Report: Project Doodlejump

This report will be a short overview of my game design choices.

Model-View-Controller

This pattern is used in order to clarify model separation between game-state, graphical representation and the logic of the game. I separated the logic(model) of the game and representation(view) of the game in two different libraries and I use the world class as my controller.

Entity

I will only discuss the entities where I would like to say something about my design choices.

The **Platform** class is a base class for all my platforms, all the different types of platform classes derive from this class. The main reason I use polymorphism, is because that way it’s easy to avoid copying code. I can also save all different types of platforms the game generates in one set this way.

The **Bonus** class is a base class for all my bonuses, in this class I also make use of polymorphism. There can only be one bonus for every platform, that is why I save a bonus as a member in my platform class. I also have a checkCollision function in play, that checks the collision between the player and the bonus. That way I do not have to check for collision in the world or player class.

The **bg_Tile** class forms a background for the game. In order to have these tiles scroll together with the rest of the world. I saved two tiles in an array within the world class. In order to create an illusion for an endless world, whenever a tile is under the window, I place that tile above the window. The two tiles keep moving down with every update.

Observer

The observer pattern is necessary for updating the corresponding view when the model changes.

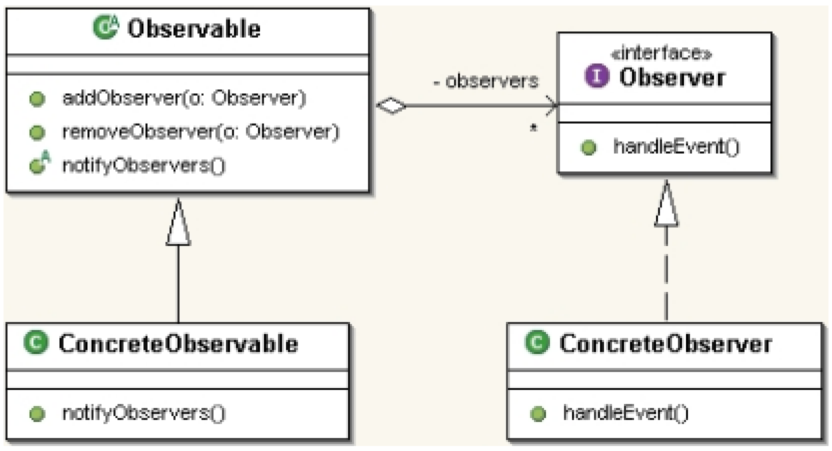

Figure 1 Observer Pattern

In Figure 1 you can see the design of the observer pattern from the course notes. I made use of an observer class and an observable class. My observable class is the base class of my entity class in my logic library. On the other hand, my observer class is the base class of my score class and my entity class in my graphics library.

For every entity model there is one entity view observer. So for every entity model I added the corresponding entity view observer in the list of observers. When an entity model has to be updated it will always call the function "notifyobservers", this function will call the "handleEvent" function for every observer in its list of observers.

Score

I only added the score observer to the vertical, horizontal and static platform model and the player model. I did not add the score observer to the bonusses because the player model will tell the score observer if there is collision with a bonus object.

I save the high score in a txt file. Every time the game ends I compare the score with the high score and edit the txt file if needed.

Abstract &Concrete Factory

The abstract factory is an interface of virtual functions that create observers. This interface is implemented in the derived class of the abstract factory class, namely the concrete factory class. The concrete factory class contains overridden functions that create and return SFML entities. These SFML entities are observers. The Game class provides a pointer to this concrete factory to the World.

World

The world class holds almost all logic entities. The overall game logic is orchestrated in this class. The creation and destruction of entities is also controlled by this class. The collision control between the player and platform class also takes place in this class.

When the constructor of world is called, the abstract factory gets initialized with a concrete factory. All of the observers that are members in the world class will also be initialized except the player observer. Because the player observer is only needed by the player model, the player observer does not need to be saved in the world class instead it is just saved in the list of observers of the player model.

By storing all observers except player in the world class, I can give a certain entity model who needs a certain observer that same observer as another model of that type. This means that for example two different static platform models will have the same static platform observer.