Balloon Burst

Nam, Jun Hyun
Ong, Zi Liang

Project Overview

Balloon burst as the name suggests is a game that the user will have to prevent the balloon from bursting. The balloon starts bouncing from the middle to the right of the screen when the user is ready. The user will hit the space bar key to keep the balloon in the air to prevent it from hitting randomly positioned spikes on the right side of the screen. Upon hitting the right wall, the balloon will bounce to the left side of the screen and it continues.

Results

The game consists with a window that has spikes at the top and the bottom of the screen. These spikes are drawn onto a black wall. The spikes at the top and the bottom of the wall are permanent and positioned orderly.

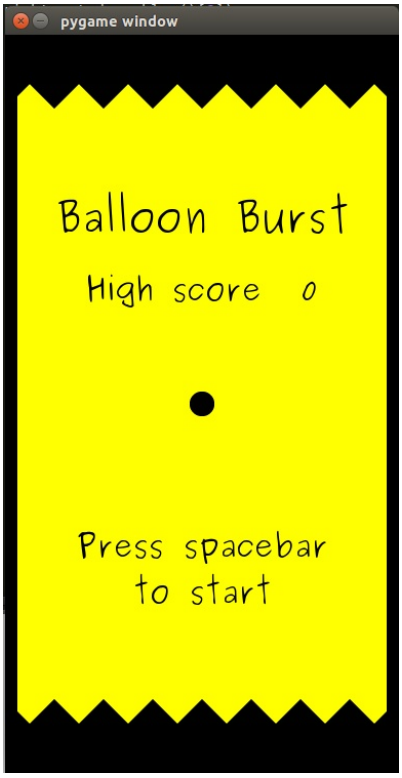

Figure 1 illustrates the start of the game

The randomly generated left and right spikes are also drawn onto the wall and the user has to get the balloon to bounce onto the empty spaces on the wall between the spikes. The amount of

bounces the user gets on the left and right wall between the spikes determine the score which will be displayed in the center of the screen.

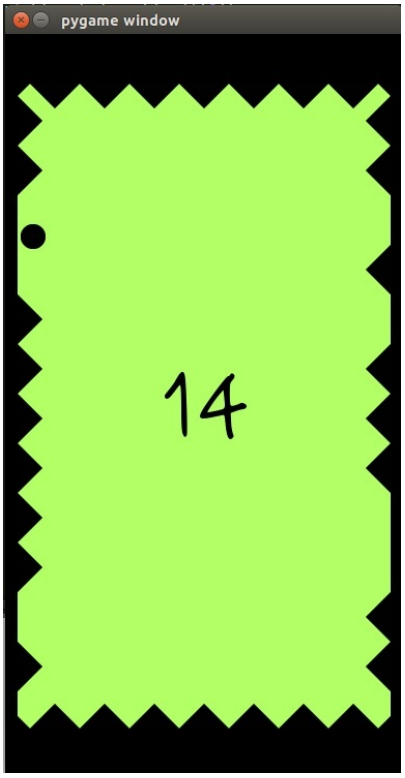

Figure 2 illustrates an instance where the ball bounces from the left side of the wall

The color of the back background changes when the score hits a certain number. For example if the score is from 0 – 9, 10 – 19, 20 – 29, 30 – 39, 40 – 49, over 50 the background color will be yellow, light green, light blue, blue, purple and orange respectively.

However, if the user failed to keep the balloon bouncing on the wall between these randomly generated spikes, and if the balloon hit the spikes, the balloon will burst and picture of birds will be shown flying away. In addition, the background also switches to red as it is a universal color for stop.

Clink on the link below to watch video of this game.

http://youtu.be/F_xe5aERlC0

Figure 3 illustrates an instance where the balloon hits a spike and it burst

Implementation

Because this game deals with collision between balloon and spikes or walls, the 3 main objects in this game are Balloon(), Wall(), and Spike(). However, we didn't start to implement code from implementing basic objects like Balloon(), Wall(), and Spike(). Instead of that, we thought about how does the entire game would be run. So we decided to use a design pattern called Model-View-Controller as we learned in the class. We chose this design pattern since our game is quite similar with flappy bird. Controller part would receive input from user, and model part would deal with all other objects in the game. Finally, view part would draw all the drawable objects.

After we decided a big picture of whole program, we had to decide which class do we need to implement the game. First, we had to display some texts include score. So we created class Texts() that contains all of the texts and some methods deal with texts. Also we create class Score() separately with Texts() since we want to add some features related to score.

The next thing we've done is creating main objects, balloon, walls, and spikes. Since there were two walls at each side, we created class Wall() to deal with single wall, and class Walls() to deal with all of the walls. Spike() class deals with spikes on each sides, and Spikes() class deals with both sides. For example, Spike() class generates spikes randomly, and Spikes() class has Spike() as a parameter and deal with collision detection. Last but not least, we implemented Balloon

class (). Balloon() class has similar attributes with Bird() class in flappy bird. We had to add some other attributes to make balloon burst and birds flying out from the balloon.

There were two important features in this game. The first one is generating spikes. To generate spikes randomly, we used list and method random.choice(). The list named spikes is a list contains 0 and 1. 0 represents no spike, and 1 represents spike. So we used method random.choice() to generate a list of 0, 1 every time when the balloon hit the wall. To control difficulty, we controlled the probability that 0 or 1 would be chosen. The probability that 1 would be chosen was 0.4 if the score is 0-9, 0.5 if the score is 10-19, and 0.6 if the score is more or equal to 20.

The other important thing was collision detection. Since our balloon is circular shape, and spikes are diamond, we couldn't use the same method as flappy bird in class. However, there is very useful built-in class in Pygame called Sprite. Sprite class has a method that can extract non-background part from image file, and detect collision with it. So we set our basic drawable class, DrawableSurface() to inherit class pygame.sprite.Sprite to use that useful method.

The UML class diagram is attached in Appendix.

Reflection

The flappy bird example in the class was a great aid to our game. We did not know that Professor Paul was going to cover flappy bird Pygame code in class. It helped us in generating the overall structure of our program in terms of the different class we can use for different categories in the game. We thought that our mini project was pretty appropriately scoped because it consists of almost all the different functions that we have been taught in class. Some of the examples are randomness, class, and inheritance etc.

Class is an important notion that is useful for long coding, and our code made use of the attributes and methods in class to access different variables. We think that it is a good skill for managing long codes in the future. It will be even better if we were more familiar with Pygame so that it would be easier for us to use some of the basic functions.

We met up 3 times in a week to discuss the sharing of workload. We had to do pair program because each class is connected with each other. This will be tough if we were to split by class. There were small issues on getting each other to understand our intentions and ideas but it was resolved very quickly. I think we will still stick to the idea of pair programming because we will understand each other's intentions quickly and rectify the mistake immediately.

Appendix A. the UML class diagram