Java - Graphics

Overview

Frogger is an 1981 arcade game where the aim is to guide the frog over the road safely to the other side without being hit by the obstacles that are in the way.

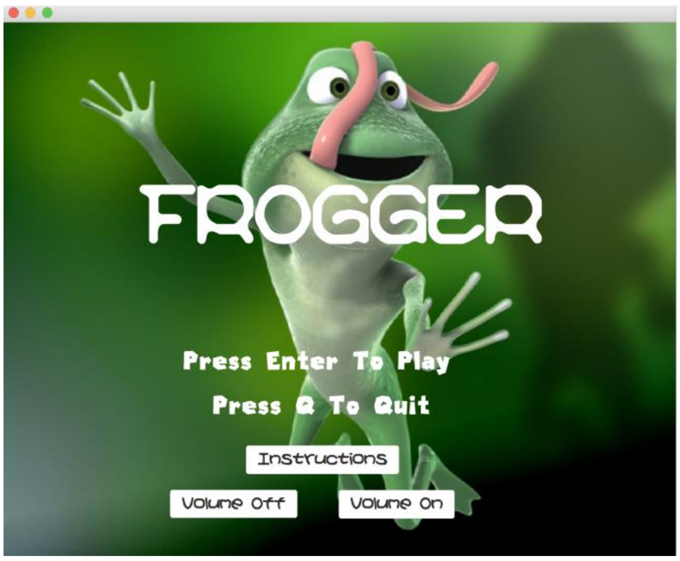
This report highlights the main features of the game, a brief description of the classes and the problems I had while developing the game using JavaFX.

How to play

Javac *.java
Java Frogger 2

Press enter to play, q to quit.
Press instructions button for instructions.
Press volume on/off button to control volume.

Screenshots



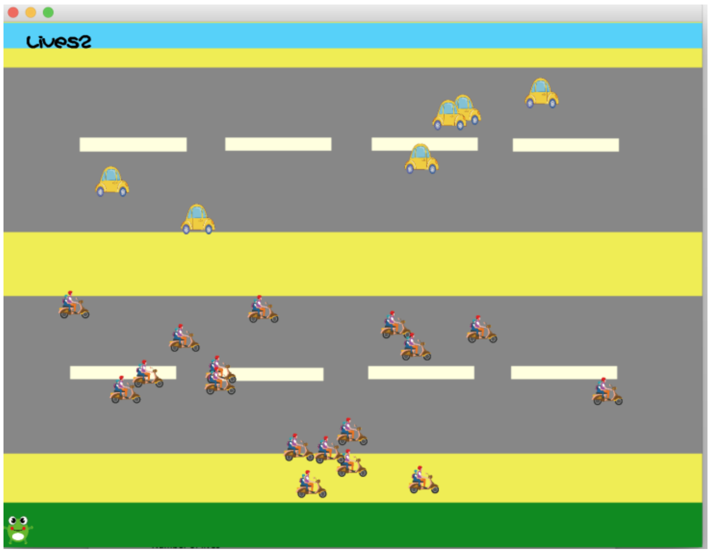


Fig. 1. Welcome Screen

Fig. 2. Main Game

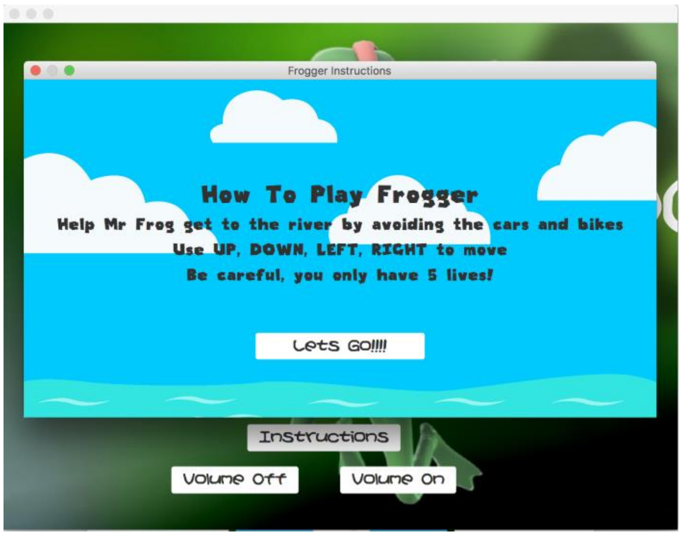

Frogger Instructions
How To Play Frogger
Help Mr Frog get to the river by avoiding the cars and bikes
Use UP, DOWN, LEFT, RIGHT to move
Be careful, you only have 5 lives!
Lets GO!!!!
Instructions
Volume Off
Volume On

Fig. 3. Instructions

<u>Features</u>

Welcome screen

The welcome screen for the game was made in the class "Frog Welcome" and "Frog Welcome Screen". I created a new pane within these classes and called them within the startGame method in "Frogger 2".

Volume on and off
To reduce the amount of repetitiveness in the code I created a style method for where there are similar features for the buttons.
Added buttons to enable the user to turn the sound on or off. This then continued to the rest of the scenes to ensure silence in the actual game.

Scrolling text
To show extra features on the home screen I included scrolling text of the title. This was done using keyFrame and start/end frame.

Instructions

Pop up
I created a new stage for the popup menu to create and show the instructions. When the button is clicked the instructions appear and explains to the user what needs to be done. A button then closes this and returns the user to the home screen.

Background using CSS
I wanted to try using a CSS file and therefore I added a background image to the instruction pop up using the layout.css file. Here I used a CSS file to add in the background image to the scene. This is to show how you can add background images in different ways, like using a CSS file.

Main Game

Different objects

Used PNGs found online for the bike, car and frog. I made the background in GIMP to imitate grass, crossing the road and getting to the water. To ensure movement across the screen I used getTranslateX and then set a new X location adding on the width of the object. This ensured a smooth transition across the screen. I then randomly created the locations along the Y axis. To ensure the cars were at the top and the bikes were at the bottom I had to work out the specific values of the axis plus the size of the images to ensure that the frog would not be hit when on the grass and in the water.

To win the game the frog had to get to the coordinate 0 on the Y axis.

To ensure that the frog wont get lost of the side of the screen and make sure the user couldn't move the frog off the screen, I had to include an if statement so that the frog can only be controlled at certain dimensions.

Music

It was easy to add in music to different parts of the game using the FrogMusic class. I managed to use turn volume on and off to ensure the music was played at the correct times within the frogger2 class. I created functions within the FrogMusic class to turn on and off the volume when the button is clicked on the main screen.

I acquired the music from https://downloads.khinsider.com/game-soundtracks/album/frogger-psx-gamerip.

Set Background
Used background Image to set the background. I made the background image using GIMP.

Collision check
To check whether the frog has hit the car/bike I used 'getBoundsInParent'. This causes a slight glitch in the fact that the bounds of the image are different to the bounds of the actual car/bike, although this is minimal and therefore think that the game still functions correctly.

Number of lives
Set the number of lives to 5. Each time the frog hits the objects, the lives go down by 1 and the frog is reset to its original position. Once the lives = to 0 then the game is over.

Problems

Firstly I created the game within one class. This was to work out the initial logic behind the game. Upon finishing this, I realised that I had to make a major design change to ensure there are minimal cyclic dependencies and I need to separate out the logic from the graphics behind the game.

The issue I had with this is that the graphics had to be added to the pane that was in the frogMake class. To overcome this I created a new class where the design of the graphics was made and then added the objects to the pane in a different class.

I had a major issue with creating the lives label. Each time the life decremented it would be placed on top of the original life. This was because instead of updating the original label I was creating a completely new label. To overcome this I created the 'die' method which was called each time the frog died and then this would overcome the issue.

Another issue I had was trying to get the bike and car to be on separate sides of the screen. This took a bit of working out to ensure that they were correctly located:

```
    car.setTranslateY((int)(Math.random() * ((260 - 40) + 1) + 40));
   bike.setTranslateY((int)(Math.random() * ((519 - 300) + 1) + 300));
```

As you can see here I had to ensure that the sizes were considering the size of the frog and themselves to make sure that they didn't overlap. This meant there was a safe space for the frog at the bottom of the screen.

I had a problem with making sure the music was on and off at the correct times. I had to include pause/stop within the Frogger2 class to ensure that at different scene changes the correct music was being played.

I also included the play of the music within the main game within the Parent of FrogMake. This was because I wanted the music to stop when the player won or lost the game. I struggled with having the music start and stop at different stages alongside getting the volume to be off and on the entire time.

Testing

It was hard to test the graphics apart from seeing what physically appeared on the screen. The only logic that was testable was the numLives where I tested to see if the number of lives went down or if the correct number of lives were received.

Classes

A brief summary of what the different classes contain to control the game.

Frogger2
Controls the scene.

FroggerWelcome
Creates the new scene for the welcome page.

FroggerWelcomeScreen
Creates the text for the welcome screen. Includes the scrolling animation.

FrogMusic
Contains all the files for the music and also how to turn the music on and off.

FrogMake
Has all the logic behind the cars and how they move. Also controls whether the frog hits the car etc etc.

FrogChild
Class used to create the car/bike and frog.

FrogBehaviour
Used to add the buttons at the end of the level.

FrogInstructions
Used to create the popup window for the instructions.

Future work

In the future, I would be able to add different levels. This would involve changing the speed of the objects as well as adding more objects. Potentially having a level where the frog could only jump on the lily pads to cross the river.

You could also add a high score where there are collectable objects to gain points.

Overall, I really enjoyed using Javafx to create the game and I think the outcome of the game looks stylish and it is fun to play.