Declan Urbaniak-Dornstauder
3716081
COMP 452
Assignment 1

Note to instructor and markers

This is the first game I have developed. Although I assume this course is intended for those with prior game development experience, I am enrolled out of interest and to earn credit for a general introductory AI course at my main institution. Given this, I am largely unfamiliar with standard game development practices, and am uncertain about the implementation of this game. I decided to use JavaFX for this implementation because I have prior experience using it for UI development, and intend to continue doing so. If possible, please let me know if this or anything else I have done in this implementation will cause issues for this course in the future, as I'm willing to adapt to make sure I meet the criteria for each assignment.

Game description

- This is a simple game where the player must avoid the enemies on-screen for 15 seconds.
- The player controls the yellow character with the mouse/cursor of their computer.
- There are two types of enemies, five in total:
  - Four blue enemies begin moving in a random direction at a random speed and 'bounce' off the boundaries of the screen
  - One red enemy seeks the players while avoiding the blue enemies. When 5 seconds remain on the timer, the red enemy moves faster and no longer avoids the other enemies.
- The player begins with 3 HP. If an enemy comes in contact with the player, the player loses 1 HP.
- If the player successfully avoids the enemies for 15 seconds, they win.
- Otherwise, if the player has 0 HP before the timer ends, they lose.

How to run

- I have provided both a zip file containing the entire project, as well as an executable jar file.
- The jar file can be ran via the command line with *java -jar a1.jar* or by simply double clicking the executable if your OS allows it.
- Otherwise, the project can be compiled and ran using *./run* in the project's root directory. You can also rebuild the provided jar using *./build* in the root directory, and the newly built jar will be in the *out/* directory.

Steering behaviours

Simple steering behaviours:

- wander():
  - Upon initialization, the blue enemies begin moving in a random direction. When a blue enemy comes in contact with the boundaries of the screen, its x and/or y velocity negates to change its direction, making it appear to 'bounce' off the boundaries.

- This behaviour is constant – nothing is required from the player to cause the blue enemies to wander other than starting the game.

```
private void wander() {
  // Check if enemy is outside borders of screen
  if ((posX >= maxX || posX < 0) || (posY >= maxY || posY < 0)) {
    if (posX >= maxX || posX < 0) deltaX = -deltaX; // Invert x if position if needed
    else deltaY = -deltaY; // Invert y position if needed
    deltaR = rng.nextDouble(2) - 1; // deltaR is the value that allows the enemy sprite to rotate
  }
}
```

Source: *src/java/com/example/a1/model/BlueEnemy.java*

- seek() :
  - The red enemy constantly moves towards the player using the difference between its position and the player's position. This is normalized using its distance to the player to ensure it moves towards the player at a constant velocity.
  - Again, this behaviour is constant.

```
private void seek(double playerX, double playerY) {
    double distanceToPLayer = distance(playerX, playerY);
    deltaX = (playerX - posX) / distanceToPLayer;
    deltaY = (playerY - posY) / distanceToPLayer;
}
```

Source: *src/java/com/example/a1/model/RedEnemy.java*

Complex steering behaviours

- seek() + avoid():
  - the combination of seek() and avoid() allows the red enemy to seek the player while modifying its velocity to avoid or repel itself from the other enemies on-screen, giving it a more natural path in its pursuit.
  - To induce this behaviour, the player must be in a position where a blue enemy is in the trajectory of the red enemy as it seeks the player.
  - In the current implementation, there is an issue with this behaviour related to the difference between a red enemy and a blue enemy's velocities that sometimes causes the red enemy to pass through the blue enemy.

```java
private void avoid(List<Enemy> enemies) {
  for (Enemy enemy : enemies) { // Calculate using positions of all enemies on screen
    if (enemy == this) continue; // skip self

    double distanceToEnemyX = this.posX - enemy.posX;
    double distanceToEnemyY = this.posY - enemy.posY;
    double distanceToEnemy = Math.sqrt(Math.pow(distanceToEnemyX, 2) + Math.pow(distanceToEnemyY, 2));

    double separationMod = 1.5; // Value to determine the boundary at which the repel should begin
    double repelMod = 1.2; // Value to adjust the strength of the repel force

    if (distanceToEnemy < size * separationMod) { // Repel if an enemy is at or within repel boundary
        double repellingForceX = (distanceToEnemyX / distanceToEnemy) * repelMod;
        double repellingForceY = (distanceToEnemyY / distanceToEnemy) * repelMod;
        deltaX += repellingForceX;
        deltaY += repellingForceY;
    }
  }
}
```

Source: *src/java/com/example/a1/model/RedEnemy.java*