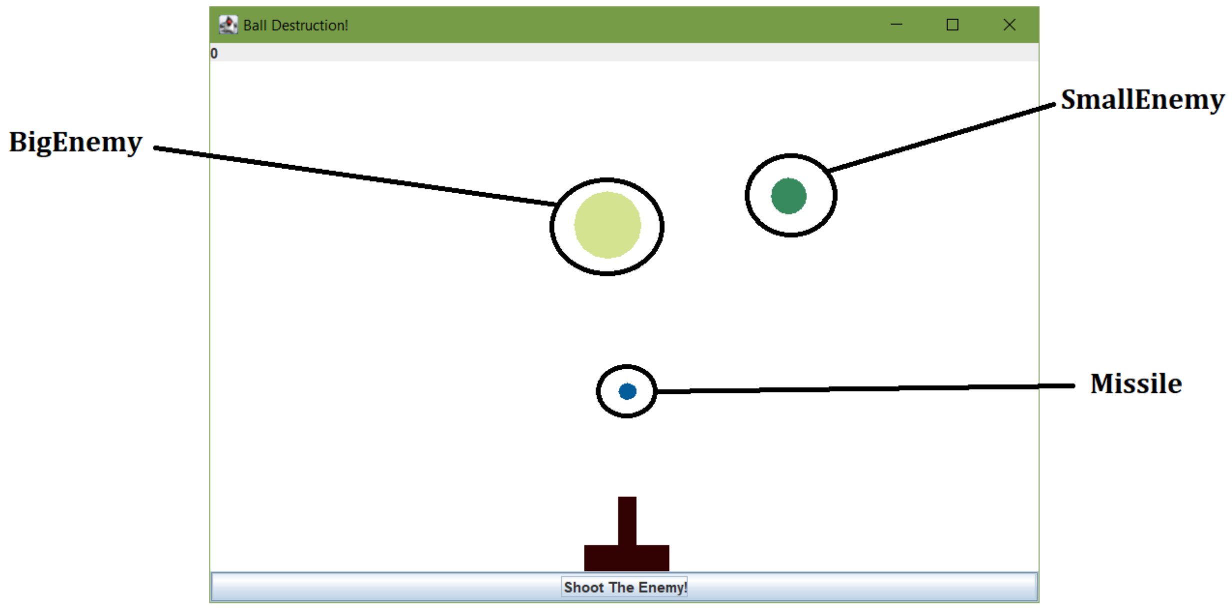
CSE 271 Project 4 – Graphical User Interfaces Video Game
Spring 2022
Assigned: 3/31/2022
Due: 4/15/2022

Introduction:

In this project, you will be creating a basic video game using the knowledge of Graphical User Interfaces (GUI) and Object-Oriented programming you have learned so far. You will be given starter code to assist in collision detection as well as a basic framework for the video game. Your goal is to design and fill out the remaining pieces to make the game functional. Furthermore, there will be **extra credit** assigned based on creativity. That is, if you put in the additional time and effort to create a unique game, you can earn an additional 15 points of extra credit (15% of a project). This will give you an opportunity to improve your projects category grade if you wish to do it. The uniqueness can come from special rules, graphics, etc. Note for projects 1-5, you must submit work to Canvas once during the first 7 days of when the project was assigned to show progress towards completion (please see the rubric).

Video Game:

Develop a GUI to contain the video game painting and allow the user to play the game. A minimal version of this is displayed below. There are two types of enemies, **SmallEnemy** and **BigEnemy**, as well as a **Missile** to hit the enemies. A JLabel keeps track of the score at the top left of the JFrame. Next, a JButton at the bottom of the JFrame allows us to shoot a Missile. Finally, a **Turret** is painted at the bottom center of the JFrame to act as the vessel to shoot a Missile.



Game Rules:

The minimal ruleset of the game is simple. You press the button to shoot a Missile. If you hit a BigEnemy, then you get 100 points. If you hit a SmallEnemy, then you get 150 points. You can shoot a total of 10 Missiles and then the game ends. If your score is 800 or above, you win, else you lose. Note that these are suggested rules and you may create your own ruleset.

Class Files:

The classes needed to complete this project are listed below. Take note of the portions of code that are provided to you as starter code. Also, for the extra credit, you are allowed to modify these. Please read the Extra Credit Hint below.

1. **Tester –** This class inherits JFrame and contains all the information necessary to display the GUI and to have the game infinitely operate.
   a. **Instance Properties:**
      i. **private static final int WINDOW_WIDTH** – The width of the JFrame.
      ii. **private static final int WINDOW_HEIGHT** – The height of the JFrame.
      iii. **private int score** – The current score of the game.
      iv. **private int timer** – The timer to assist in the game loop.
      v. **private int missilesFired** – The number of Missiles fired.
      vi. **private JLabel scoreLabel** – The JLabel to display the score.
      vii. **private JButton fireButton** – The JButton to fire a Missile.
      viii. **private GamePanel panel** – The custom JPanel to contain the GUI objects.
   b. **Methods:**
      i. **public Tester()** – **(PARTIALLY PROVIDED)** The default (or empty) constructor to initialize the instance properties and setup the necessary GUI pieces such as the DefaultCloseOperation, Visibility, Layout, etc.
      ii. **public void start() – (PROVIDED)** This method is called to start the video game. It will call the infinite gameLoop() method to execute all aspects of the game.
      iii. **public void gameLoop() – (PARTIALLY PROVIDED)** The game loop to infinitely run the game until it finishes. The game loop does a sequential set of activities each iteration. That is, pause the game, detect collision of Missiles and Enemies, calculate score, move the Enemies and Missiles, repaint the Enemies and Missiles, check if the game has finished, and determine if a new Enemy needs to be added (say if the timer is equal to 300).
      iv. **public void pauseGame()** – **(PROVIDED)** This method pauses the game for 30ms to allow calculations to be done for each gameLoop iteration before the next frame is shown to the user.
      v. **public void centerFrame(JFrame frame) – (PROVIDED)** This method simply centers the JFrame to the middle of the users monitor.
      vi. **public void setTimer() – (PROVIDED)** This method randomly sets the Enemy spawn timer.
      vii. **public static void main(String[] args) – (PROVIDED)** The main method to execute the program. Here you make the JFrame of type Tester to start the game.
2. **GamePanel –** This class inherits JPanel and contains all the GUI elements for the custom JPanel to make the game function.
   a. **Instance Properties:**
      i. **private int totalScore** – The score to update the game by.
      ii. **private boolean isNextEnemyBig** – A boolean to determine which enemy to generate next, either a BigEnemy or SmallEnemy.
      iii. **private Turret turret** – A Turret object to use throughout the class.
      iv. **private ArrayList<Enemy>** – A polymorphic ArrayList to hold both BigEnemy and SmallEnemy objects.
      v. **private ArrayList<Missile>** – An ArrayList to hold Missile objects.
   b. **Methods:**

        i. **public GamePanel()** – The default (or empty) constructor to initialize the instance properties. It also adds a new BigEnemy and a new SmallEnemy to the Enemy ArrayList to start the game.
        ii. **public void paintComponent(Graphics g)** – **(PARTIALLY PROVIDED)** Paint the JPanel white, for every Enemy and every Missile, call their respective paintComponent() methods, and paint the Turret with the Turret's respective paintComponent() method.
        iii. **public void move()** – For every Enemy and every Missile, calls the corresponding Enemy and Missile move() methods in their respective classes.
        iv. **public void addMissile()** – Adds a new Missile to the Missile ArrayList.
        v. **public void addEnemy()** – Adds a new BigEnemy or SmallEnemy to the Enemy ArrayList depending on the isNextEnemyBig boolean instance property.
        vi. **public int getTotalScore()** – Returns the totalScore instance property.
        vii. **public void detectCollision()– (PROVIDED)** This method detects if a Missile hits an Enemy and updates the score. The Missile that hit the enemy is removed and the Enemy is affected by calling the processCollission() method in the corresponding Enemy class.

3. **Turret –** This class inherits JComponent and holds the information necessary for a **Turret**.
    a. **Instance Properties:**
        i. **private Rectangle base** – The rectangle that represents the base of the Turret.
        ii. **private Rectangle turret** – The rectangle that represents the turret of the Turret.
        iii. **private Color turrentColor** – The color of the Turret.
    b. **Methods:**
        i. **public Turret()** – The default (or empty) constructor to initialize the instance properties. The values for the Turret's Rectangle objects, as well as the color, are for you to decide. Recall that a Rectangle has a constructor that takes in the x coordinate, y coordinate, width, and height.
        ii. **public paintComponent(Graphics g)** – The paintComponent() method that paints the Turret base and Turret barrel.
4. **Missile –** This class inherits JComponent and holds the information necessary to a **Missile**. Hint: The **Missile** class inherits JComponent. This means that from the constructor, and the child classes, when we are working with the missileXCoord, missileYCoord, missileHeight, and missileWidth, these can be set using the super.setBounds() method. Furthermore, we can access each of these four instance properties with the super.getX() for example.
    a. **Instance Properties:**
        i. **private int missileSpeed** – The speed of the Missile.
        ii. **private Color missileColor** – The color of the Missile.
    b. **Methods:**
        i. **public Missile()** – The default (or empty) constructor to initialize the instance properties. The values for the Missile, as well as the color, are for you to decide.
        ii. **public paintComponent(Graphics g)** – The paintComponent() method that paints the Missile based on the current x and y coordinates.
        iii. **public void move(int width, int height, ArrayList<Missile> list, int missile)** – Determine if the missile is off the screen and remove it from the ArrayList if it is. Also, determine the next position on the screen the Missile should appear on after being repainted.
5. **Enemy –** This class is ***abstract***, inherits JComponent, and holds the information necessary to an **Enemy**. The children to this class are **BigEnemy** and **SmallEnemy**. Hint: The **Enemy** class inherits JComponent. This means that from the constructor, and the child classes, when we are working with the enemyXCoord, enemyYCoord, enemyHeight, and enemyWidth, these can be

set using the super.setBounds() method. Furthermore, we can access each of these four instance properties with the super.getX() for example.

   a. **Instance Properties:**
      i. **private double enemySpeed** – The speed of the Enemy.
      ii. **private Color enemyColor** – The color of the Enemy.
   b. **Methods:**
      i. **public Enemy(int enemyXCoord, int enemyYCoord, int enemyHeight, int enemyWidth, double enemySpeed)** – A partial constructor that sets the coordinates, size, and speed of the Enemy.
      ii. **public abstract void processCollision(ArrayList<Enemy> list, int enemy)** – An abstract method which determines what occurs when a Missile hits an Enemy.
      iii. **public abstract setColor()** – An abstract method which generates and sets the color of the Enemy.
      iv. **public abstract void move(int width, int height)** – An abstract method which computes and updates the next position of the Enemy with respect to the JFrame width and height.
      v. **public paintComponent(Graphics g)** – The paintComponent() method that paints the Enemy based on the current x and y coordinates.
      vi. **Getters and Setters for each instance property.**

6. **BigEnemy –** This class inherits the class **Enemy** and must implement the abstract methods it contains.
   a. **Instance Properties:**
      i. None.
   b. **Methods:**
      i. **public BigEnemy()** – The default (or empty) constructor to initialize the instance properties. The values for the BigEnemy, as well as the color, are for you to decide.
      ii. **public void setColor()** – This method is the implementation of the Enemy's abstract setColor() method. You will choose the color of the BigEnemy and set the parent's color accordingly.
      iii. **public void move(int width, int height)** – This method is the implementation of the Enemy's abstract move() method. You will check if the BigEnemy has hit a wall or not, then reverse the direction if it has. Also, you will update the coordinates of the BigEnemy for the next frame.
      iv. **public void processCollision(ArrayList<Enemy>, int bigEnemy)** – This method is the implementation of the Enemy's abstract processCollision() method. If this method is called, then a Missile hit a BigEnemy. You will check to see if either the BigEnemy's height or width is less than zero. If so, remove the BigEnemy from the ArrayList. If not, shrink the size of the BigEnemy by some amount you decide.
7. **SmallEnemy –** This class inherits the class **Enemy** and must implement the abstract methods it contains.
   a. **Instance Properties:**
      i. None.
   b. **Methods:**
      i. **public SmallEnemy()** – The default (or empty) constructor to initialize the instance properties. The values for the SmallEnemy, as well as the color, are for you to decide.

ii. **public void setColor()** – This method is the implementation of the Enemy's abstract setColor() method. You will choose the color of the SmallEnemy and set the parent's color accordingly.

iii. **public void move(int width, int height)** – This method is the implementation of the Enemy's abstract move() method. You will check if the SmallEnemy has hit a wall or not, then reverse the direction if it has. Also, you will update the coordinates of the SmallEnemy for the next frame. Finally, you will update the speed to go faster each move() call depending on the positive or negative direction the SmallEnemy is currently going in.

iv. **public void processCollision(ArrayList<Enemy>, int bigEnemy)** – This method is the implementation of the Enemy's abstract processCollision() method. If this method is called, then a Missile hit a SmallEnemy. You will check to see if either the SmallEnemy's height or width is less than zero. If so, remove the SmallEnemy from the ArrayList. If not, shrink the size of the SmallEnemy by some amount you decide that is less than what you did for BigEnemy's processCollision().

Extra Credit Hint:

You may want to change the rule set to make it more complicated or exciting. You can also investigate introducing image files and sound effects using a bit of additional research. Perhaps moving the entire turret could be cool. Finally, maybe introducing key clicks or mouse clicks instead of the basic button at the bottom of the JFrame could be interesting. As long as you can shoot a missile at enemies and get a game over screen, you can be as creative as you want (this includes changing classes, adding classes, etc.)!

JavaDoc Style Comments:

You are required to make JavaDoc comments for all classes and methods including parameters and return descriptions.

Important Note:

Please make sure the file names are correct and the code is well commented!

Submission Instructions:

Submit your .java files (**Tester.java, GamePanel.java, Turret.java, Missile.java, Enemy.java, BigEnemy.java, SmallEnemy.java, and any other .javas you made for this project**) zipped to the corresponding Project 4 Canvas folder.

Rubric:

Task	**Grade**
GUI	
The GUI is visible and contains components that display information such as the score and any other necessary pieces of information to play the game	10
The GUI has the Title, DefaultCloseOperation, Size, and Visibility correctly set	10
Functionality	
The game loop is functioning as intended following the provided instructions	10
The user can interact with the GUI to fire Missiles and hit Enemies	10
When an Enemy is hit with a Missile, the score increases, and the Enemy is visually effected	7
Enemies	
The inheritance hierarchy with the abstract Enemy is followed	8
Both the SmallEnemy and BigEnemy are visually unique and have specific rules in their move() and processCollision() methods	8
The Enemies are painted and move on the GUI	7
Missiles	
The Missiles are painted and move on the GUI	7
The Missile is removed when it hits an Enemy or goes off the screen	6
Turret	
The Turret is painted on the GUI	6
It visually looks like the Missiles are fired from the Turret	6
JavaDoc and Intermediate Submission	
JavaDoc present for all classes and methods and followed the Miami University coding guidelines (can only lose points)	0 (-10)
Provided an intermediate submission with noticeable progress	5
Total	**100**
Extra Credit	
Provided unique additions to the game that are reasonably past the minimal set of guidelines provided for the project	15
Total	**115**