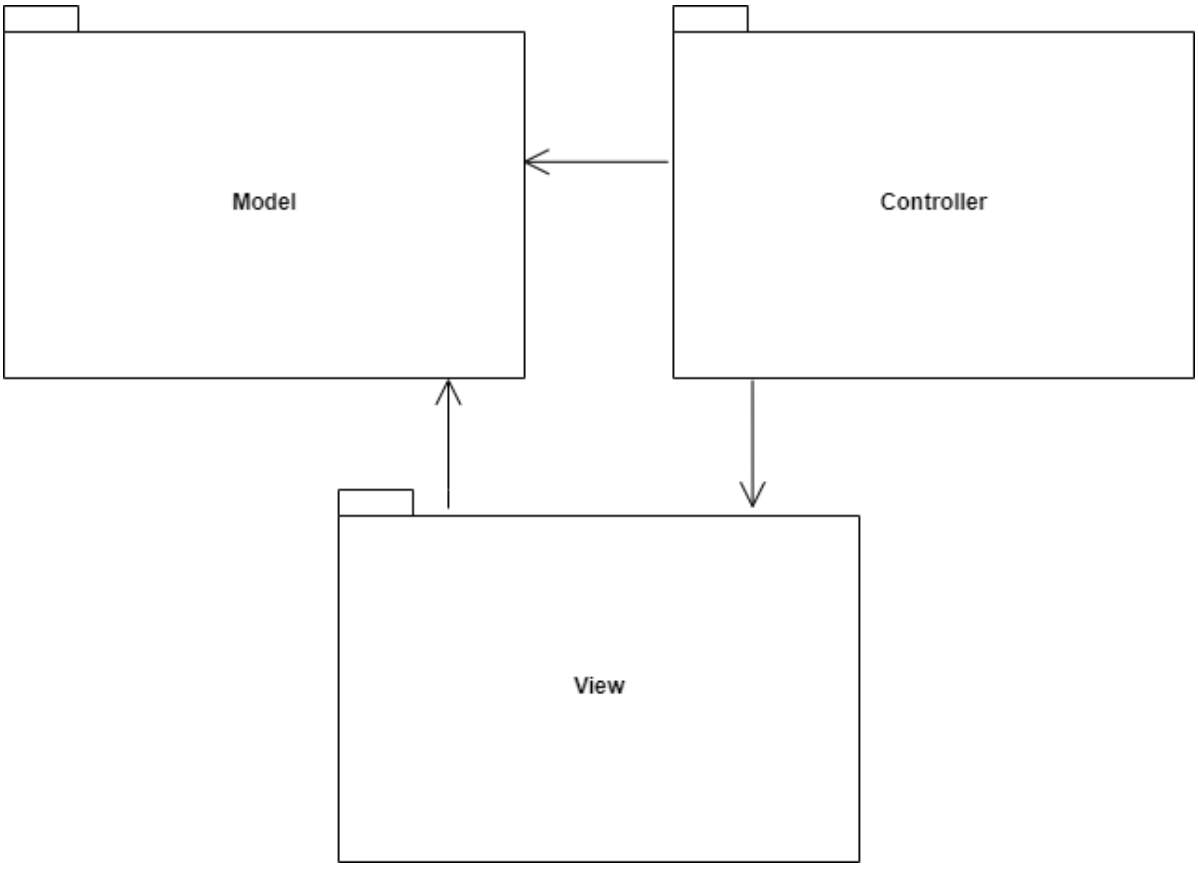

Figure 1: Package-diagram

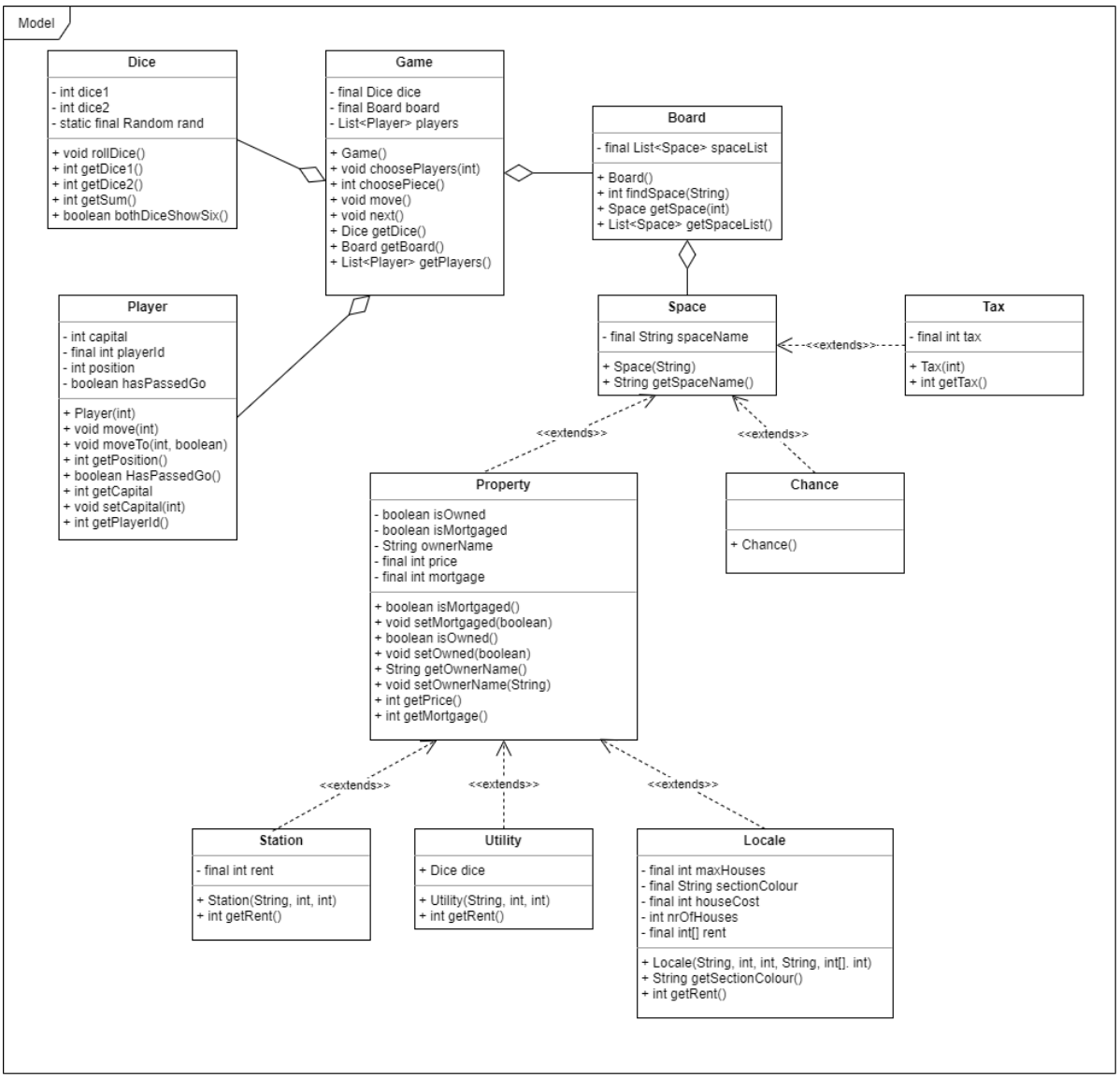

Figure 2: Model-diagram

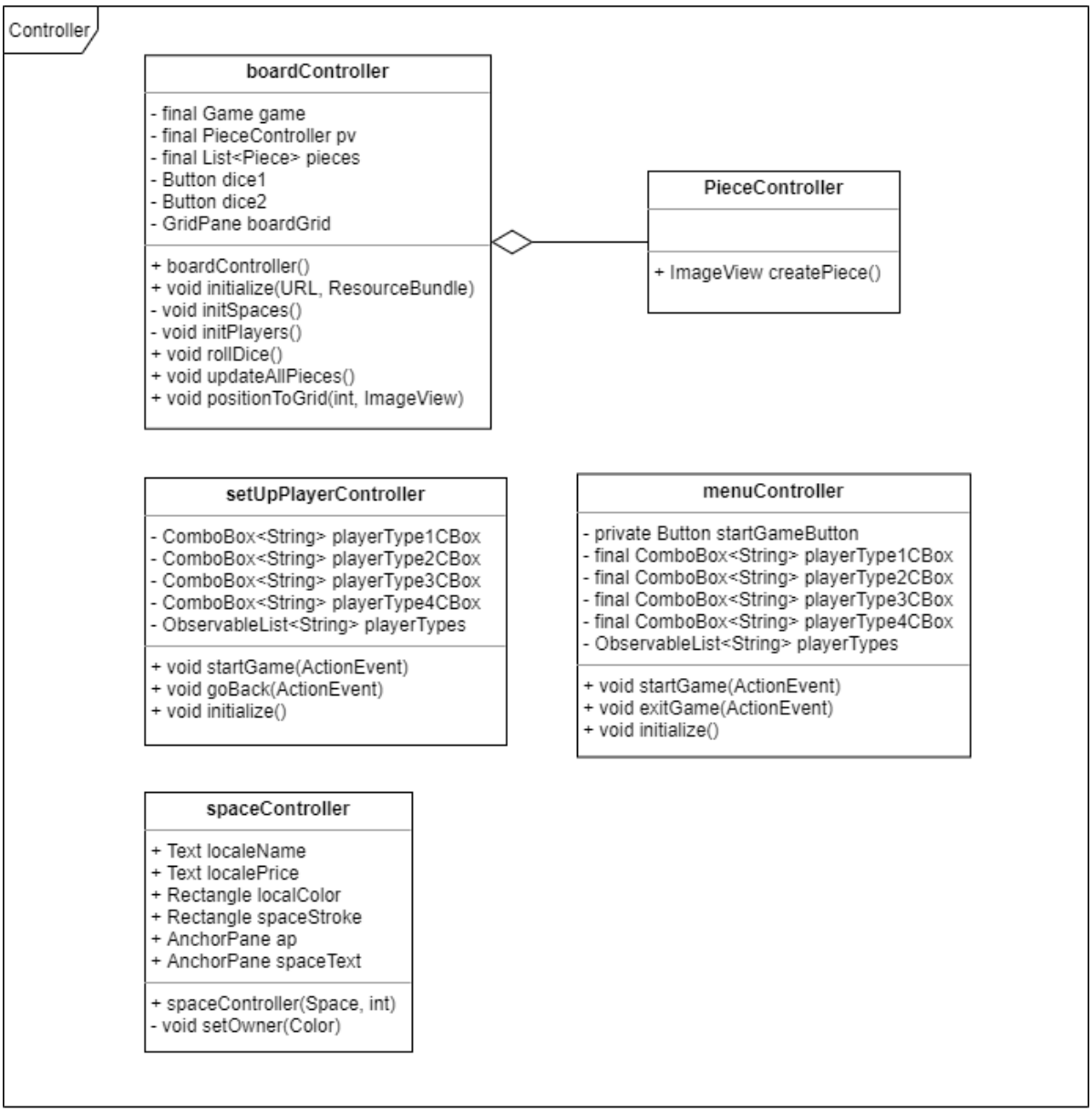

Figure 3: Controller-diagram

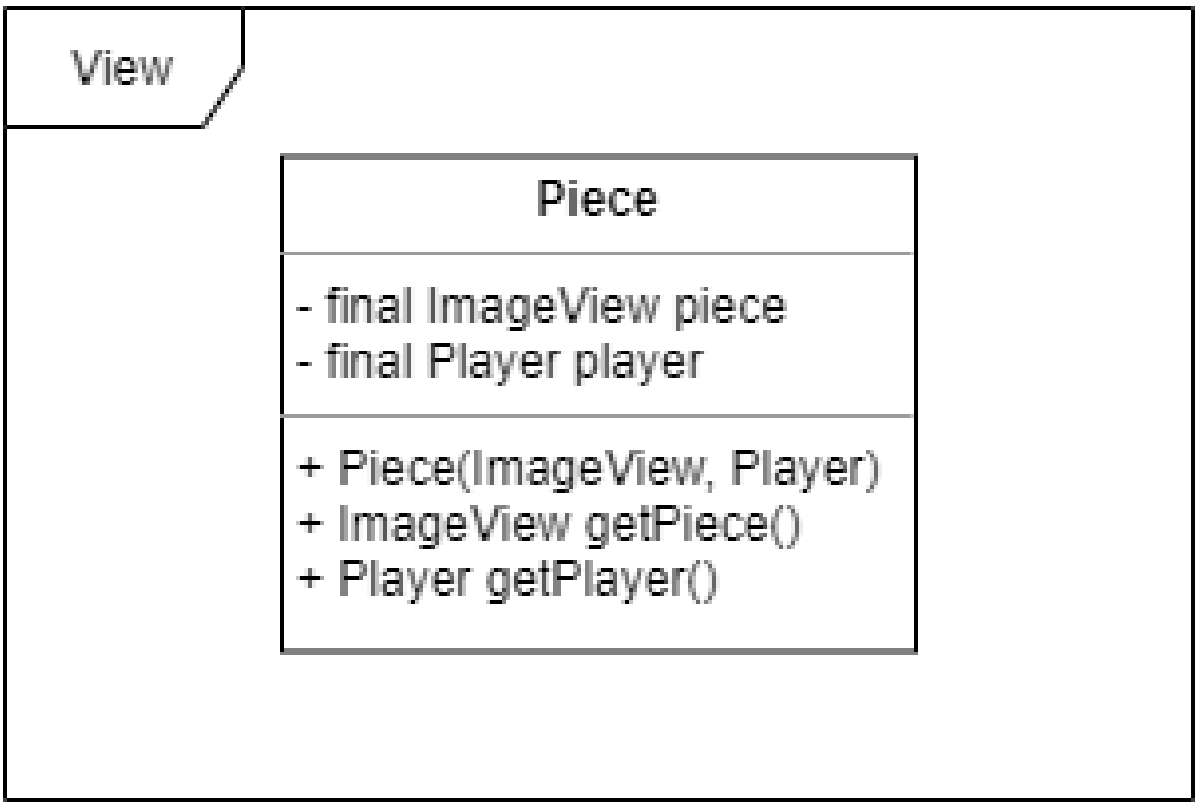

Figure 4: View-diagram

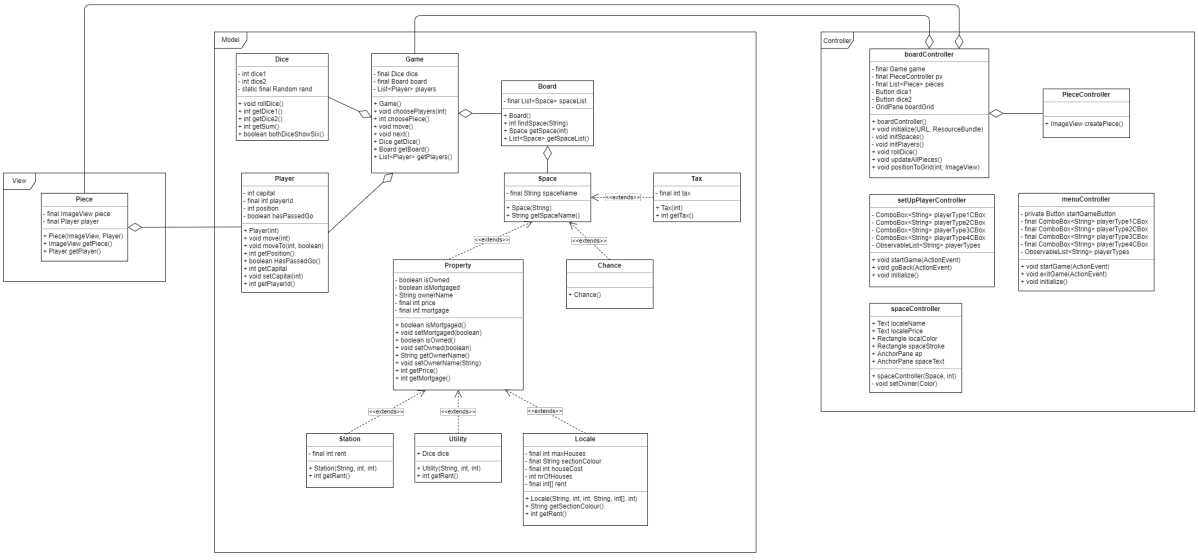

Figure 5: UML-diagram

4 Persistent data management

MonOOPPoly has no user profiles or other persistent data, the only real resources used are small icons used to represent the players which are stored in a ”resources” folder in the src-directory.

5 Quality

To obtain a high quality of the application it is necessary to test the program and check that the different classes and methods act like they are supposed to do. The tests of MonOOPPoly can be found the in the test-directory inside of the src-directory. The tests are done with JUnit by asserting for different requirements. For example, there is a test for the dices and pieces to check that the pieces move the right amount of steps following the dices. This is done by asserting that the sum of the dice and the steps moved are the same. If it's true, then the test has passed. There are no known big issues for now.

- Describe how you test your application and where to find these tests. If applicable, give a link to your continuous integration.
- List all known issues.
- Run analytical tools on your software and show the results. Use for example:
  - Dependencies: STAN or similar.
  - Quality tool reports, like PMD.

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.

5.1 Access control and security

MonOOPPoly uses no login or similar system, there is no need for security or different user roles.

6 References

List all references to external tools, platforms, libraries, papers, etc. The purpose is that the reader can find additional information quickly and use this to understand how your application works.

JUnit.