```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Parent;
import javafx.scene.layout.Pane;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.geometry.Pos;
import javafx.animation.AnimationTimer;
import java.util.List;
import javafx.scene.control.Label;
import java.util.Timer;
import java.util.TimerTask;
import javafx.scene.input.KeyEvent;
import javafx.event.EventHandler;
import java.util.ArrayList;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Shape;

public class Game extends Application {

    /** ---------- **/
    /*-Time-Variables-*/
    /** ---------- **/
    private int time = 0;
    private int timeBetweenSummmon;
    private int startTime;

    /** ------------------------ **/
    /*-Possible-Letts-Of-Bad-Guys-*/
    /** ------------------------ **/
    private static String[] possibleLetters = { "a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
            "k", "l", "m", "n", "o", "p", "q", "r", "s", "t",
            "u", "v", "w", "x", "y", "z" };

    /** ---------- **/
    /** -Core-game- **/
    /** ---------- **/
    Pane root = new Pane();
    Region protect = new Region();
    TextField shooter = new TextField();
    Circle protectOutline = new Circle();
```

```
boolean gameOver;
private boolean firstTime;
// rounds/make harder
private int numberDestroyed;
private boolean wasEnemyKilled;

/** ---------------------- */
/*-Variables-for-BadGuys-*/
/** ---------------------- */
volatile boolean isSelected = false;

/**
 * Class for the letters that are flying at you
 */
private class BadGuys extends Label {
    int distX;
    int distY;

    /**
     * Constructor
     *
     * @param x     X location of enemy
     * @param y     Y location of enemy
     * @param distX Distance to earth/protect
     * @param distY Distance to earth/protect
     */
    BadGuys(int x, int y, int distX, int distY) {
        super(possibleLetters[(int) (Math.random() * possibleLetters.length)]);
        this.distX = distX;
        this.distY = distY;
        setTranslateX(x);
        setTranslateY(y);
        setPrefSize(50, 50);
        setAlignment(Pos.CENTER);
        setId("meteor");
    }

    /**
     * @return the distX which is distance in x to protect
     */
    public double getDistX() {
        return distX;
    }
```

```
    /**
     * @return the distY which is distance in y to protect
     */
    public double getDistY() {
        return distY;
    }

}

/**
 * Bullets that will be shot at words
 */
private class Bullet extends Region {
    BadGuys target;
    int distX;
    int distY;

    /**
     * Constructor
     *
     * @param target constructor takes BadGuy instance to target
     */
    Bullet(BadGuys target) {
        setPrefSize(10, 10);
        this.target = target;
        setId("bullet");
        // places at center of earth
        setTranslateX(protectOutline.getLayoutX());
        setTranslateY(protectOutline.getLayoutY());
        distY = Math.abs((int) (protectOutline.getLayoutX() - target.getTranslateY()));
        distX = Math.abs((int) (protectOutline.getLayoutY() - target.getTranslateX()));
    }

    /**
     * @return returns the target this bullet is chasing
     */
    public BadGuys getTarget() {
        return this.target;
    }

    /**
     * @return returns distX to target
     */
    public double getDistX() {
```

```
        return this.distX;
    }

    /**
     * @return returns distY to target
     */
    public double getDistY() {
        return this.distY;
    }
}
/**
 * Helper method that returns the enemies from all elements in the root
 * @return List of type bad guys
 */
private List<BadGuys> getBadGuys() {
    List<BadGuys> enemies = new ArrayList<BadGuys>();

    root.getChildren().forEach(e -> {
        if (e instanceof BadGuys) {
            enemies.add((BadGuys) e);
        }
    });
    return enemies;
}

/**
 * Helper method that returns the bullets from all elements in the root
 * @return List of all bullets in root
 */
private List<Bullet> getBullets() {
    List<Bullet> bullets = new ArrayList<Bullet>();
    root.getChildren().forEach(e -> {
        if (e instanceof Bullet)
            bullets.add((Bullet) e);
    });
    return bullets;
}

/**
 * Method that makes enemy and places it randomly around the root border
 */
private void makeEnemy() {
    int x;
    int y;
```

```
    int distX;
    int distY;

    if (Math.random() > .50) {
        // summons far right or left
        if (Math.random() > .50) {
            x = (int) (root.getPrefWidth() + (Math.random() * 100));
            distX = (int) (x - 400);
        } else {
            x = -100 + (int) (Math.random() * 100);
            distX = (int) (400 - x);
        }
        y = (int) (Math.random() * root.getPrefHeight());
        distY = Math.abs((int) (400 - y));
    } else {
        if (Math.random() > 0.5) {
            y = (int) (root.getPrefHeight() + (Math.random() * 100));
            distY = (int) (y - 400);
        } else {
            y = (int) (-100 + (Math.random() * 100));
            distY = (int) (400 - y);
        }
        x = (int) (Math.random() * root.getPrefWidth());
        distX = Math.abs((int) (400 - x));
    }
    root.getChildren().add(new BadGuys(x, y, distX, distY));
}

/**
 * This method moves the enemies closer to earth.
 * If the enemies intersect with the earth then
 * boolean gameOver = true;
 */
private void updateEnemies() {
    if (getBadGuys() != null) {
        for (BadGuys e : getBadGuys()) {
            Circle cast = new Circle(25);
            cast.setTranslateX(e.getTranslateX() + (e.getWidth() / 2));
            cast.setTranslateY(e.getTranslateY() + (e.getHeight() / 2));
            cast.setFill(Color.TRANSPARENT);
            root.getChildren().add(cast);
            if (e.getTranslateX() < 400) {
                e.setTranslateX((e.getTranslateX() + (e.getDistX() / 300)));
            }
```

```
            if (e.getTranslateX() > 400) {
                e.setTranslateX(e.getTranslateX() - (e.getDistX() / 300));
            }
            if (e.getTranslateY() < 400) {
                e.setTranslateY(e.getTranslateY() + (e.getDistY() / 300));
            }
            if (e.getTranslateY() > 400) {
                e.setTranslateY(e.getTranslateY() - (e.getDistY() / 300));
            }
            if (Shape.intersect(cast, protectOutline).getLayoutBounds().getMinX() != 0
                    || Shape.intersect(cast, protectOutline).getLayoutBounds().getMinY() != 0)
                gameOver = true;
        }
    }
}

/**
 * Method moves bullets closer to their targets.
 * If they intersect then remove both from root.
 */
private void updateBullets() {
    if (getBullets() != null) {
        for (Bullet b : getBullets()) {
            if (b.getTarget().getTranslateX() > 400)
                b.setTranslateX(b.getTranslateX() + (b.getDistX() / 30));
            if (b.getTarget().getTranslateX() < 400)
                b.setTranslateX(b.getTranslateX() - (b.getDistX() / 30));
            if (b.getTarget().getTranslateY() > 400)
                b.setTranslateY(b.getTranslateY() + (b.getDistY() / 30));
            if (b.getTarget().getTranslateY() < 400)
                b.setTranslateY(b.getTranslateY() - (b.getDistY() / 30));
            if (b.getBoundsInParent().intersects(b.getTarget().getBoundsInParent()))
                root.getChildren().removeAll(b, b.getTarget());
        }
    }
}

/**
 * Animation timer running every 0.016 seconds.
 * Update() contains moving/updating everything in game.
 */
AnimationTimer timer = new AnimationTimer() {
    @Override
    public void handle(long now) {
```

```
        update();
    }
};

/**
 * The eventHandler for the shooter.
 * When letter is pressed on keyboard if an
 * ememy exists with the letter create bullet
 * targeting that latter. Else then create two
 * enemies.
 */
EventHandler<KeyEvent> keyInput = new EventHandler<KeyEvent>() {
    @Override
    public void handle(KeyEvent e) {
        if (e.getText().length() != 0) {
            shooter.setEditable(true);
            if (getBadGuys() != null) {
                for (BadGuys x : getBadGuys()) {
                    if (e.getText().charAt(0) == x.getText().charAt(0)) {
                        if (x.getId() == "meteor") {
                            x.setId("meteorSelected");
                            shooter.setEditable(false);
                            root.getChildren().add(new Bullet(x));
                            numberDestroyed++;
                            wasEnemyKilled = true;
                            break;
                        }
                    }
                }
                if (shooter.isEditable()) {
                    makeEnemy();
                    makeEnemy();
                }
            }
            shooter.setEditable(false);
        }
    }
};

/**
 * Method updates all characters in game.
 * Method makes enemy if enough time has passed.
 * If five new enemies have been killed then decrease
 * the time between summoning enemies.
```

```
 * If gameOver is true then reset game.
 */
private void update() {
    time += 16;
    updateEnemies();
    updateBullets();

    if (time % timeBetweenSummmon == 0) {
        makeEnemy();
    }

    if (wasEnemyKilled && numberDestroyed % 5 == 0) {
        timeBetweenSummmon -= 64;
        wasEnemyKilled = false;
    }

    if (gameOver) {
        firstTime = false;
        shooter.removeEventHandler(KeyEvent.KEY_PRESSED, keyInput);
        timer.stop();
        resetGame();
    }
}

/** ------------------------ **/
/** -Add-The-Sprites of Game- **/
/** ------------------------ **/
/**
 * Method is ran to create the elements of the game scene.
 * @return Parent which will be the scene.
 */
private Parent createGameContent() {

    root.setPrefSize(800, 800);

    protectOutline.setLayoutX(root.getPrefWidth() / 2);
    protectOutline.setLayoutY(root.getPrefHeight() / 2);
    protectOutline.setFill(Color.TRANSPARENT);
    protectOutline.setRadius(50);

    protect.setPrefSize(100, 100);
    protect.setLayoutX((root.getPrefWidth() / 2) - (protect.getPrefWidth() / 2));
    protect.setLayoutY((root.getPrefWidth() / 2) - (protect.getPrefHeight() / 2));
    protect.setId("earth");
```

```
    shooter.setAlignment(Pos.CENTER);
    shooter.setPrefWidth(50);
    shooter.setPrefHeight(50);
    shooter.setLayoutX((root.getPrefWidth() / 2) - (shooter.getPrefWidth() / 2));
    shooter.setLayoutY((root.getPrefHeight() / 2) - (shooter.getPrefHeight() / 2));
    shooter.setId("transparent");
    shooter.requestFocus();

    root.getChildren().addAll(protect, shooter, protectOutline);

    shooter.addEventHandler(KeyEvent.KEY_PRESSED, keyInput);
    numberDestroyed = 0;
    wasEnemyKilled = false;
    timeBetweenSummmon = 504;
    gameOver = false;
    firstTime = true;

    return root;
}

/**
 * Method is ran to declare all variables back
 * to before game started.
 * Add the logo, clickToStart and results if it
 * is not the first time playing. Make clickToStart
 * on click start game.
 */
private void resetGame() {
    Label results = new Label("You protected Earth for " + time / 1000 + " seconds!");

    time = 0;
    startTime = 0;
    numberDestroyed = 0;
    timeBetweenSummmon = 1008;
    gameOver = false;
    wasEnemyKilled = false;
    shooter.setEditable(false);
    shooter.removeEventHandler(KeyEvent.KEY_PRESSED, keyInput);

    results.setAlignment(Pos.CENTER);
    results.setPrefSize(500, 50);
    results.setLayoutX((root.getPrefWidth() / 2) - (results.getPrefWidth() / 2));
    results.setLayoutY((root.getPrefHeight() / 2) + (protect.getPrefWidth()));
```

```
results.setId("results-text");

Region logo = new Region();
logo.setPrefSize(1000, 400);
logo.setId("logo");
logo.setLayoutX((root.getPrefWidth() / 2) - (logo.getPrefWidth() / 2));

Region clickToStart = new Region();
clickToStart.setPrefSize(200, 50);
clickToStart.getStyleClass().add("clickToStart");
clickToStart.setLayoutX((root.getPrefWidth() / 2) - (clickToStart.getPrefWidth() / 2));
clickToStart.setLayoutY((root.getPrefHeight() / 2) - (clickToStart.getPrefHeight() * 2));

root.getChildren().addAll(logo, clickToStart);

if (!firstTime) {
    root.getChildren().add(results);
}

clickToStart.setOnMouseClicked(e -> {
    clickToStart.setDisable(true);
    Timer startTimer = new Timer();
    TimerTask startTask = new TimerTask() {
        @Override
        public void run() {
            if (startTime < 3000) {
                startTime += 1000;
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        if (startTime == 1000)
                            clickToStart.setId("three");
                        if (startTime == 2000)
                            clickToStart.setId("two");
                        if (startTime == 3000)
                            clickToStart.setId("one");
                    }
                });
            } else {
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        root.getChildren().removeAll(clickToStart, results, logo);
                        removeElements();
```

```
                    }
                });
                shooter.addEventHandler(KeyEvent.KEY_PRESSED, keyInput);
                startTimer.cancel();
                timer.start();
            }
        }
    };
    startTimer.schedule(startTask, 0, 1000);
  });
}

/**
 * Method is ran and removes the bullets and
 * enemies from scene. Called at starting new
 * game.
 */
private void removeElements() {
    if (getBadGuys() != null) {
        for (BadGuys e : getBadGuys()) {
            root.getChildren().remove(e);
        }
    }
    if (getBullets() != null) {
        for (Bullet b : getBullets()) {
            root.getChildren().remove(b);
        }
    }
}

/**
 * Start method calls other methods and sets scene.
 * Adds the style sheet.
 */
public void start(Stage primaryStage) {
    Scene scene = new Scene(createGameContent());
    scene.getStylesheets().add("com/example/styleSheet.css");
    primaryStage.setScene(scene);
    primaryStage.setResizable(false);
    resetGame();
    primaryStage.show();
}
/**
 * Main method to start game.
```

```
     * @param args
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```