Dylan Young, Kaung Si Thu, Maisha Supritee Chowdhury, Sajandeep Toor

CMPT 276 Phase 3 - Group 19

Features that needed to be Unit Tested :

CharacterModel (100% Line Coverage, Branch Coverage 95%):

- CharacterModel constructor function
- getPosition() - Returns position of character
- setPosition(Position pos) - Sets the Position argument as the position of the character
- setPosition(int x, int y) - Sets the arguments as x and y coordinates of the Position of the character
- moveForward() - Moves the character one unit up, changing the Position
- moveBackward() - Moves the character one unit down, changing the Position
- moveLeft() - Moves the character to the left, changing the Position
- moveRight() - Moves the character to the right, changing the Position

Enemy (83% Line Coverage, Branch Coverage 76%):

- move() - Moves the enemy to the next given position closes to the player

Player (100% Line Coverage, Branch Coverage 100%) :

- getInstance() - Follows singleton pattern to create a player instance. Initialized the player's position and score with default values
- getScore() - Returns score of the player
- setScore(int score) - Sets the argument score as the score of the player
- updateScore(int score) - Updates the score of the player by adding the argument to score
- reset() - Returns the player to its original state, i.e. resets position and score

Entities (97% Line Coverage, Branch Coverage 100%) :

- getInstance() - This method returns the instance of GameInput (Singleton)
- getRewards()/Enemies/Traps - Returns all rewards/enemies/traps
- getReward(Position position)/Enemy/Trap - Checks if there is a reward/enemy/trap at a position and returns it
- getReward(int i)/Enemy/Trap - Gets the reward/enemy/trap at index i
- containsReward(Position position)/Enemy/Trap - Checks if a position contains a reward/enemy/trap in the maze
- getRewardCount() - Gets the number of rewards in the maze
- getNumBonusRewards() - Gets the number of bonus rewards in the maze
- setNumBonusRewards(int num) - Sets the number of bonus rewards in the maze
- updateBonusRewardsCollected(int num) - Updates the number of bonus rewards collected by num
- getBonusRewardsCollected() - Returns the number of bonus rewards collected
- addReward(Reward reward) - Adds a reward to the list of rewards
- addEnemy(Enemy enemy) - Adds an enemy to the list of rewards
- addTrap(Trap trap) - Adds a trap to the list of rewards
- removeReward(Reward reward)/Enemy/Trap - Removes a reward/enemy/trap from the list of rewards

- clear() - Removes all entities from the game. Entities are traps, enemies and rewards

EntitiesGenerator(92% Line Coverage, Branch Coverage 88%) :
- getInstance() - This method returns the instance of EntityGenerator (Singleton)
- generateEntities() - Generates all entities in the game, such as traps, rewards, and enemies and adds them to the maze
- generateBonusReward() - Generates a bonus reward

GameInput (100% Line Coverage, Branch Coverage 100%):
- getInstance() - This method returns the instance of GameInput (Singleton)
- movePlayer() - Move the player in the direction specified by the movement. Checks if there is a collision with a wall and acts accordingly
- movePlayer() - Moves the player in the direction most recently specified by the player.
- addMovement() - Adds the given move to the list of moves
- removeMovement() - Removes the given move from the list of moves
- checkMovement() - Checks if the list contains a move
- resetMovement() - Resets the player's movement inputs.

Timer (93% Line Coverage, Branch Coverage 100%):
- getInstance() - This method is used to get the instance of the Timer class (Singleton)
- updateTime() - This method sets the time (in seconds) elapsed since the start of the game
- getTimeElapsed() - This method returns the time (in seconds) elapsed since the start of the game
- reset() - This method resets the timer to 0

Leaderboard (79% Line Coverage, Branch Coverage 91%):
- getInstance() - Returns singleton leaderboard Instance
- addPlayerScore() - Adds the playerScore argument to the Leaderboard
- getPlayerScore() - Returns player name and score at the index argument
- toString()
- getLeaderboardSize() - Return the number of PlayerScores in the leaderboard
- getMinimumScore() - Return the minimum score of a player in the leaderboard

PlayerScore (100% Line Coverage, Branch Coverage 100%):
- getName() - Returns the name of the player
- setName(String name) - Sets the argument as the name of the player
- getScore() - Returns the score of the player
- setScore(int score) - Sets the argument as the score of the player
- toString()

Cell (78% Line Coverage, Branch Coverage 100%):
- cell() - Represents a cell in a maze. Cell type is defaulted to WALL.
- getPosition() - Returns the position of the cell

- getCellType() - Returns the cell type of current cell
- setCellType(Celltype celltype) - Changes the content of the cell
- setEmpty() - Changes the cell type to empty/CellType.PATH
- setWall()/Trap/Reward - Changes cell type to wall/trap/reward
- isWall()/Empty/Start/End/Trap - Returns true if the cell is a wall/empty/start/end/trap, and false otherwise
- getX()/getY() - Returns the x/y position of the cell
- toString()

Maze (94% Line Coverage, Branch Coverage 92%):
- getInstance() - Returns an instance of the maze
- getCell(int x, int y) - Returns a cell at position (x,y)
- getCellPos()
- isWall(Position position)/Trap/Path/End - Returns true if the cell at position is a wall/trap/path/end
- isSolvable() - Checks if the maze is solvable
- getAdjacentPositions() - Returns all adjacent positions of the current position, looking up, down, right, left. These positions don't have to be in the maze, could be out of bounds.

BonusReward (100% Line Coverage, Branch Coverage 100%):
- getEndTime() - Checks if it gets the expected end times
- setEndTime() - Checks if if gets the expected start time

Reward (100% Line Coverage, Branch Coverage 100%):
- getPosition() - Checks if the position matches expected
- setPosition() - Checks if after setting position, the position matches expected.
- getPoints() - Checks if the points match expected
- setPoints() - Checks if after setting points, the points match expected.

Trap (72% Line Coverage, Branch Coverage 66%):
- getPosition() - Check if the position returned is as expected
- setPosition() - Check if after setting position, the position matches expected
- getTrapType() - Check if the type return is as expected
- setTrapType() - Check if after setting the type, the type match expected

Functions (93% Line Coverage, Branch Coverage 100%):
- updatePosition() - Update position based off movement enum
- getRandomNumber() - Random number in some range
- getRandomPosition() - Random position in some range
- validatePosition() - Checks if a given position/coordinates is in the maze

Position (100% Line Coverage, Branch Coverage 100%):
- testPositionGetters():: Tests if x and y matches expected.
- testPositionSetters(): Test if after setting the x and y matches expected.

- testPositionCopyConstructor(): Tests if after creating a copy, the positions have the same x and y.
- testPositionEquals: Tests if 2 identical positions are equal.
- testPositionNotEquals: Test if 2 positions that are different are not equal.
- testPositionEqualsObject: Tests if a position and an object are not equal.

Line and Branch Coverage of your Tests

- See figure 4. We achieved coverage. For game our coverage was lower, this is because GameController didn't need to be tested since it just set booleans and started threads.

Learnings from Writing Tests

- None of us have used JUnit or written unit tests before, although we were at least exposed to tests in some assignments in classes such as CMPT 125, CMPT 225 to test our assignment code but we never wrote tests like this before.
- Realizing the importance of Javadocs or documentation in general and stating assumptions. We decided to test code we didn't write ourselves to avoid designer bias and this is an issue that came up. At times we would update Javadocs with those assumptions so it's more clear or make changes to the method itself to pass the test.

Test Code Responsibilities

- Dylan - RewardTest, BonusRewardTest, TrapTest, TimerTest, EnemyTest
- Kuang - PlayerScoreTest, LeaderboardTest, EntitiesGeneratorTest, GameInputTest
- Maisha - CellTest, CharacterModelTest, PlayerTest, EntitiesTest
- Sajandeep - Refactored GameController to subclasses Entities, EntitiesGenerator, GameInput, GameLogic, Timer and GameController, tested Maze and Position.

Changes to Production Code

- Refactored GameController to subclasses Entities, EntitiesGenerator, GameInput, GameLogic, Timer and GameController. Splitting the one big class helped make the code more neat, organised and more understandable. Different members handled testing different subclasses which made it easier and saved time.
- In the character package, a reset() method was added to the Player to enable resetting the player to the start position, with no score.
- Cell, CellType and Maze classes were moved to the maze package to allow classes in the Game package to use protected methods.
- Added getTraps() method in the Entities class to easily access the traps list and make testing smoother.
- Added a secondary constructor function and Clone() method in LeaderBoard class for testing

Identify Bugs and Fixes

- To document bugs, we used the issues tab on Gitlab, this allowed us to document the issue and mention commits that have worked on a bug or have fixed a bug. This allowed

us to stay organized. We wrote a short description of the bug, how to reproduce and some possible reasons, sometimes small code snippets. (See figure 1)

- One major bug we found was with enemy movement
    - The issue with the movement was at points the enemy will “get stuck” and move back and forth randomly. It was difficult to pinpoint what was going wrong since there were multiple method calls and our solution to move the enemy was a recursive method.
    - We tried changing things and fixing by looking at the code by inspection, this didn’t end up working, we were unable to find the problem.
    - Next we decided to run unit tests for all the methods involved after setting them to public temporarily.
    - What we found was our distance method wasn’t actually returning the shortest distance as we thought, this made sense since it was using depth first search which doesn’t find the shortest distance to a certain position.
    - After discovering that, we decided to ditch the distance method entirely as it was largely inefficient as we had to calculate the distance of all possible moves the enemy could make and a lot of calculations would be repeated. They also had to be repeated if the player didn’t move which wasn’t ideal.
    - The solution to this problem was finding the shortest path using breadth first search, this path would be an ArrayList of Positions and traversing the path in the move method. The path would only be regenerated if the player moves and regenerates on a separate thread for performance reasons.

- Another bug we had was “Phantom Rewards and Traps”: Sometimes when collecting a reward or stepping on a trap, the game would crash with a NullPointerException. The problem we had was with thread synchronization. Another thread accessing the reward at the same time would result in it being null. To fix this we checked null whenever we tried to access the reward and changed our iterators to traditional for loops instead of iterator loops since hasNext() could break if something was removed in another thread.
- One small thing was forgetting to reset the timer after starting a new game. To fix this we set the timer to 0 when the startGame method in GameController was invoked.
- Fixed generateBonusReward() method in EntitiesGenerator class to update the numBonusRewards attribute from Entities class

Quality of Code

- Ensuring every member had similar formatting for their code, helped keep the code neat.
    - We followed Java Code Standards in our formatting (except in the case of switch statements, which we argued should be indented).
- Adding appropriate comments describing each test method enabled everyone to understand each others’ code.
- We discussed every class in depth while creating the test code, to ensure we were covering all the possible outcomes, scenarios and loopholes. This helped create test cases that covered every situation a method might create.

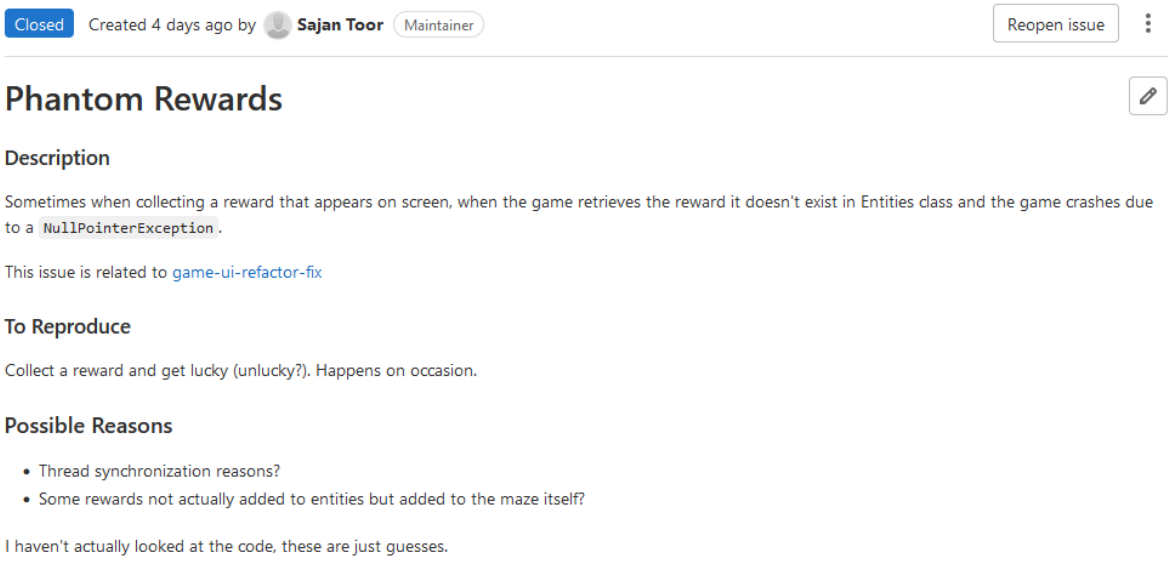
- Readability of the code was also very important and spaghetti code was refactored into more readable code.
- Example: Figure 2 was very difficult to read and understand, it was transformed to Figure 3, something more readable.

Important Findings

- Singletons, although helpful for creation, cannot be tested easily.
- While some test methods might work fine when run individually, they might not work the same way when running the test class as a whole. Therefore, test code needs to be written in a way which will work regardless of being run on its own or with the class as a whole.

Figures

Figure 1



Closed Created 4 days ago by **Sajan Toor** Maintainer Reopen issue

Phantom Rewards

Description

Sometimes when collecting a reward that appears on screen, when the game retrieves the reward it doesn't exist in Entities class and the game crashes due to a `NullPointerException`.

This issue is related to game-ui-refactor-fix

To Reproduce

Collect a reward and get lucky (unlucky?). Happens on occasion.

Possible Reasons

- Thread synchronization reasons?
- Some rewards not actually added to entities but added to the maze itself?

I haven't actually looked at the code, these are just guesses.

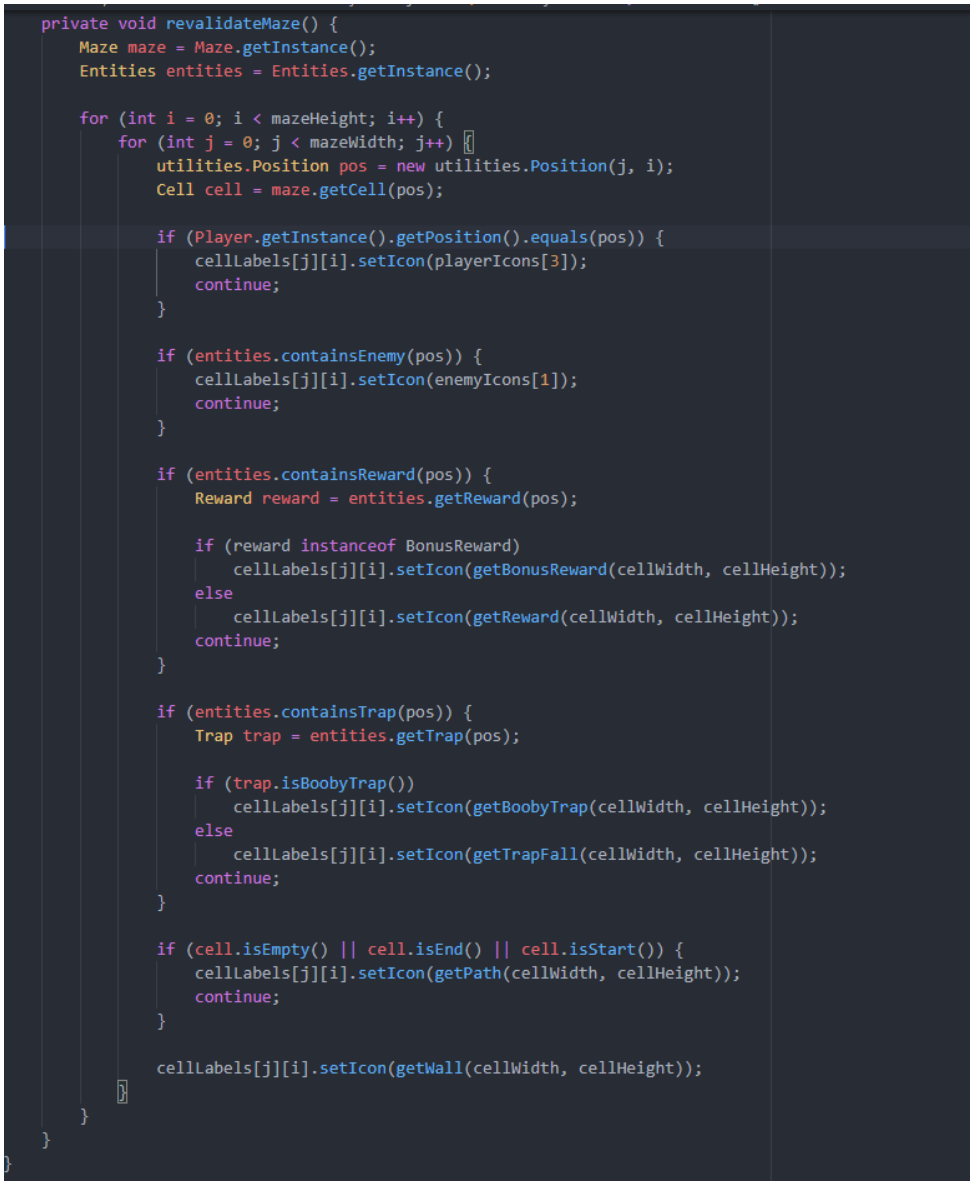
https://csil-git1.cs.surrey.sfu.ca/cmpt276f21_group19/project/-/issues/8

Figure 2

```
private void revalidateMaze() {
    for (int i = 0; i < mazeHeight; i++) {
        for (int j = 0; j < mazeWidth; j++) {
            if (GameController.getInstance().getPlayer().getPosition().getX() == j && GameController.getInstance().getPlayer().getPosition().getY() == i) {
                cellLabels[j][i].setIcon(playerIcons[3]);
            } else if (GameController.getInstance().containsEnemy(j, i)) {
                cellLabels[j][i].setIcon(enemyIcons[1]);
            } else if (GameController.getInstance().containsReward(j, i)) {
                Reward reward = GameController.getInstance().getReward(j, i);
                if (reward instanceof BonusReward) {
                    cellLabels[j][i].setIcon(getBonusReward(cellWidth, cellHeight));
                } else {
                    cellLabels[j][i].setIcon(getReward(cellWidth, cellHeight));
                }
            } else if (GameController.getInstance().containsTrap(j, i)) {
                Trap trap = GameController.getInstance().getTrap(j, i);
                if (trap.getTrapType() == TrapType.BOOBYTRAP) {
                    cellLabels[j][i].setIcon(getBoobyTrap(cellWidth, cellHeight));
                } else {
                    cellLabels[j][i].setIcon(getTrapFall(cellWidth, cellHeight));
                }
            } else if (Maze.getInstance().getCell(j, i).getCellType() == CellType.PATH || Maze.getInstance().getCell(j, i).getCellType() == CellType.END || Maze.getInstance().getCell(j, i).getCellType()
            == CellType.START) {
                cellLabels[j][i].setIcon(getPath(cellWidth, cellHeight));
            } else {
                cellLabels[j][i].setIcon(getWall(cellWidth, cellHeight));
            }
        }
    }
}
```

https://csil-git1.cs.surrey.sfu.ca/cmpt276f21_group19/project/-/blob/2a3b699f4615ef4d4f2d65b6715aff32f12b8c36/src/main/java/ui/Screens/GamePlayScreen.java#L156

Figure 3



```
    private void revalidateMaze() {
        Maze maze = Maze.getInstance();
        Entities entities = Entities.getInstance();

        for (int i = 0; i < mazeHeight; i++) {
            for (int j = 0; j < mazeWidth; j++) {
                utilities.Position pos = new utilities.Position(j, i);
                Cell cell = maze.getCell(pos);

                if (Player.getInstance().getPosition().equals(pos)) {
                    cellLabels[j][i].setIcon(playerIcons[3]);
                    continue;
                }

                if (entities.containsEnemy(pos)) {
                    cellLabels[j][i].setIcon(enemyIcons[1]);
                    continue;
                }

                if (entities.containsReward(pos)) {
                    Reward reward = entities.getReward(pos);

                    if (reward instanceof BonusReward)
                        cellLabels[j][i].setIcon(getBonusReward(cellWidth, cellHeight));
                    else
                        cellLabels[j][i].setIcon(getReward(cellWidth, cellHeight));
                    continue;
                }

                if (entities.containsTrap(pos)) {
                    Trap trap = entities.getTrap(pos);

                    if (trap.isBoobyTrap())
                        cellLabels[j][i].setIcon(getBoobyTrap(cellWidth, cellHeight));
                    else
                        cellLabels[j][i].setIcon(getTrapFall(cellWidth, cellHeight));
                    continue;
                }

                if (cell.isEmpty() || cell.isEnd() || cell.isStart()) {
                    cellLabels[j][i].setIcon(getPath(cellWidth, cellHeight));
                    continue;
                }

                cellLabels[j][i].setIcon(getWall(cellWidth, cellHeight));
            }
        }
    }
}
```

https://csil-git1.cs.surrey.sfu.ca/cmpt276f21_group19/project/-/blob/master/src/main/java/ui/Screens/GamePlayScreen.java#L177

Figure 4

Element	Class, %	Method, %	Line, %
character	100% (4/4)	95% (22/23)	84% (108/1...
com			
game	46% (6/13)	62% (64/102)	53% (217/4...
images			
java			
javax			
jdk			
leaderboard	100% (2/2)	94% (17/18)	81% (76/93)
main	100% (2/2)	100% (2/2)	100% (5/5)
maze	100% (4/4)	95% (44/46)	91% (168/1...
META-INF			
netscape			
org			
reward	100% (6/6)	87% (21/24)	88% (48/54)
sun			
toolbarButtonGraphics			
ui	0% (0/32)	0% (0/98)	0% (0/707)
utilities	80% (4/5)	100% (20/20)	96% (61/63)