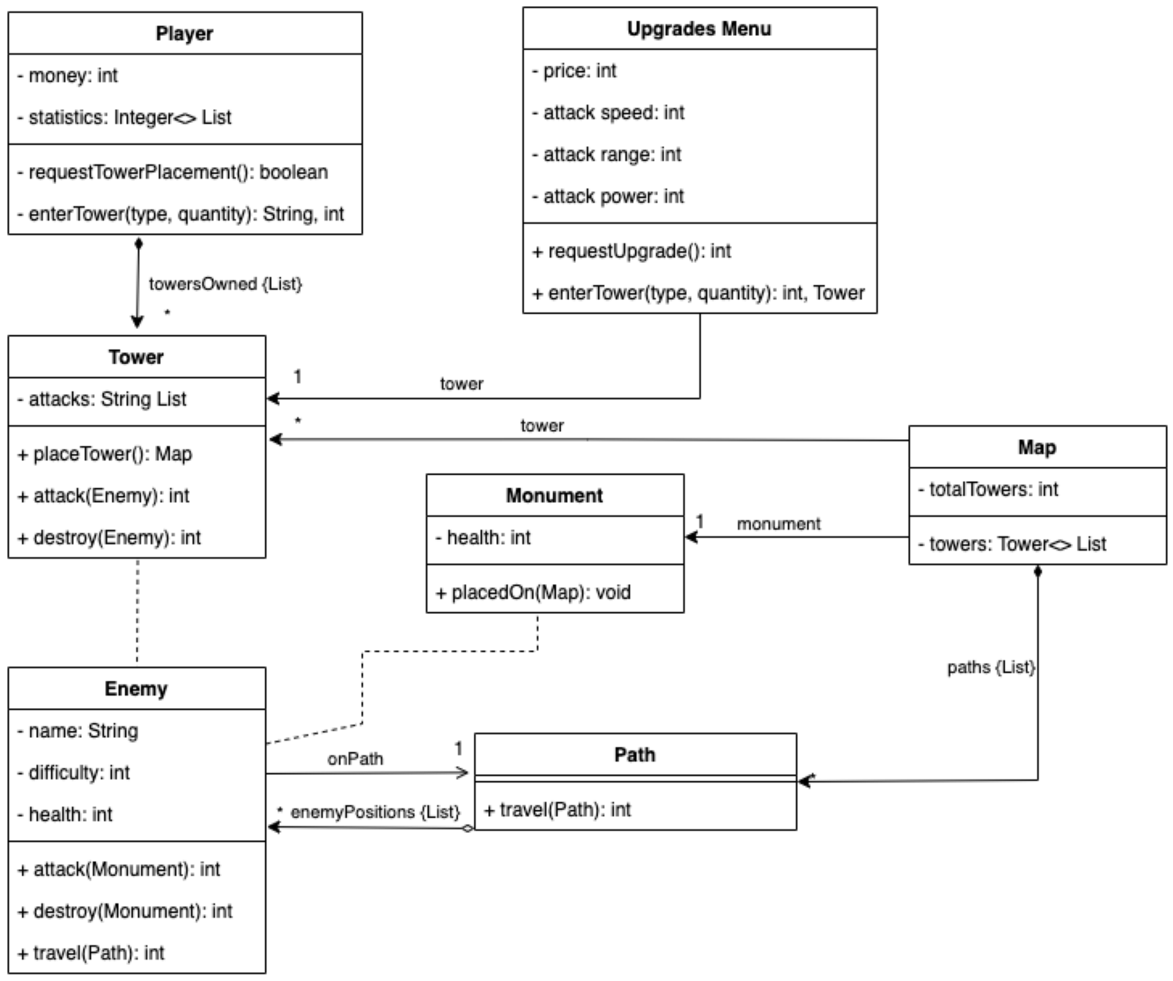
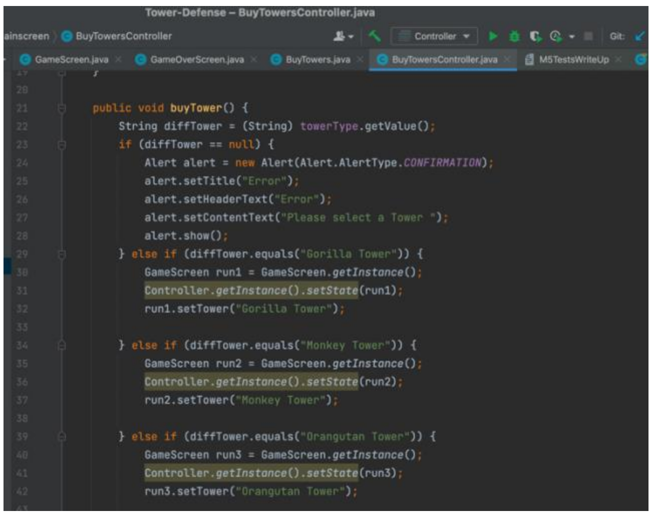
DCD



Player
- money: int
- statistics: Integer<> List
- requestTowerPlacement(): boolean
- enterTower(type, quantity): String, int
towersOwned {List}
*
Upgrades Menu
- price: int
- attack speed: int
- attack range: int
- attack power: int
+ requestUpgrade(): int
+ enterTower(type, quantity): int, Tower
Tower
- attacks: String List
+ placeTower(): Map
+ attack(Enemy): int
+ destroy(Enemy): int
1
tower
*
tower
Map
- totalTowers: int
- towers: Tower<> List
Monument
- health: int
+ placedOn(Map): void
1
monument
paths {List}
Enemy
- name: String
- difficulty: int
- health: int
+ attack(Monument): int
+ destroy(Monument): int
+ travel(Path): int
onPath
1
Path
+ travel(Path): int
* enemyPositions {List}

SOLID/GRASP WRITEUP

1: Single Responsibility Principle (SOLID)



```java
    public void buyTower() {
        String diffTower = (String) towerType.getValue();
        if (diffTower == null) {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Error");
            alert.setHeaderText("Error");
            alert.setContentText("Please select a Tower ");
            alert.show();
        } else if (diffTower.equals("Gorilla Tower")) {
            GameScreen run1 = GameScreen.getInstance();
            Controller.getInstance().setState(run1);
            run1.setTower("Gorilla Tower");

        } else if (diffTower.equals("Monkey Tower")) {
            GameScreen run2 = GameScreen.getInstance();
            Controller.getInstance().setState(run2);
            run2.setTower("Monkey Tower");

        } else if (diffTower.equals("Orangutan Tower")) {
            GameScreen run3 = GameScreen.getInstance();
            Controller.getInstance().setState(run3);
            run3.setTower("Orangutan Tower");
```

The single responsibility principle states that a class should only be responsible for one thing.

The 'BuyTowersController' class has a precise name.

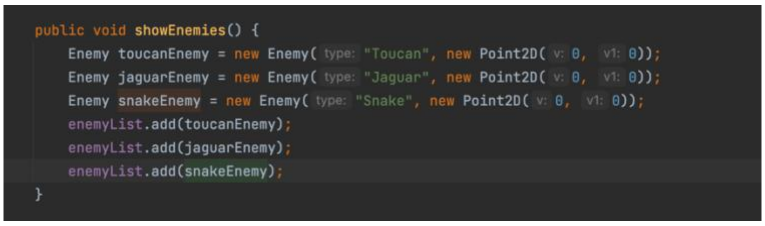
As shown in the screenshot to the left, the controller class has only one distinct function which is 'purchase' the tower that the user has inputted by asserting the tower type is equal to the correct string.

2: Creator (GRASP)

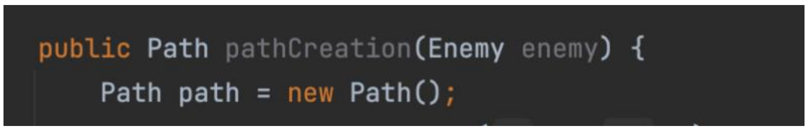
The creator principle states that we can assign B to create class A if class B contains or aggregates A, records A, closely uses A, or has the initializing data for A.

In our code, the creator principle is implanted with class A being the 'Enemy' class and class B being the 'GameScreen class.'

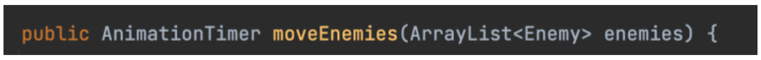
Shown by the first screenshot, the GameScreen class creates instances of the Enemy class because it closely uses the Enemy class by creating a path for the enemy to move along, as well as implementing the moveEnemies and attackEnemies methods.



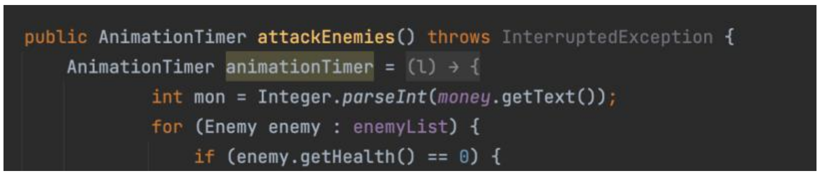
```java
public void showEnemies() {
    Enemy toucanEnemy = new Enemy( type: "Toucan", new Point2D( v: 0,  v1: 0));
    Enemy jaguarEnemy = new Enemy( type: "Jaguar", new Point2D( v: 0,  v1: 0));
    Enemy snakeEnemy = new Enemy( type: "Snake", new Point2D( v: 0,  v1: 0));
    enemyList.add(toucanEnemy);
    enemyList.add(jaguarEnemy);
    enemyList.add(snakeEnemy);
}
```



```java
public Path pathCreation(Enemy enemy) {
    Path path = new Path();
```

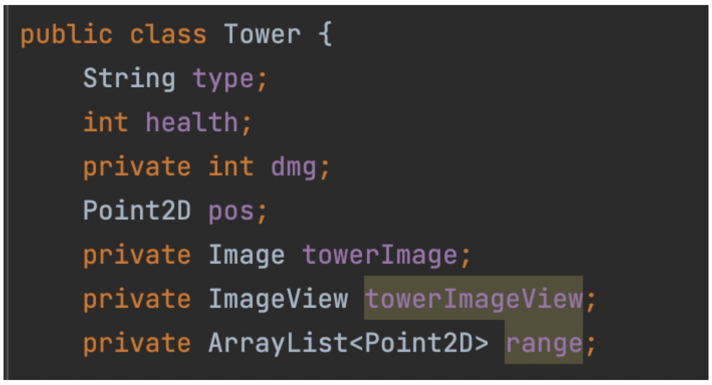


```java
public AnimationTimer moveEnemies(ArrayList<Enemy> enemies) {
```

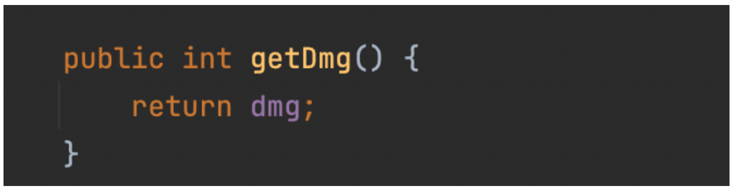


```java
public AnimationTimer attackEnemies() throws InterruptedException {
    AnimationTimer animationTimer = (l) → {
            int mon = Integer.parseInt(money.getText());
            for (Enemy enemy : enemyList) {
                if (enemy.getHealth() == 0) {
```

3: Information Expert (GRASP)



```
public class Tower {
    String type;
    int health;
    private int dmg;
    Point2D pos;
    private Image towerImage;
    private ImageView towerImageView;
    private ArrayList<Point2D> range;
```



```
public int getDmg() {
    return dmg;
}
```

The information expert principle states that we should assign a responsibility to the class that has the information necessary to fulfil the responsibility.

Before creating the Tower class, methods related to the Tower class were implemented in the Game Screen. Now, the Tower class is responsible for knowing its own information such as its damage.

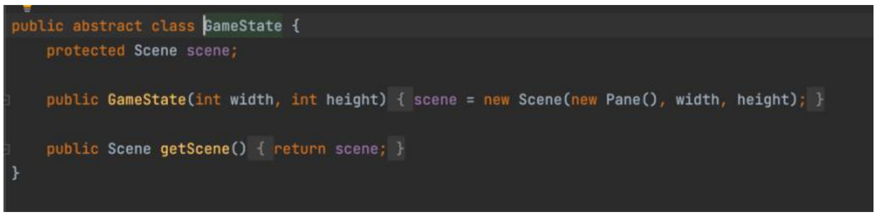
It is hence able to use a getter method to get the damage done.

4:Open/Closed Principle (SOLID)

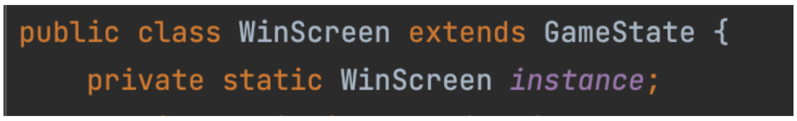
The open/closed principle states that the behaviour of a module should be open for extension and closed for modification.

The class ‘GameState’ allows follows the ‘open for extension’ principle as it many screens extend from this class and other screens could extend from this class in the future.

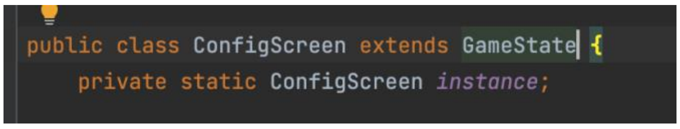
Examples of this include the ConfigScreen, WinScreen, and GameOverScreen.



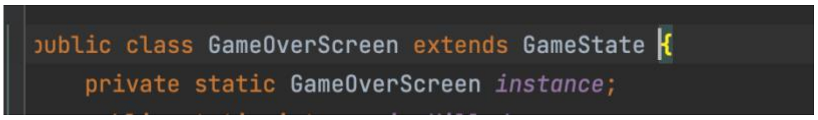
```
public abstract class GameState {
    protected Scene scene;

    public GameState(int width, int height) { scene = new Scene(new Pane(), width, height); }

    public Scene getScene() { return scene; }
}
```



```
public class WinScreen extends GameState {
    private static WinScreen instance;
```



```
public class ConfigScreen extends GameState {
    private static ConfigScreen instance;
```



```
ublic class GameOverScreen extends GameState {
    private static GameOverScreen instance;
```

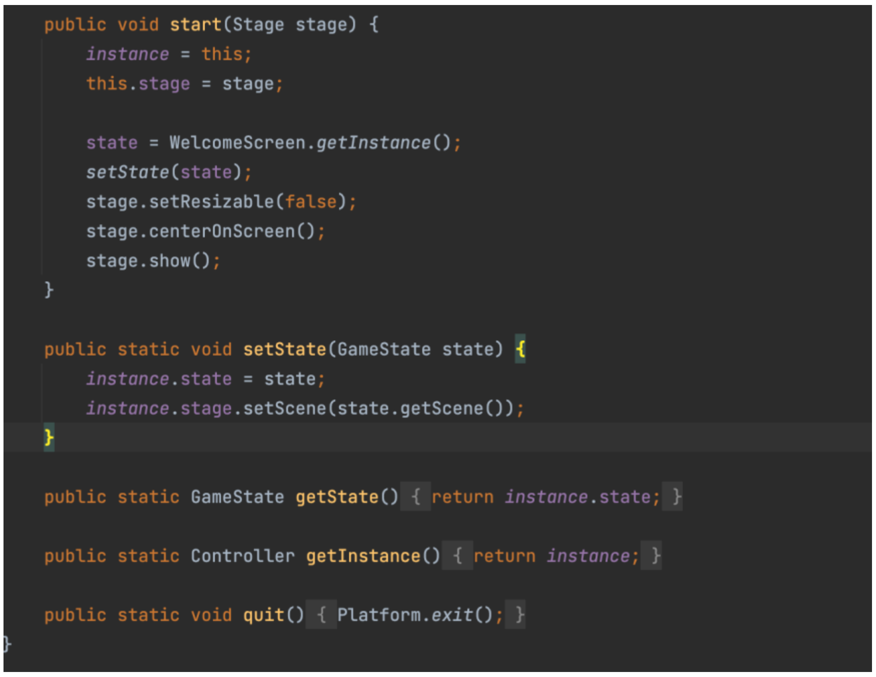
Adding extensions does not result in changes to the source code in GameState.

5: Controller (GRASP)

The Controller class in our game represents the overall system. It coordinates and controls activity without doing a lot of work itself.

The Controller class is a very short class containing only a start method, a quit method, and methods to get and set the state of the game.

The Controller performs system related invents such as creating an instance and initializing the first state of the game.



```
    public void start(Stage stage) {
        instance = this;
        this.stage = stage;

        state = WelcomeScreen.getInstance();
        setState(state);
        stage.setResizable(false);
        stage.centerOnScreen();
        stage.show();
    }

    public static void setState(GameState state) {
        instance.state = state;
        instance.stage.setScene(state.getScene());
    }

    public static GameState getState() { return instance.state; }

    public static Controller getInstance() { return instance; }

    public static void quit() { Platform.exit(); }
}
```

Every Screen class and Controller class hence relies on Controller when getting and setting the game instance and game state.

Extra Credit

Team 26 Semester Project Outline

Over these five milestones, students will create a Drawing App similar to MS Paint or Photoshop using Android Studio or JavaFX. The app will have the ability to create a canvas of a chosen size, and load and save images. A toolbar will contain all of the tools available for use. The toolbar will contain items such as paintbrushes, at least three shapes, and an eraser. Users will be able to transform the image in a variety of ways. They can flip it horizontally, vertically, and rotate it 90 degrees. Additionally, they will be able to invert the colors for the entire image. Users will be able to draw their own artwork or design, or edit an existing work. For more detailed requirements, see the project outline below.

M2

- Create a canvas creation screen
  - A way to create a blank canvas
  - Users can choose the dimensions of the canvas
- Blank Canvas
  - A blank screen where users will be able to draw
  - Must be the same dimensions as specified in the canvas creation screen
- Toolbar
  - Include icons for at least a paintbrush, eraser, and pencil
  - Clicking a tool highlights it
  - Does not have to be functional or connected yet
  - Displayed next to the canvas

M3

- Three types of paint brushes
  - Brushes must vary in texture or opacity
  - Each brush must be adjustable in size
- Color selection
  - Users should be able to choose any possible RGB color for their brush
- Eraser
  - Anything under the eraser should revert back to the original canvas color

M4

- Shape tools
  - Shapes can be placed and transformed on the canvas

  - Can be any color
  - Must have at least three types of shapes
- Save images
  - Images can be saved as JPEGs

M5

- Image color invert button
  - Inverts all RGB colors on the screen
- Flip
  - Horizontal and vertical flip
- Rotate
  - Rotates image and canvas 90 degrees

M6

- Screen after saving image
  - Create another image option
  - View Image
- Load Image
  - Load existing art from a JPEG
- One additional creative tool
  - Should impact the usage of the app in a significant way
  - Allows for a variety of new types of drawings or makes an existing drawing method easier
  - Must be complex (Ask TA if unsure)