Ethan Behar, Pratyush Duklan, Snehal Patare
10/18/2021
Professor Rawlins
P532: Team 2 Assignment 5

Architecture Description

This GameMaker application attempts to use the MVC Pattern. We say attempts because the first group did not properly build the application with the MVC Pattern. Additionally, we did not set correcting these mistakes as a high priority. The program was functional, so we focused on building games instead of correcting the architecture. We never got far enough in our tasks to start correcting the previous group's mistakes. Additionally, correcting these mistakes is no easy task. The biggest foul the previous group made was making the model data observers of the view. This allows the view to change and the model to directly accept those changes. This flow excludes the use of a controller which is against the MVC paradigm.

Beyond that, majority of the program was set up in a sane way that made implementing new code to accomplish game building fairly simple. The UI did a great job at leveraging JavaFX and its use of the Composite Pattern so any additional UI that was needed was added in with ease. The same can be said about saving and loading since it is using the Jackson library which leverages the Composite Pattern. Beyond saving and loading a file when the game is previewed by a user the application loads a copy of it into the game demo scene. This ensures the model data is not accidentally modified by the game running.

The architecture does a great job at implementing the strategy for executing game logic. There is a GeneralStrategy interface used for key and time events and a CollisionStrategy interface that is used for collision events. There is also a way for a Strategy to use both the GeneralStrategy and CollisionStrategy by simply implementing both interfaces. This dual implementation usually involves the CollisionStrategy method calling the GeneralStrategy method for reusability.

Lastly, the Observer Pattern was used, as mentioned, in the UI as well as used to update the sprites during gameplay. The sprites are observers of the game loop and receive updates for each tick. Furthermore, every sprite listens for key events as the default behavior. When a sprite recognizes a key was pressed that is associated with one of its events, the sprite will trigger that event and thus triggering the associated game logic strategy. A similar process is done for collision events. The main difference is the class that checks for collisions will inform the two sprites when they hit. So, instead of listening for being hit they are told when they are hit or did hit another sprite.