Ethan Behar, Maazin Jawad, Saurabh Gulati
9/19/2021
Professor Rawlins
P532: Group 4 Assignment 5

Architecture Description

The GameMakerApplication architecture is heavily influenced by the MVC Design Pattern. The architecture has all three standard actors: Model, View, and Controller. The View handles the creation of the U.I., layout of the U.I., and user input. Since we are leveraging JavaFX's UI library, this inherently uses the Composite Pattern. Additionally, it handles packaging the input data and sending it to the Controller. The Controller handles the calls and the data the View sends it. The Controller unpacks the input data and transforms it into a digestible form for the Model. The Controller communicates with the Model via Commands and the CommandInvoker. Each task the Controller tells the Model to perform is captured in a Command. Then the CommandInvoker and Commands call the Model's various methods to update the Model's data.

The Model's data consists of background, sprite, event, and action information. The simplest is the background data which is a color and an audio file path(for the game's background track). Next is the Sprite. Each Sprite contains information for a label, id, position, velocity, dimension, color, visibility, and display text. Due to the fact that each Sprite can have Events and Actions added dynamically, the Sprites themselves were designed to be as "dumb" as possible. That is, they contain no game logic. Events contain a Sprite reference, an Action reference, and specific information that triggers the event. For Time Events this is an interval. For Key Code Events this is a pressed key. For Mouse Events this is the pressed mouse button. Lastly, for Collision Events, this is the type of collision it is. Actions can have various data. It is completely dependent on what the Action is and what information is needed to perform the Action.

Since Sprites have no game logic, Events and Actions are the main driving force behind any created game. Each type of Event has a handler class that checks for when an Event might occur and if its triggering condition is met, the handler tells the event to execute its action. These handler classes are observers of the game loop, which is an observable. The game engine acts as a notifier to these handlers. Each time the game is played and then stopped; we utilize the Memento Pattern to restore the Sprite to its pre-play state.

Saving and loading is accomplished via the Composite Pattern. Saving and loading starts with the Model and trickles down to each type of data that was explained earlier. The functionality reaches the child nodes which performs its own saving or loading. Then the child returns to the parent allowing the parent to finish its own saving or loading. Eventually it reaches back to the Model.