Report for Flappy Bird Game (Object Oriented Programing Java)

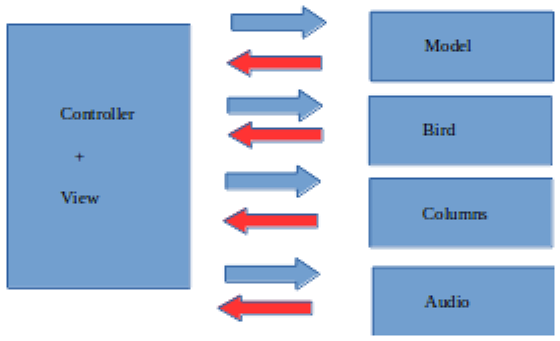
I always wanted to create a Flappy Bird Game: this final assignment gave me the opportunity to play with this idea. For the majority of the game's logic and main idea on how to start implementing such a game, I followed a video tutorial from: http://sjbuzz.net/video/watch/vid01R83xtgbO2o4. Note that no source code was ever given, I had to work my way through this video (which was not very easy to follow), to understand how JavaFX works. JavaFX is not necessarily a gaming library; therefore, much reading on how its functions work and which tools I needed to get the job done was investigated.

The Design of the program.

Since the beginning of the program, I thought that having a Model View Controller design would be optimal, specially when it comes down to applications such as this video game. However, as it will be shown in the next section, implementing this type of design proved to be difficult when using JavaFX and its functions. As a result, I opted for a "hybrid" View-Controller in one class and a Model to hold all of the logic in another class. Of course, other parts of the game such as: the bird, the cloud, the columns, the coins, the audio, are separated in their belonging classes. By following this type of design, I was able to keep true to an Object Oriented approach, despite the fact that JavaFX pushes the developer to follow the JavaFX approach. At times, this program "feels" as if it is repeating itself: for example, in the Model class, there is a function which subtracts the X coordinate of the cloud to return it back to the JavaFX class to help the cloud move in every single frame. Of course, one would say that it would be best to just subtract this integer in the same class where the cloud is being displayed; but, by having a separate class called FlappyBirdModel, I am trying to emphasize the idea that all operations which involve some kind of logic should, therefore, take place in the Model class. As such, the Model class contains functions which involve gravity, movement, ticks, scores, levels, and so on.

Problems encountered while programing in JavaFX

One of the biggest and most obvious problems which I encountered while writing Flappy Birds was the design problem. JavaFX, it seems, will always push the developer to develop the JavaFX way. I spent a lot of time researching and finding ways in which I could design this program in an Model View Controller approach; in the end, I could not find a solid answer as it seems people either don't develop with JavaFX in an Model View Controller approach, or JavaFX does not allow for this design to take place. Hence, by merging the View and the Controller, I could start to design the program in an Objected Oriented way. The View-Controller class, which I call FlappyBirdView.java, is where the JavaFX application takes place. This class controls and displays everything in the game: take a look at the diagram below:



As one can see from the diagram above, all the other classes do not know of the existence of each other; they all have to intermediate between the View-Controller class. This was a design choice, as the controller in a normal MVC design is the one in charge of getting and setting information and data for the rest of the classes. This design actually proved to be nice to work with as most of the time I found myself referring back to the Model class to keep things organized, consistent and robust.

Graphics

Implementing the graphics for this game was one of the most challenging, yet most fun and rewarding experience of the whole assignment. At the beginning, I had to realize that even before I would put images into the video game, I had to implement the first logic of the game. That was an object (a bird) had to fall to the ground. For that task I created an Ellipse which would increase its Y axis (getCenterY and setCenterY) every time a key frame came in. KeyFrames work together with the Timeline function in JavaFX: by setting the Timeline.setCycleCount(Animation.INDEFINITE), I was able to create a never ending loop. With this, I had to listen to every KeyFrame (which are set to 20 milliseconds) and then I created a function to be called at every KeyFrame to increase the Y axis of the Ellipse object. This creates the behaviour of an object falling to the ground. There is a listening event inside the Keyframes to pick up any time the user presses the UP key. When the user presses the UP key, then the Ellipse is decreases by -10, thus making the object jump up in the next keyframe.

Next up are the columns: thanks to the video tutorial provided by User Miku, I understood that if I needed a bunch of columns to appear at every keyframe, I had to create them inside an ArrayList<Rectangle>. This allowed me to have a loop inside every KeyFrame to get every Rectangle and to decrease their X axis so that they would appear as if they are moving towards the bird. I found this trick quite clever, and so I implemented in the game. After a Rectangle has left the window, it is therefore destroyed to open a bit of memory.

Then, after that, the last thing to implement was the collision. So I created a function called collision() which would check whether the bird would “intersect” with a column by using the JavaFx functions getBoundsInParent() and intersects; if it did, I pause the game to have a button come up for the user to click on it so that the game can be restarted once more.

That is the basics of the game: an object needs to not intersect certain bounds created by the developer; if it does, the game is over. After that, the majority of the time (which was a great time) was spent splitting up the program into its corresponding classes. Trying to understand the pipeline of JavaFX was very difficult.

Testing

The biggest bulk of testing happens, of course, in the Model class. This is because all of the logic and behaviour of the game takes place inside this class. There are a few other unit tests in other object classes, such as the Column class, to make sure that things are going in the right direction. For the way the game would look and feel, I used the “sight” testing technique in order to have good feeling of how the game might look once it is compiled.

Extensions

Ideally, if I had a bit more time, I would like to include several other types of items for the bird to pick up along. Every time the bird would pick one of these different items, a different sort of sound would be displayed. Items such as cake, bananas, etc, each with their own particular sound. I would also like to introduce changing the background of the game along the levels; that is, every time the bird moves on to a new level, the back could either change colour or display a different colour. Lastly, I would love to (which might be one of my side projects) to implement this video game into an android app to share with friends and colleagues.