Advanced Programming: Gradius (2nd session)

Christ-Offert Nsana Matundu

S0164117

2017-2018

Introduction

In this report I will be explaining the design and general decisions I made when making this project. The general idea was to approach the development with a modular design consisting of a Model, View and Controller. The Model to hold the data, logic and rules. The Controller to receive input to manipulate that data of the Model. The View to get a visual representation of the information contained by the Model. There are also other components that offer a helping hand to these big modules, these are grouped under Utilities. And finally the Game makes use of all these subsystems to implement an interactive game. Now let's look at these components in more detail.

Model:

The Model is the center for all data and logic in the game. Therefore the Model also manages the entities. Entities are self-contained and are fully functioning objects that are responsible for their own behavior. The Model is only concerned about letting these objects know that they should update themselves or perform certain behavior (like handling collision). The general decision of the Entity class tree is quite simple. There is an abstract base Entity and the next level consists of Friendly, Neutral and Enemy. This distinction in entities is handy for collision detection where Friendly on Friendly on collision shouldn't be damaging, but Friendly on Enemy should. Entities that derive from these base are entities with their own movement behavior.

Since the Model is concerned with managing all the data, it also concerned with creation / deletion of the entities. The Model handles the deletion of the entities by removing / adding entities to its entities vector. The Model delegates the construction of its entites to its EntityFactor object.

Entities still need a way to contact the Model to let certain changes happen (spawing of bullets, notifying its death et cetera) and this communication is handled by using events. There is a EventQueue that is accessible by all the components of the Model module and the Model can access this queue and execute all these events.

View:

The View is concerned with the visual representation of the state of the Model. The View is constantly being provided with information from the Model by using the observer pattern. The entities of the Model tell the View (which acts observer) crucial information like their identy, location et cetera. The view then uses this information to generate a visual representation. The View is helped by the Transformation class that converts the 4x3 game-coordinates to actual screen pixels to draw.

Controller:

The Controller handles input events and uses those events to manipulate the Model accordingly. This is done by accessing the public API of the Model. The Controller is not concerned with underlying implementations and just acts based on input.

Utilities:

The utilities classes used in this project are all designed using the Singleton Pattern. This module contains the classes Stopwatch and Transformation. Stopwatch is obviously very important to the game as a whole to keep a consistent framerate. The Transformation class has already been mentioned when discussing the View. These helper classes play an important role in making everything work.

Game:

The Game makes the actual shooter game possible by combining all these components together. The Game is responsible for the creation and management of these components and performs the main game loop. Other typical functionality of games like the menu with options and the scoreboard are implemented in this module.

Extensions:

Extra things I implemented that weren't specified in the assignment.

-Animations:

The game makes use of animated sprites. When the player or the enemies die there is also an explosion animation.

-Rectangular collision detection:

I decided to make use of rectangular collision detection instead of circle on circle collision, because it made more sense for more generally rectangular sprites.

-Pause option:

The game can be paused, to see which control does this, please read the README

-Fullscreen:

Due to issues with the Unity sidebar used in the Ubuntu OS, my first idea to implement complete borderless fullscreen did not work out. I decided to allow the game to maximize within the boundaries that exclude the unity sidebar, but maintain the correct 4x3 aspect ratio on any screen (by padding with black bars on the sides or at the top and bottom).