Maryfrances Umeora

mumeora

Project 4

Lab Times: TR 11:05-12:20

I did not collaborate with anyone on this assignment.

**Rules of the Game (PLEASE READ)**

As you’ve probably already guessed, the player has to keep the bouncy ball bouncing for as long as possible. If the ball hits the bottom of a screen, they lose a life. There are three levels.

So the object of the game is to get as many points as possible in the allotted time. In the first level, the beginning timer is set to 200 seconds (subject to change, please look at variable levelonelimit in code to be sure). If the user keeps the ball bouncing for that long, they get to move to the second level. In the second level, they get less time (150 seconds, but again, subject to change). In level 3, they get even less time, and if they’re still alive by then, they win the game.

Lives are not reset every level so it should be a bit hard to win the game (or maybe I’m just bad at games, it’d be hard for *me*). However, if they lose all three lives in any level, they automatically lose, an ominous voice says GAME OVER and the window closes once they accept their fate.

**The Sound Class**

I added sound effects for extra credit (apparently, they’re worth up to ten points).

I downloaded a bunch of sound effects as wav files from my account on Freesound (it’s a great site, check it out), then attached them to the package on which my code is in. I’ll include these files in my zipped file submission but I don’t know how they show up when you open the file so please make sure that they are in the same package as the rest of the classes.

Anyway, in my sound class, I basically created new AudioClip objects using the sound files and I play them at various times throughout my code.

**The Game Class**

Now this is where all the magic happens. Since I have my ball and paddle as different classes, I use my game class to allow them to interact by created objects of those classes. The game class is also the class I add a frame too (in the main method).

The Game class’ constructor just builds the basic GUI that you see here. It’s not very pretty as the score, lives and timer are represented with simple labels but hey, I fulfilled the requirements and that’s all that matters.

**Methods in the Game Class**

* **Action Performed**

All I do in this class is increment the value of time elapsed and update the time label accordingly. Also, I call my move() class, which I will explain later.

* **Paint**

In my paint method, I call the paint component methods of ball and paddle. Also, if my boolean value disappear isn’t false, I also draw the obstacle.

* **Draw Progress Bar**

I do not use this method. Please ignore it. I just didn’t want to delete it since it works but was too troublesome to fully implement.

* **Move**

This calls the move method of ball and paddle.

* **Level Two**

I use this class to inform the player that they are moving on to level two. At this point, I give them bonus points as per the requirements and reset their lives to 3.

* **Level Three**

Pretty much the same as level two.

* **Game Over**

If this method is called, then an ominous voice chuckles that the player has lost and the system exits. You don’t even get to see your final score because you suck.

* **Win Game**

On the other hand, if the player actually wins, they get a chorus yelling “yayyyyy,” a message informing them of their victory and their final score, and the system exits, because winners don’t have to click the close button by themselves 😎

* **Main**

This is complicated so after I explain what’s going on in my ball and paddle classes, I will explain the main method in full.

**The Ball Class**

Okay, so I lied. Most of the magic happens in the ball class. But the ball needs a game to be in so I make sure to make an object of the Game class.

**Methods in the Ball Class**

* **Paint**

This draws the ball at its starting position.

* **Get Bounds**

This returns the bounds of the ball.

* **Touch**

This is a Boolean value that returns whether or not the bounds of the ball intersects the bounds of the paddle.

* **Move**

This is pretty self-explanatory if you just look at the code. The first three if-statements check if the ball has collided with the frame walls. If the ball collides with the bottom of the screen, then they lose a life. If lives is now equal to zero, game over. If not, it just bounces off.

Also, if touch() is true, i.e. if the ball’s bounds intersects the paddle’s bounds, then the ball bounces off the paddle and the player scores one point. I do not play a sound when this happens. I only play the sound if the ball bounces off the walls.

Finally, something happens with Ching(), which I will explain under Obstacle Class.

**The Paddle Class**

Like the ball class, the paddle needs a game to be in so I make sure to make an object of the Game class in the paddle class.

**Methods in the Paddle Class**

* **Paint**

This draws the paddle at its starting position.

* **Get Bounds**

This returns the bounds of the paddle.

* **Key Pressed**

Depending on whether the key pressed is the left or right arrow, the velocity of x is set to positive or negative 1. You might notice that none of the other key listener methods are present in this class. That is because paddle does not extend key listener: Game does, and I take care of all of that ish in Game’s constructor.

* **Move**

As long as the paddle isn’t moving out of the screen, the x value of the paddle is incremented by the velocity determined in the keyPressed method.

**The Obstacle Class**

For extra credit, I created a randomly-appearing, floating rectangle that vanishes upon contact with the ball and gives the user ten extra points. I also play a sound when this happens.

So how I do this is with a boolean value named disappear which I declare in the Game class. At the beginning of each level, disappear is set to false. While disappear is false and the ball brushes the area where the obstacle is, then they get an extra point and I play a sound, then set disappear to true. Now that disappear is true, you won’t see the obstacle anymore.

**if** (game.obs.ching() && game.disappear == **false**) {

Sound.***BELL***.play();

game.score+=10;

game.scorelabel.setText("Score: " + game.score);

game.disappear = **true**;

game.repaint();

}

This code will make more sense when you understand what ching() is.

**Methods in the Obstacle Class**

* **Paint**

This draws the obstacle. I set it so that it has a random x value, y value, height and width every time it is called.

* **Get Bounds**

This returns the bounds of the obstacle

* **Ching**

This returns a boolean value checking whether the bounds of the obstacle intersects the bounds of the ball. I called it Ching because that’s the sound I play whenever this is true.

**The Main Method**

I’m not sure if my main method should even have as much as it does, but that’s how it is so here’s the breakdown of it.

First, of course, I create a JFrame and a Game object and add the game to the frame. This ensures that you can actually *see* all my hard work.

The very next line is while (true). All this does is makes sure that all the code afterwards runs only if main is being run.

The next few lines is for Level 1. While the variable time which increments every second is not equal to levelonelimit, I call the move method is my game class. Remember that *game.move* just calls moveBall and movePaddle. After that I have a *Thread.sleep(10)* command, which tells the code to rerun every ten milliseconds. If time gets to the level one limit and the player still has more than 0 lives left, it’s time for level two.

Before I start Level 2, I restart the timer, reset time to 0, and re-randomize the values a, b and c for the obstacle. Then I do the exact same thing I did for level one. While the player is still alive, call *game.move* etc.

For Level 3, I do the exact same thing one more time, except this time, if time gets to levelthreelimit and the player is still alive, then they win the game.

The end!

**Requirements**

* player can control a paddle that slides horizontally back and forth at the bottom of the screen ✓
* ball is launched at start of game ✓
* ball bounces off walls and paddle ✓
* player scores point when ball bounces off paddle ✓
* player gets pre-set number of lives ✓
* program updates score and lives ✓
* there are levels of the game ✓
* there is a countdown timer ✓
* if ball is still bouncing when time is up, they have won the level ✓
* if player wins level, player earns bonus points ✓

**Things Done For Extra Credit**

* Added sound effects (worth up to 10%)
* Added randomly – appearing obstacle worth bonus points effects (worth up to 10%)
* Used Graphics2D package to make animation smooth (I’m just making stuff up at this point)
* Clear and prominent documentation (worth up to grader)