Kevin Ratovo-Mohamed Rached Waly

Pong’Em Up

I/Brief introduction of the game:

Pong’ em up is a game born from the fusion of a shoot ‘em up and Pong. The goal of the player is to clear all the enemy waves while both managing to keep the ball from passing the stick and dodging incoming projectiles.

II/Controls:

Arrow Keys to move. Spacebar to fire a projectile.

III/Functionalities:

- *Start up screen* : We wanted the main menu screen to be as catchy as possible. So we came up with the idea to have a minor preview of the game on its background(The ball will be bouncing on the walls and the stick will follow it automatically). This background is handled by the AnimationHandler Class.

  In the main menu we have 3 Buttons:
  -New Game: starts a new instance of the game
  -Options: opens the options pane
  -Quit: exits the game

- *Options Menu*: Here we wanted to give the user some freedom to customize the looks of the stick and the color of the ball. You can do it either while playing or in the startup screen. The changes will happen dynamically.

- *Pause menu*: The pause menu is set as a glasspane for our main Frame “ PongEmUp”. It has 5 buttons in total:
  - Resume: unpause the game and resume playing
  - Retry: restart the game from the beginning ie from the first wave
  - Options: opens the customization menu for the looks of the stick and the color of the ball
  - Go Back to Main Menu and Go To desktop.
- *3 Types of enemies*: Enemies inherit from a generic class Shooter, which is shared with Stick. The enemy’s behavior is defined in the method behaviorUpdate. Sentries loop a left right movement, while spinners move up and down.
  They can also have different fire patterns, and different colliders.

- *Enemy spawn by wave*: All enemies in a level are stored in a single array. They are then added by packs once the game detects that there are none left from the previous wave. As of now, we only have a single level, with a total of 5 waves.

- *<u>Collision system with attempted physical accuracy</u>*: The collision system tests intersection between all objects every game tic. We attempted to implement a nearest neighbor but we didn't have time. Implementing a nearest neighbor algorithm on our own is a bit tricky and we didn't want to import external libraries. Collisions relies on the CustomShape class. Each inherited class lists an intersect function for all other CustomShapes.

  As of now, only 2 different shapes are supported : Rectangle and Circle. A more generic PolygonShape was planned, but ended up taking too much time to debug.

  Each Entity implements a whenCollided method, which dictates what to do depending on what entity the collider hit. The most complex one is the Ball class one. Ball's collider is a CircleShape, and reads the normal hit in its intersect methods. We then compute the ball's new direction and speed thanks to this. There are still some slight bugs with the ball movement, mostly due to the rollforward paradigm we chose. After a collision, the ball is updated until it no longer intersects with the previous object. This leads to out of bounds and object traversing when the wrong normal is hit. Another edge case is when the ball hits two sides at once. It arbitrarily chooses a normal in a clockwise direction from top left. This is bandaid fixed by resetting the ball when such cases happen (without destroying the stick).

- *<u>Death animation and respawn</u>*:
  When enemies or the stick get destroyed, it starts a destruction sequence illustrated by a consecutifs photos stored in the resources folder.
  Also whenever the enemies get hit, their illustration will turn red for a brief period of time.
  Along with the death animation, the stick then respawns after 2.5 seconds where it starts blinking. The player can then resume moving the stick, and it is invulnerable to any incoming damage. In the meantime, the ball instantly respawns and an arrow indicates it's starting direction with a countdown.

- *<u>Bonus drops</u>*: Each death of an enemy will generate a random bonus from the 3 already defined bonuses. Each bonus will drop vertically:
  - Shield bonus: will make the stick invulnerable to projectiles for a certain amount of delay.
  - Length bonus: will make the stick longer but also will boost its health in the process
  - Life bonus: will add a life to the remaining healths of the player

  Bonuses can be acquired either by colliding with the stick or with the ball if it falls down it will be discarded.

- *<u>Status bar</u>*: it lists lives remaining, player's score and acquired bonuses

IV/The 2 most important abstract classes:

1)“Handler”:

-It has all the declarations and implementations of the methods that we need to run the game. It is the one taking the user’s input, updating the game, moving the objects, and solving the collisions. It has access to all the entities created, can modify them or remove them.

Most important attribute: physicalObjects an arrayList of collidable entities
It has 2 crucial methods for the running of our game:

- update: This method will be called each tick of the timer; updates all the entities, calls the solveCollisions.
- solveCollisions: in this method we will loop on all the physical objects in order to check for collisions and call the appropriate whenCollided method.

From this class we get the GameHandler and the AnimationHandler classes. The first one takes care of the game in itself and the second one will handle the preview of the game on the background of the main menu.

2)“Entity”:

-This class will represent each object the player can see on the screen.
It has 3 crucial methods for the running of our game:

- drawEntity: handles the drawing of each entity.
- whenCollided: takes into a parameter another entity. It will change the behaviour of the entity based on the type of collision.
- getShape: returns an instance of the CustomShape class that will be used in detecting collisions.