1.

- The implementation of the game loop had low cohesion and high coupling. Game logic functions that were called only in the level screen existed as methods in entity classes. These functions themselves used too many arguments: both the PlayerCharacter object and its position member, where only the position member was necessary to perform the function.
- These game logic functions were removed from the entity classes and implemented as helper functions in the same class as the game loop.
- commit: 0cdd2ec

```
    playerCharacter.testWall(walls.getWallPosition());
    //test whether basic objective is at PlayerCharacter's location
    basicObjectives.testObjectives(playerCharacter.getPosition(), playerCharacter);
    //test whether bonus objective is at PlayerCharacter's location
    bonusObjectives.testObjectives(playerCharacter.getPosition(), playerCharacter);
    //test whether the player is trying to go across the wall
}
//refresh for next tick player movement
keyState = 0;

//test whether enemies are contacting player
stationaryEnemies.testEnemies(playerCharacter.getPosition(), playerCharacter);

//if the player encounters moving enemies, then lose screen shows
if (movingEnemies.testMovingEnemies(playerCharacter.getPosition(), playerCharacter)) {
    CardLayout cl = (CardLayout) screenContainer.getLayout();
    cl.show(screenContainer, "LOSE");
    timer.cancel();
    bonusObjectives.stopCounting();
}
```

```
@Override
public void run() {
    //player position before tick move
    Position playerPos = playerCharacter.getPosition();

    //test for win
    testForWin(playerPos);

    //increment timer
    incrementTimer();

    //enemy movement
    moveEnemy(playerPos);

    //player movement
    movePlayer(playerPos);

    //test for lose
    testForLose(playerPos);

    //reset key state for next player movement input
    keyState = Directions.STAY;

    //new player position after tick move
    playerPos = playerCharacter.getPosition();

    //various checks against player's new position
    testForContact(playerPos, stationaryEnemies);
    testForContact(playerPos, basicObjectives);
    testForContact(playerPos, bonusObjectives);

    //check if bonus is still there
    checkBonus();

    // paint new game state
    update();
}
};
```

2.

- We found that there are multiple moving functions (“moveLeft”, “moveUp”, “moveRight”, “moveDown”) in “PlayerCharacter” and “MovingEnemy” and their codes are similar.
- We want to combine those functions into one function. So we refactor those functions into one function called the “move” function. It takes a direction parameter to decide the direction of the movement.
- commit: 8bc5fa56

```
public void moveLeft(int n) { position[0] = position[0] - n; }

/**
 * @param n number of grid positions to move
 */
public void moveUp(int n) {
    position[1] = position[1] - n;
}

/**
 * @param n number of grid positions to move
 */
public void moveRight(int n) { position[0] = position[0] + n; }

/**
 * @param n number of grid positions to move
 */
public void moveDown(int n) {
    position[1] = position[1] + n;
```

```
public void move(int n, Directions d) {
    switch(d) {
        case LEFT:
            position[0] = position[0] - n;
            break;
        case UP:
            position[1] = position[1] - n;
            break;
        case RIGHT:
            position[0] = position[0] + n;
            break;
        case DOWN:
            position[1] = position[1] + n;
            break;
    }
}
```

3.

- In “WallDetection”, we found that the methods of testing whether the cell to some directions of the given position is a wall are quite similar.
- We want to erase those similarities. So we decided to write a function that can perform all those operations and substitute those methods. We make a “validMove” function and take a direction parameter and we use a switch case to judge the directions and perform the corresponding actions.
- commit: 4ead5d98

```
public Boolean validLeftMove(){
    return position[0]-1 >= 0 && map[position[1]][position[0]-1] != 1;
}

/**
 * Method to test if the cell above the given position is a wall
 * @return a boolean whether there is a wall or not
 */
public Boolean validUpMove(){
    return position[1]-1 >= 0 && map[position[1]-1][position[0]] != 1;
}

/**
 * Method to test if the cell to the right of the given position is a wall
 * @return a boolean whether there is a wall or not
 */
public Boolean validRightMove(){
    return position[0]+1 < map[0].length && map[position[1]][position[0]+1] != 1;
}

/**
 * Method to test if the cell below the given position is a wall
 * @return a boolean whether there is a wall or not
 */
public Boolean validDownMove(){
    return position[1]+1 < map.length && map[position[1]+1][position[0]] != 1;
```

```
public Boolean validMove(Directions d) {
    switch(d) {
        case LEFT:
            return position[0]-1 >= 0 && map[position[1]][position[0]-1] != 1;
        case UP:
            return position[1]-1 >= 0 && map[position[1]-1][position[0]] != 1;
        case RIGHT:
            return position[0]+1 < map[0].length && map[position[1]][position[0]+1] != 1;
        case DOWN:
            return position[1]+1 < map.length && map[position[1]+1][position[0]] != 1;
        default:
            return true;
    }
}
```

4.

- After we finished the “validMove” function. We changed all previous functions about “validMove” to the new function we modified. We found that there is an unnecessary switch/case statement about “valiMove” in the “movePlayer” function in “GameLoop”.
- The previous switch/case is used to check the value of “keyState” to judge the direction and then perform the previous corresponding “validMove”. But after we rewrite the “validMove” function, this switch/case can be easily substituted by the new function because we already have the switch/case statement in the new function.
- commit:4ead5d98

```
switch(keyState){
    case LEFT -> {
        if(playerWallDetection.validLeftMove())
            playerCharacter.move(1, Directions.LEFT);
    }
    case UP -> {
        if(playerWallDetection.validUpMove())
            playerCharacter.move(1, Directions.UP);
    }
    case RIGHT -> {
        if(playerWallDetection.validRightMove())
            playerCharacter.move(1, Directions.RIGHT);
    }
    case DOWN -> {
        if(playerWallDetection.validDownMove())
            playerCharacter.move(1, Directions.DOWN);
    }
```

```
if(playerWallDetection.validMove(keyState)) {
    playerCharacter.move(1, keyState);
}
```

5.

- The Frame class consisted of a small amount of code in one constructor function to set up the window, so it could be merged with ScreenManager to consolidate all the display responsibilities in one class.
- commit: d12f4a3

6.

- In the function “moveEnemy” in “GameLoop”, there is a long and similar code to calculate and check the best moving direction of moving enemies.
- We construct a function called “checkBestMove” to check the best moving direction of moving enemies. It takes the enemy position and player position as the parameters and uses them to do some calculation to check the best moving direction.
- commit: d12f4a31

```
if (enemyWallDetection.validMove(Directions.LEFT)) {
    testDistance = Math.abs(enemyPos.x - 1 - playerPos.x) + Math.abs(enemyPos.y - playerPos.y);
    if (testDistance < bestDistance) {
        bestMove = Directions.LEFT;
        bestDistance = testDistance;
    }
}
//up
if (enemyWallDetection.validMove(Directions.UP)) {
    testDistance = Math.abs(enemyPos.x - playerPos.x) + Math.abs(enemyPos.y - 1 - playerPos.y);
    if (testDistance < bestDistance) {
        bestMove = Directions.UP;
        bestDistance = testDistance;
    }
}
//right
if (enemyWallDetection.validMove(Directions.RIGHT)) {
    testDistance = Math.abs(enemyPos.x + 1 - playerPos.x) + Math.abs(enemyPos.y - playerPos.y);
    if (testDistance < bestDistance) {
        bestMove = Directions.RIGHT;
        bestDistance = testDistance;
    }
}
//down
if (enemyWallDetection.validMove(Directions.DOWN)) {
    testDistance = Math.abs(enemyPos.x - playerPos.x) + Math.abs(enemyPos.y + 1 - playerPos.y);
    if (testDistance < bestDistance) {
```

```
            // check the bestMove Direction of the movingEnemy
            Directions bestMove = checkBestMove(enemyPos, playerPos);

            //check for lose before executing enemy movements
            //calculate next player position
@@ -394,6 +359,33 @@ public class GameLoop {
        }
    }

    /**
     * check the bestMove of movingEnemy
     */
    public Directions checkBestMove(Position enemyPos, Position playerPos) {
        int bestDistance = Math.abs(enemyPos.x - playerPos.x) + Math.abs(enemyPos.y - playerPos.y);
        int testDistance = 100;
        Directions bestMove = Directions.STAY;
        WallDetection enemyWallDetection = new WallDetection(map, enemyPos);
        for (Directions d: new Directions[] {Directions.LEFT, Directions.UP, Directions.RIGHT,
Directions.DOWN}) {
```