Report - Assignment 3

Refactored code as when we found bad smells:

1. *GamePanel class -> GamePanelData class:* To refactor the code included in the GamePanel class and prevent it from being a God Class or Blob object and having a huge data clump at the start of the class, most of its primitive variables and objects were extracted and stored into a new data class - GamePanelData class. Each of the source code and test code classes were adjusted accordingly to use the GamePanelData class to access the variables. Java documentation was added for the same.

2. *CollisionChecker class:* To avoid a data clump and rather increase code understandability the variables of the *checkTile* method of this class were all bundled up in an object called *TileData* by making a nested class in this class and passing the entity object to the constructor. Java documentation was added for the same.

3. *GamePanelData class*: The newly constructed data class was refactored and unused data variables like worldWidth and worldHeight were deleted in order to improve the code efficiency and optimization.

4. *CollisionChecker class:* The *checkTile* method is refactored to avoid code duplication and thus the common if condition is extracted out of the switch-case statements to set the collision variable.

5. *CollisionCheckerClass*: Similar to the *checkTile* method, the if condition in *checkObject* that is used to set the collision and index variables is extracted out of the cases and placed at the end as a common condition to avoid code duplication and increase efficiency

6. *CollisionCheckerClass:* To avoid having unnecessary long methods, improve code efficiency and understandability, and reduce code duplication methods (*checkObject*, *checkEnemy*, *checkPlayer*) were refactored to call a new method (*setSolidArea*) that contained the solid Area assignment statements that were common to them. Java documentation was added for the same.

7. *CollisionCheckerClass:* To avoid having unnecessary long methods, improve code efficiency and understandability, and reduce code duplication methods (*checkObject*, *checkEnemy*, *checkPlayer*) were refactored to call a new method (*setVariablesEntity and setVariablesObject*) that contained the checking intersection statements common to them. Java documentation was added for the same.

8. *AssetSetterClass*: The class' *setObject* method was refactored to avoid code duplication and long method; and instead call new method *placeObject* to place object on map by setting variables (x and y coordinates).

9. *AssetSetterClass*: The class’ *setEnemy* method was refactored to avoid code duplication and long method; and instead call new method *placeEnemy* to place object on map by setting variables (x and y coordinates).
10. *KeyHandlerClass*: The class keyPressed use if for multiple times,to avoid the Code duplication, the class was refactored to use switch.
11. *StartScreenClass*: To avoid Unnecessary if/else or switch/case statements, the paintComponent inside if(check == true) can be changed to if(check).
12. *PauseAndResumeClass*: To avoid Unnecessary if/else or switch/case statements, The switch part of PauseAndResume draw.

```
<<======================================
switch(state){
          case "pause":
             image = pause;
              gp.start.paintComponent((Graphics) g2, false);
             break;
          case "resume":
             image = resume;
             break;
       }
======================================>>
```

can be modified as follows

```
<<======================================
       switch (state) {
          case "pause" -> {
             image = pause;
             gp.start.paintComponent((Graphics) g2, false);
          }
          case "resume" -> image = resume;
       }
======================================>>
```