Report Doodle jump Project

First I want to briefly discuss my class hierarchy. I stayed with the structure roughly described in the assignment, because this was a good start to make my design flexible. It is very easy to add new entities under the class Entity models (you must of course also implement the correct corresponding view under ‘entity_view’). It is also possible to add more specific entity models without extra effort, such as extra platform variants under the class ‘Platform’.

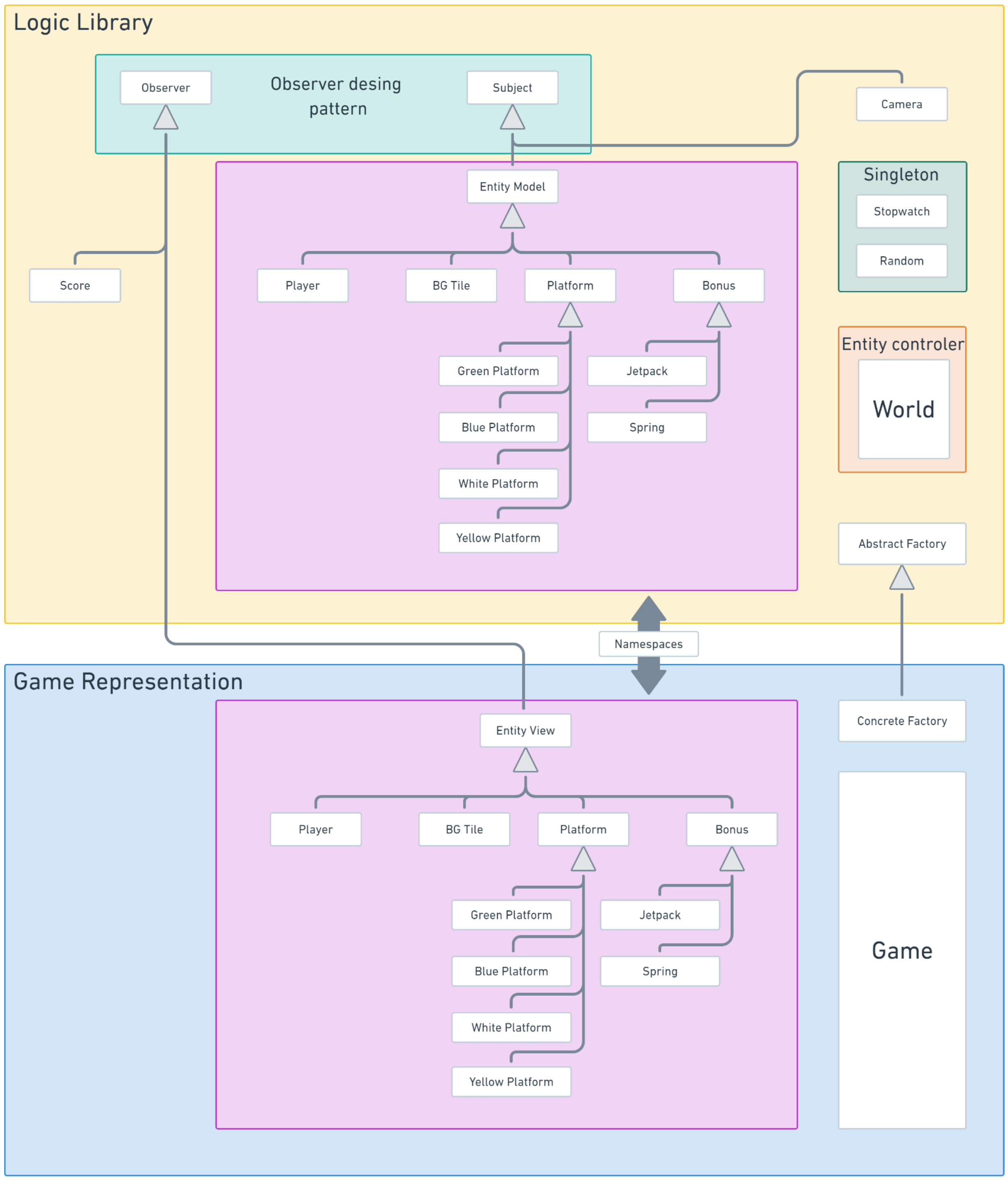
Furthermore there is the need for good and flexible design patterns. The design patterns that I have incorporated into my design are the Model-View-Controller, Observer, Abstract Factory and the Singleton. These are also the obligatory patterns we had to use. The easiest pattern to implement was the singleton. These include the stopwatch and the random class. Therefore, it is not necessary to create a new instance of the stopwatch or random class every time we want to use them, we just have to initiate it once, so every other time we want to use these classes the object already exists and we can work with it.

The next design pattern is the factory or more precisely the abstract factory. This pattern is used to provide a simple interface that the World can use to create new Entities with the appropriate View already attached. To achieve this, I made the abstract factory a pure virtual class, all the methods are specified in the concrete factory. I have also chosen to save the Views in the concrete factory. This may seem strange at first glance, but I thought it would be more convenient to store them in the Concrete factory and then access them in the world (instead of just putting them in the factory temporarily and then moving them into the world). The factory is also very accessible from the world, so if I delete a platform/bonus in the world, I can easily communicate this to the factory.

It is important to explain how I store the views in the concrete factory to justify my choice. The data types I use are maps (for faster retrieval of the values) whose keys are shared pointers to entity models and the values are weak pointers to their view. As you can see I do not give ownership of the views to the factory. The only owners of the views are of course the models, so if a model goes out of scope the view will automatically be deleted too. This is a very convenient feature of the observer design pattern. Since I do have a shared pointer as a key in the factory it looks like it has partial ownership of the models, but this is where the model view controller comes into play. In this pattern, the World can be seen as an Entity Controller. When an entity is deleted in the world. It is not meant to exist anymore, but it is still maintained by the shared pointer in the factory. Nevertheless, the factory will not have the final say. When deleting an entity in the world it will also immediately call a delete method of the factory, so the controller still controls all entities. By implementing the factory this way I still managed to keep the graphical representation and the logic of the game separate. To make this separation clear in the implementation I also used two namespaces, being "Model" and "View", to distinguish between the entity classes.

Finally, I would like to give some more clarification on the observer design pattern. In terms of implementation I have not really deviated from the theory and everything has been done in a proper fashion. My subjects (e.g. the Models) keep a list of observers (e.g. the Views) and they will be updated in the event of an important change of subjects. I keep the list because there is a possibility to have several observers, which is the case for the player. It is observed by the score and the player view. There is also the case in which there are observers who observe different subjects, as implemented for the score. It observes the camera to update the score and the player accordingly, because it should know what platform it has been on in order to calculate the correct score.

To conclude this report, I made a diagram of the hierarchy of my project. I took the names I used in the implementation to assist in understanding my explanations/ clarifications of this project.


Logic Library
Observer
Observer desing pattern
Subject
Camera
Entity Model
Singleton
Stopwatch
Random
Score
Player
BG Tile
Platform
Bonus
Green Platform
Jetpack
Blue Platform
Spring
White Platform
Yellow Platform
Entity controler
World
Abstract Factory
Namespaces
Game Representation
Entity View
Concrete Factory
Player
BG Tile
Platform
Bonus
Green Platform
Jetpack
Blue Platform
Spring
White Platform
Yellow Platform
Game