1. Smell: Code Duplication

Commits: **2065a88c 9dd78530** “created wall detection class and updated to enhanced switch case”

In the game logic within the Level class, the player character and the moving enemy would have to check their position against the position of all the walls within the map. This code was explicitly written every time the character was required to check if there was a wall in the right, left, up, down directions. For example, if you wanted to check if the position to the left of the player was a wall, the code would look like the following: `playerPos[0]-1 >= 0 && map[playerPos[1]][playerPos[0]-1] != 1`

This was duplicated (with slight modifications) for each of the possible directions, and each of the possible moving entities. This resulted in code that was hard to read without the assistance of comments.

Solution: We created a WallDetection class that took the position of a character, either the player character or the moving enemy and the map encoding the position of the walls. This class would have four methods that would return whether a move in each of the four directions was valid. Now, the code above looks like the following: `WallDetection playerWallDetection = new WallDetection(map,playerPos);`

`playerWallDetection.validLeftMove()`

2. Smell: Lack of Documentation

Commits: **2a79571e 3b233920** “added documentations”

Solution: While writing our tests for phase 3, we had to write additional getters and setters to test specific features of the system. However, when we wrote these methods, we forgot to add documentation. We addressed this by adding Javadocs.

3. Smell: Long List of Method Parameters

Commit: **116ca7fc** “Addressed long parameter list of createButtons method”

Previously, our CreateButtons method asked for 5 parameters to initialize the button. We added a setButtonBounds method to help with setting the position and size of the button.

Solution: Added a helper method

4. Smell: Unused or Useless Variables

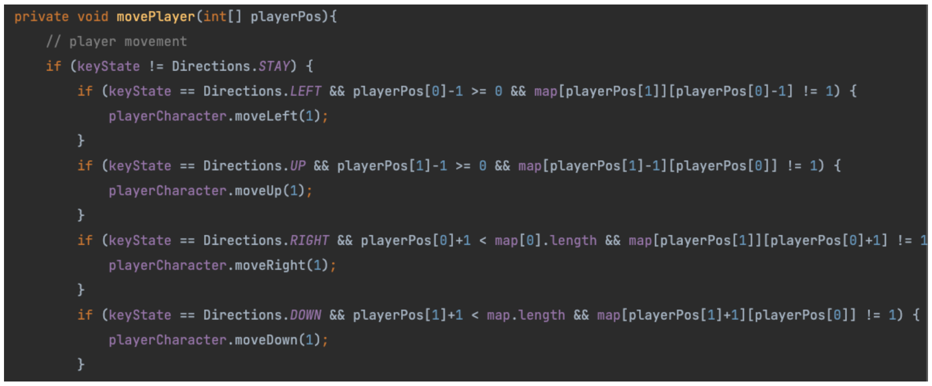
Commit: **b53f4469** “removed unused variables/parameters”

Solution: While refactoring our code in Phases 2 and 3, some variables were left unused. In the “Level” class we deleted `private LevelPlan levelPlans;` and `private TimerTask gameloop;` as well as one of the unused constructor parameters. Deleting these variables is beneficial as it reduces the change of an inadvertent error.

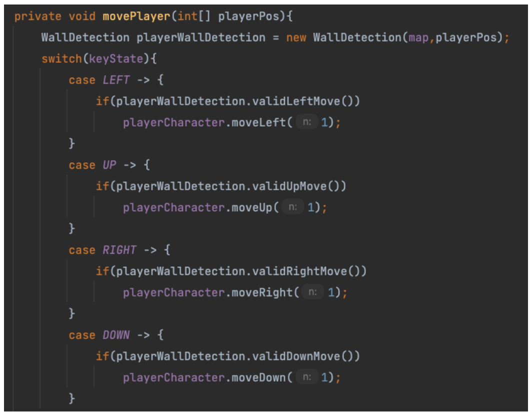
5. Smell: Long conditional statements

Commit: **bb2c2f21** “Replaced if/else with switch statements”

Before refactoring, some of the if statements had very long and confusing conditions.



```
private void movePlayer(int[] playerPos){
    // player movement
    if (keyState != Directions.STAY) {
        if (keyState == Directions.LEFT && playerPos[0]-1 >= 0 && map[playerPos[1]][playerPos[0]-1] != 1) {
            playerCharacter.moveLeft(1);
        }
        if (keyState == Directions.UP && playerPos[1]-1 >= 0 && map[playerPos[1]-1][playerPos[0]] != 1) {
            playerCharacter.moveUp(1);
        }
        if (keyState == Directions.RIGHT && playerPos[0]+1 < map[0].length && map[playerPos[1]][playerPos[0]+1] != 1
            playerCharacter.moveRight(1);
        }
        if (keyState == Directions.DOWN && playerPos[1]+1 < map.length && map[playerPos[1]+1][playerPos[0]] != 1) {
            playerCharacter.moveDown(1);
        }
```



```
private void movePlayer(int[] playerPos){
    WallDetection playerWallDetection = new WallDetection(map,playerPos);
    switch(keyState){
        case LEFT -> {
            if(playerWallDetection.validLeftMove())
                playerCharacter.moveLeft( n: 1);
        }
        case UP -> {
            if(playerWallDetection.validUpMove())
                playerCharacter.moveUp( n: 1);
        }
        case RIGHT -> {
            if(playerWallDetection.validRightMove())
                playerCharacter.moveRight( n: 1);
        }
        case DOWN -> {
            if(playerWallDetection.validDownMove())
                playerCharacter.moveDown( n: 1);
        }
```

Solution: We addressed this by extracting the condition relating to the KeyState into a switch case. The other portion of the conditional statement would be placed in a nested if statement. This improved clarity and readability.

6. Smell: Classes that are too large and/or try to do too much

Commit: **df265d14** “Extracted game logic from "Level" class into new "GameLoop" class”
Previously, our “Level” screen class contained all of the logic for the game as well as both the responsibilities of painting the entity objects on the frame and responding to the user’s button clicks. Normally, the “Screen” subclasses are assigned the responsibilities of painting on the frame and responding to the button clicks.
Solution: We separated the concerns of a “Screen” class and those of game loop by extracting the game logic and placing it into a new GameLoop class. The GameLoop would communicate with the Level class in the form of a single update method that updated the entities that were to be drawn. This greatly increased cohesion while only slightly increasing coupling.

7. Smell: Unjustified Use of Primitives

Commit: **f3f97e04 ac35848d 0c047128** “refactored unjustified use of primitives” and “replaced Point with Position”

We used an int[2] to store the x and y coordinates of entity objects. This led to code that was confusing to read. To access the x-coordinate of the entity, you would call position[0]. Accessing the y-coordinate can be done through position[1]. At first glance, it is confusing what you are modifying when you call the x and y coordinates.
Solution: We created a class called “Position” to encode the position. This class extended the Point class and also contained other methods for changing the position. Now accessing the x and y coordinates could be done through position.x and position.y. The new methods also removed the need for some switch cases in the GameLoop class .