Contents

Introduction 2
=== September 24th, 2020 - started @ 9:32PM 2
=== September 25th, 2020 – Started @ 1:00PM 2
=== September 26th, 2020 – Started @ 8:40PM 3
=== September 27th, 2020 – started @ 2:30PM 4
=== September 28th, 2020 – started @ 1:00 PM 6
=== September 29th, 2020 – started @ 3:00 PM 7
=== September 30th, 2020 – started @ 3:00 PM 9
=== October 1st, 2020 – started @ 12:00 PM 12
=== October 2nd, 2020 – started @ 2:30 PM 14
=== October 3rd, 2020 – started @ 10:30 AM 16
=== October 4th, 2020 – started @ 3:00 PM 17
=== October 5th, 2020 – started @ 5:00 PM 20
=== October 6th, 2020 – started @ 1:00 PM 21
=== October 7th, 2020 – started @ 2:30 PM 24

Introduction

This project focuses on replicating a Maze Runner Game. This project intends on establishing an interactive and entertaining game experience for all ages.

=== September 24th, 2020 - started @ 9:32PM

Started researching for OOP java projects that interest me. Estimate: 20 minutes

@ 9:40PM after my research, I have chosen an interactive ATM machine program that allows users to interact with the program like a real-life ATM machine.

10:00 PM after some consideration, I have changed my plans to create a game using TUI, since I feel like an ATM machine program will not be complex enough and not that interesting. I will be making a Maze Runner game.

10:10 PM initiated my proposal for the assignment: “With technology advancing at a rapid pace, the price of laptops have increased, this can be hard for people who simply want to play games. Most games also require complex configuration, large download files and demand high specifications to run. I have designed a game that is easy to implement and run, which is also able to run on low specification hardware and most importantly… its fun!”

Stopped @ 10:20 PM

=== September 25th, 2020 – Started @ 1:00PM

Started brainstorming about the features and functionalities of the application. Estimated: 20 minutes

1:30PM Initiated the **features and functions** of my application.

- The user will be prompted by a welcome screen
- The user will be asked to input the maze height and the maze width
- After inputting the information, the user will be greeted with a text regarding a character named Casey, which is a maze runner trapped in a maze. The user will be given instructions on how to play the game.
- After reading the instructions the user will need to press ENTER to continue onto the game
- A grid will then appear, where users can navigate around the grid trying to find the objective using arrow keys
- Information such as keys found, hint to the nearest key, the current visibility and health will be displayed
- Users will have limitations on where they can navigate to, when they hit a boundary of a wall they will be reminded that they cannot walk through a wall,
- As users move around the maze, their health deceases. All users start off with 250 health and duplicates for each move they make.
- Users would need to find 2 keys in order to get out of the maze.
- The larger the maze, the more difficult it gets!

Stopped @ 2:00 PM

=== September 26th, 2020 – Started @ 8:40PM

Started thinking about what classes I want to create for my project. Estimate: 20 minutes

9:00 PM decided on the packages I want to use for my project. The project will involve 4 classes.

- a main package where the program starts its execution. This package will have a Launcher class.
- a main.game package which controls the display, the map, the different levels of games and keeps track of the progress of the game. The main.game package will have the following classes:
  - Display
  - Game
  - GameMap
  - LevelOne
  - LevelTwo
  - ProgressListener
  - main.game_object package which focuses on the character, elements of the game such as the fog, torch, the movement of the character and what happens when a user exits. This package will contain the following classes:
  - Exit
  - GameObject
  - Key
  - Movable
  - Player
  - Position
  - Torch
- Main.maze generator package focuses on an algorithm for maze generation. This package will contain a MazeGenerator class.

10:00 PM trying to research a way to output elements of the game to the console.

10:10 PM After some research, I have realised that I can utilize a 2D array to store characters representing the game map with maze and game objects. I will create a Display class which is used for encapsulating methods required for printing things to the screen

Here is a link that helped me, where this person used a 2D array to develop a chest game.

https://gamedev.stackexchange.com/questions/138697/2d-array-map-java-lost-in-map

10:20 PM as I think more about the game, I would need to initialize variables for the fog, the player, the height and width of the game since that is what will be asked upon program execution.

```
private static final char FOG = '#';
    private final char[][] map;
```

```
    private final Player player;
    private final int height;
    private final int width;
```

Stopped @ 10:25 PM

=== September 27th, 2020 – started @ 2:30PM

Today I will be making a method that utilises the use of ArrayList and keys to update the game map and statistics when printed onto the console. Estimated: 30 minutes

The game map can be printed in two ways based on code. If the visibility mode is equals to true, a partially covered game map will be generated and displayed, else a fully visible game map will be displayed.

```
void update(ArrayList<Key> keys) {
        clearScreen();
        char[][] displayMap = (Game.isVisibilityMode) ? createCoveredMap() : map;
```

Its important to note that this method is meant to be run only on level 1 since keys will only appear on level 1.

I will then use a for loop to produce rows in the map, and a switch .. case.. condition to update the keys, the key hint, visibility and health every time the map is updated which is every time a user makes a move.

```
void update(ArrayList<Key> keys) {
        clearScreen();
        char[][] displayMap = (Game.isVisibilityMode) ? createCoveredMap() : map;

        for (int row = 0; row < displayMap.length; row++) {
            System.out.print(new String(displayMap[row]));
            switch (row) {
                case 2:
                    System.out.printf("\tKey(s) collected: %d/%d\n",
LevelOne.NUMBER_OF_KEYS - keys.size(), LevelOne.NUMBER_OF_KEYS);
                    break;
                case 4:
                    System.out.printf("\tHint to the nearest key: %s\n",
getNearestKeyHints(keys));
                    break;
                case 6:
                    System.out.printf("\tVisibility: %d blocks\n",
player.getVisibility());
                    break;
                case 8:
                    System.out.printf("\tHealth: %d/%d\n", (player.isDead()) ? 0 :
player.getHealthLevel(), Player.INITIAL_HEALTH_LEVEL);
                    break;
                default:
                    System.out.println();
            }
        }
    }
```

3:00PM I will then need to think about how to provide hints to the user on where the closest key object will be just in case they get lost based on their current position.

3:05PM I will create a private method that gets the nearest key hints passing the ArrayList and keys. This method will return a string object representing the direction (hints) to the nearest key object from the player position or a '-' symbol if all keys have been collected.

I will use an if .. else … condition to check if the size of keys are equal to 0, which would mean the keys are found and have been cleared to 0. If so, it will print an "-" symbol to let the user know that all keys are found.

If the size of the keys are equal to one, a method will be called (getDirectionHints) which Returns a String object representing the direction hints to this Key object from the specified player's position.

Else a for loop will be used to determine the distance of the player to the key.

```
private String getNearestKeyHints(ArrayList<Key> keys) {
        if (keys.size() == 0) {
            return "-";
        } else if (keys.size() == 1) {
            return keys.get(0).getDirectionHints(player.getPosition());
        } else {
            Key nearestKey = keys.get(0);
            for (int i = 1; i < keys.size(); i++) {
                if (keys.get(i).distanceTo(player) <
nearestKey.distanceTo(player))
                    nearestKey = keys.get(i);
            }
            return nearestKey.getDirectionHints(player.getPosition());
        }
    }
```

3:20PM I will then create a createCoveredMap() method which returns a 2D array of characters representing the partially covered game map. The method will first create a fully-covered game map, then it uncovers the player and the rest game map based on the player's visibility.

3:25PM I realised I need to do some sort of validation to the walls since players cannot walk through walls.

```
private char[][] createCoveredMap() {

        char[][] coveredMap = new char[height][width];

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                coveredMap[i][j] = FOG;
            }
        }
```

This code covers the whole map with a shadow initially.

```
        coveredMap[player.getY()][player.getX()] = player.getIcon();
```

This code presents the player

```
        for (int j = player.getX() - 2; j <= player.getX() + 2; j++) {
```

```
            for (int i = player.getY() - 1; i >= player.getY()
            player.getVisibility() * 2 - 1 && i >= 0; i--) {
            coveredMap[i][j] = map[i][j];
            if (map[i][player.getX()] == HORIZONTAL_WALL) break;
           }
```

This code deals with the map being presented to the player from the player position upwards

```
for (int i = player.getY() + 1; i <= player.getY() + player.getVisibility() * 2 +
1 && i < height; i++) {
                coveredMap[i][j] = map[i][j];
                if (map[i][player.getX()] == HORIZONTAL_WALL) break;
            }
        }
```

This code deals with the map being presented to the player from the player position downwards

```
for (int i = player.getY() - 1; i <= player.getY() + 1; i++) {
            for (int j = player.getX() - 1; j >= player.getX() -
player.getVisibility() * 4 - 2 && j >= 0; j--) {
                coveredMap[i][j] = map[i][j];
                if (map[player.getY()][j] == VERTICAL_WALL) break;
            }
```

This code uncovers the map from the player position to the left onwards

```
for (int j = player.getX() + 1; j <= player.getX() + player.getVisibility() * 4 +
2 && j < width; j++) {
                coveredMap[i][j] = map[i][j];
                if (map[player.getY()][j] == VERTICAL_WALL) break;
            }
        }
```

This code uncovers the map from the player position to the right onwards

Stopped @ 3:40PM

=== September 28th, 2020 – started @ 1:00 PM

Today I will be creating a method which prints the game introduction text which is invoked before level 1 is started. Estimated: 20 minutes

I will then print the string with a delay to give a more dramatic effect and allow users to read the script slowly.

1:20PM I will use a try … do… while… loop to make the program wait for the user to press enter.

```
try
            Scanner s = new Scanner(System.in)) {
                String input;
                do {
                    input = s.nextLine();
                } while (!input.equals(""));
            }
```

1:40 I will now create a method that deals with printing a victory message on the console which will be invoked when the player has completed both levels successfully.

```
void winMessage() {
        int mid_H = height / 2;
        int mid_W = width / 2;

        clear(mid_H, mid_W);

        map[mid_H][mid_W - 3] = 'V';
        map[mid_H][mid_W - 2] = 'I';
        map[mid_H][mid_W - 1] = 'C';
        map[mid_H][mid_W] = 'T';
        map[mid_H][mid_W + 1] = 'O';
        map[mid_H][mid_W + 2] = 'R';
        map[mid_H][mid_W + 3] = 'Y';
        map[mid_H][mid_W + 4] = '!';

        Game.isVisibilityMode = false;
        update();
    }
```

This will display a “victory” pattern on the map. I will then create a method that

- deals with losing (printing game over)
- a method that deals with indicating next level which will be invoked when a player has finished level 1
- a method that indicates an invalid movement when the players movement is not valid e.g trying to walk past the maze wall.

Stopped @ 2:00PM

=== September 29th, 2020 – started @ 3:00 PM

Today I will focus on making the game more dramatic. I will create a method that prints a specified string message character by character followed by a delay for the specified number of milliseconds after each character is printed. Estimated: 30 minutes

This will give the user a dramatic “typing effect”

```
private void printWithDelay(String message, int millis) {
        if (message.length() > 0) {
            for (int i = 0; i < message.length(); i++) {
                System.out.print(message.charAt(i));
                delay(millis);
            }
```

Thread.sleep() to delay for a specific number of milliseconds

```
private void delay(int millis) {
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
        }
    }
```

3:30PM I will now work on my Game class; this class focuses on encapsulating the logic for all the game states. I will create 7 classes.

`static boolean isVisibilityMode = true;` which will determine whether a partially covered game map will be generated and displayed, else a fully visible game map will be displayed.

`private JFrame jFrame;` which constructs a new frame that is initially invisible.

`private GameMap gameMap; which is` a class encapsulating the maze with game objects and the methods to add and remove game objects from the maze.

`private Player player;` which is a class representing the 'player' of this game.

`private Display display;` which is a class encapsulating the methods required for printing stuff to the console.

`private LevelOne levelOne;` which is a class encapsulating the game logic of level one.

Level details:

•Mission: collect all keys to reach level two

•Bonus: collect torch to increase visibility

•Each valid movement costs one drop of blood.Game over when the health level of the player is equals to zero.

`private LevelTwo levelTwo;` which is a class encapsulating the game logic of level two.

Level details:

Mission: find the exit to win the game

Health level of the player will decrease by 5 every second. Game Over when the health level of the player is less than or equals to zero.

I will then create a init() method that Prompts the player to enter the maze height and maze width and show the game intro. The difficulty of the game increases with the size of the maze.

Note: the maze height and width must be at least 5 because there will be 5 game objects in level one. Each game object will be scattered randomly in different rows and columns. No two game objects will appear in the same row nor the same column. Therefore, the maze must have a minimum size of 5x5.

```
private void init() {
        Scanner s = new Scanner(System.in);
        int mazeHeight, mazeWidth;
        do {
            System.out.print("Enter maze height (min 5): ");
            while (!s.hasNextInt()) {
                System.out.print("That's not a number! Enter again: ");
                s.next();
            }
            mazeHeight = s.nextInt();
```

```
        } while (mazeHeight < 5);
        do {
            System.out.print("Enter maze width (min 5): ");
            while (!s.hasNextInt()) {
                System.out.print("That's not a number! Enter again: ");
                s.next();
            }
            mazeWidth = s.nextInt();
        } while (mazeWidth < 5);

        gameMap = new GameMap(mazeHeight, mazeWidth);
        player = new Player(gameMap.getRandomPosition());
        display = new Display(gameMap.getMap(), player);
        display.gameIntroMessage();
    }
```

Stopped @ 4:30PM

=== September 30th, 2020 – started @ 3:00 PM

Today I will continue working on my Game class. I will make a method in charge of creating a small window on the top left corner of the screen using jFrame.

KeyEvents cannot be detected in the console without JNI or GUI. By setting up a small window and hook the KeyListener to the window, player can control the Player's movement by using the arrow keys.

Note: The player must click on the window first before playing so that the KeyEvents can be captured by the window.

```
private void setupWindowForKeyListener() {
        jFrame = new JFrame();
        jFrame.setVisible(true);
        jFrame.setSize(100, 100);
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

I will then create a startLevelOne method that Initializes LevelOne, setup the ProgressListener and KeyListener before starting level one.

```
private void startLevelOne() {
        levelOne = new LevelOne(gameMap, player, display);
        levelOne.addProgressListener(this);
        jFrame.addKeyListener(levelOne);
        levelOne.start();
    }
```

I will do the same thing for level 2. In my game class I will call all 3 methods:

```
init();
setupWindowForKeyListener();
startLevelOne();
```

I will now work on my GameMap class, which focuses on encapsulating the maze with game objects and the methods to add and remove game objects from the maze.

I will create a GameMap method with mazeHeight and mazeWidth as the parameters. mazeHeight is the number of rows of the maze defined by the player and mazeWidth is the number of columns of the maze defined by the player.

```
GameMap(int mazeHeight, int mazeWidth) {

        this.map = new MazeGenerator(mazeHeight, mazeWidth).generate();
```

this will return a 2D array of characters representing the maze generated by the algorithm

```
        this.height = map.length;
        this.width = map[0].length;
        randomize_X();
        randomize_Y();

    }
```

Randomize_X generates a list of random x-coordinates that are valid in the map. This method is used to ensure that there are no duplicate coordinates being generated.

The sequence of valid x-coordinates is an arithmetic progression such that:

- the first term is 2
- the difference between the consecutive terms is 4
- the largest term is less than the width of the game map

Example: 2, 6, 10, 14, 18, ...

Randomize_Y generates a list of random y-coordinates that are valid in the map. This method is used to ensure that there are no duplicate coordinates being generated.

The sequence of valid y-coordinates is an arithmetic progression such that:

- the first term is 1
- the difference between the consecutive terms is 2
- the largest term is less than the height of the game map

Example: 1, 3, 5, 7, 9, ...

I will then Return a 2D array of characters representing the game map.

```
char[][] getMap() {
        return map;
    }
```

As well as the height of the map

```
public int getHeight() {
        return height;
    }
```

And lastly the width of the map

```
public int getWidth() {
        return width;
    }
```

4:30PM I will then add an icon character of the specified GameObject to the game map at its position.

Note: GameObject is the abstract base class for all game objects. This class encapsulates the icon in character and the Position of a game object.

```
void addToMap(GameObject object) {
        map[object.getY()][object.getX()] = object.getIcon();
    }
```

I will then remove an icon character of the specified GameObject to the game map at its position.

```
void removeFromMap(GameObject object) {
        map[object.getY()][object.getX()] = EMPTY_SPACE;
    }
```

I will then make a randomize_X method that generates a list of random x-coordinates that are valid in the map. This method is used to ensure that there's no duplicate coordinates being generated.

The sequence of valid x-coordinates is an arithmetic progression such that:

- the first term is 2

- the difference between the consecutive terms is 4

- the largest term is less than the {@link #width} of the game map

For Example: 2, 6, 10, 14, 18,

```
private void randomize_X() {
        this.random_X = new ArrayList<>();
        for (int i = 2; i < this.width; i += 4) this.random_X.add(i);
        Collections.shuffle(this.random_X);
    }
```

I will then generate a list of random y-coordinates using the same method as above.

5:00PM I will then Returns a Position object generated by using the random x and y coordinates. The result is then passed to the constructor while creating GameObject so that we can have our game objects scattered randomly on the game map for each new game.

```
Position getRandomPosition() {
        return new Position(random_X.remove(0), random_Y.remove(0));
    }
```

I will then create a method that returns true if the specified direction the specified Player is trying to move is valid. A valid movement is not blocked by any walls. The validity is determined by checking if the character in the movement's direction is a 'space' that represents 'no wall'.

```
boolean validateMovement(Player player, int direction) {
        switch (direction) {
            case Movable.DIRECTION_UP:
                return (player.getY() > 1 && this.map[player.getY() -
1][player.getX()] == EMPTY_SPACE);
            case Movable.DIRECTION_DOWN:
                return (player.getY() < height - 2 && this.map[player.getY() +
1][player.getX()] == EMPTY_SPACE);
            case Movable.DIRECTION_LEFT:
```

```
                return (player.getX() > 2 && this.map[player.getY()][player.getX()
- 2] == EMPTY_SPACE);
            case Movable.DIRECTION_RIGHT:
                return (player.getX() < width - 3 &&
this.map[player.getY()][player.getX() + 2] == EMPTY_SPACE);
            default:
                return false;
        }
```

Stopped @ 5:20PM

=== October 1st, 2020 – started @ 12:00 PM

Today I will be working on my LevelOne class, the purpose of this class is encapsulate the game logic of level one.

Level details:

- Mission: collect all keys to reach level two

- Bonus: collect torch to increase visibility

- Each valid movement costs one drop of blood.

- Game over when the health level of the player is equals to zero.

I will develop a LevelOne class with multiple classes.

```
class LevelOne implements KeyListener {

    static final int NUMBER_OF_KEYS = 3;
    private final GameMap gameMap;
```

A class encapsulating the maze with game objects and the methods to add and remove game objects from the maze.

```
    private final Player player;
```

A class representing the 'player' of this game.

```
    private final Display display;
```

A class encapsulating the methods required for printing stuff to the console.

```
    private ProgressListener progressListener;
```

The listener interface for receiving level progress feedback. The class that is interested in processing level progress feedback has to implement this interface.

A progress feedback is generated when the player has either completed or failed a level.

```
    private Torch torch;
```

A class representing the 'torch' game object.

```
    private ArrayList<Key> keys;
```

1:30PM I will now work on creating a method that initializes level one by adding game objects to the game map.

```
private void init() {
        gameMap.addToMap(player);

        keys = new ArrayList<>();
```

```
        for (int i = 0; i < NUMBER_OF_KEYS; i++) {
            Key key = new Key(gameMap.getRandomPosition());
            keys.add(key);
            gameMap.addToMap(key);
        }
        torch = new Torch(gameMap.getRandomPosition());
        gameMap.addToMap(torch);

        display.update(keys);
```

Updates the game map and game statistics to be printed on the console. There are two possible types of game map that can be printed based on the game mode (Game.isVisibilityMode).

If Game.isVisibilityMode is true, a partially-covered game map generated by createCoveredMap() will be printed on the console, else a fully visible game map which is not covered will be printed instead.

Note: this method is meant to be invoked in LevelOne because keys will only appear in level one.

2:00PM I will now create a method that starts the game loop of level one. I will a if loop with a try catch condition. The logic is if the player health is less than or equal to 0, the player will be invoked to end the game. If the size of the keys equal 0, which means all keys are collected, then removes the icon character of the specified GameObject from the game map at its position and invoke the player that level one is completed.

Note: 0 means all keys are collected

```
void start() {
        while (true) {
            if (player.isDead()) {
                progressListener.levelFailed();
                break;
            }
            if (keys.size() == 0) {
                gameMap.removeFromMap(torch);
                progressListener.levelOneCompleted();
                break;
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

2:30PM I will now create a method that updates the game statistics if the player has reached specified items in the game.

```
private void checkIfPlayerHasReachedItem() {
        for (Key key : keys) {
            if (player.getPosition().equals(key.getPosition())) {
                keys.remove(key);
                keys.trimToSize();
```

```
                System.out.println(keys.size());
                return;
            }
        }
        if (!player.hasTorch() &&
player.getPosition().equals(torch.getPosition()))
            player.foundTorch();
    }
```

Lastly, I will create a method that invokes when a key has been pressed.

```
public void keyPressed(KeyEvent e) {
        switch (e.getKeyCode()) {
            case KeyEvent.VK_UP:
            case KeyEvent.VK_DOWN:
            case KeyEvent.VK_LEFT:
            case KeyEvent.VK_RIGHT:
                final int DIRECTION = e.getKeyCode();
                if (gameMap.validateMovement(player, DIRECTION)) {
                    gameMap.removeFromMap(player);
                    player.move(DIRECTION);
                    gameMap.addToMap(player);
                    checkIfPlayerHasReachedItem();
                    player.reduceHealthLevelBy(1);
                    display.update(keys);
                } else {
                    display.invalidMovementMessage();
                }
                break;
            case KeyEvent.VK_F6:
                Game.isVisibilityMode = !Game.isVisibilityMode;
                display.update(keys);
                break;
            default:
                break;
        }
    }
```

If the arrow keys are pressed, the movement is validated first. If the direction of movement is valid (not blocked by any walls), the player icon at the position before moving is first removed from the game map. The player icon is added back to the game map at the new position after moving. The health level of the player is reduced by one after each valid movement.

The F6 key is used for debugging purposes to reveal the map even the map is previously covered. This is the hidden easter egg.

Stopped @ 3:00PM

=== October 2nd, 2020 – started @ 2:30 PM

Today I will be working on my LevelTwo class which encapsulate the game logic of level two. I will only be stating what's different in this class as LevelOne and LevelTwo class share similar attributes.

Level details:

Mission: find the exit to win the game

- Health level of the player will decrease by 5 every second.

- Game Over when the health level of the player is less than or equals to zero.

In the LevelTwo class it will consist of the same beginning classes that are in LevelOne class except for two new additions.

```
private Exit exit; A class representing the 'exit' of the maze.
private boolean hasFoundExit = false;
```

This class will also have a progress listener.

I will then create a method that initializes level two by adding game objects to the game map.

```
private void init() {
        gameMap.addToMap(player);
        exit = new Exit(Exit.generateExitPosition(gameMap, player));
```

Returns a Position object which is in a quadrant that is opposite to thespecified Player's quadrant in the maze.

```
        gameMap.addToMap(exit);
```

Adds the icon character of the specified GameObject to the game map at its position.

```
        display.update();
```

Updates the game map and game statistics to be printed on the console.

I will then create a method that updates the game stats if player has reached the exit.

```
private void checkIfPlayerHasFoundExit() {
        if (player.getPosition().equals(exit.getPosition()))
            hasFoundExit = true;
    }
```

Except that, other elements of this class is the same as LevelOne class.

Lastly, I will now work on my ProgressListener class, this serves as the listener interface for receiving level progress feedback. The class that is interested in processing level progress feedback has to implements this interface.

A progress feedback is generated when the player has either completed or failed a level.

```
public interface ProgressListener {
```

invoked when level 1 is completed

```
    void levelOneCompleted();
```

invoked when level 2 is completed

```
    void levelTwoCompleted();
```

Invoked when the Player is dead to end the game.

```
    void levelFailed();
}
```

Stopped @ 4:00PM

=== October 3rd, 2020 – started @ 10:30 AM

Today I will be working on my main.game_objects class, which is a class that represents the 'exit' of the maze. I will create a method that returns a position object which is in a quadrant that is opposite to the specified players quadrant in the maze. I will use an if … else… statement to achieve this.

```
public static Position generateExitPosition(GameMap gameMap, Player player) {
        int x, y;
        if (player.getX() > gameMap.getWidth() / 2) {
            if (player.getY() > gameMap.getHeight() / 2) {
                for example, if player is in quadrant 4, set exit to quadrant 2
                x = 2;
                y = 1;
            } else {
                For example, if player is in quadrant 1, set exit to quadrant 3
                x = 2;
                y = gameMap.getHeight() - 2;
            }
        } else {
            if (player.getY() > gameMap.getHeight() / 2) {
                For example, if player is in quadrant 3, set exit to quadrant 1
                x = gameMap.getWidth() - 3;
                y = 1;
            } else {
                For example, if player is in quadrant 2, set exit to quadrant 4
                x = gameMap.getWidth() - 3;
                y = gameMap.getHeight() - 2;
            }
        }
        return new Position(x, y);
    }
}
```

I will then work on my GameObject class, this class is an abstract class for all the game objects. This class encapsulates the icon in character and the position of the game object. I will be creating multiple methods.

```
public abstract class GameObject {

    private final char icon;
    private Position position;
```

Position is an immutable class encapsulating the x-coordinate and y-coordinate of a GameObject in the game map.

```
GameObject(char icon, Position position) {
        this.icon = icon;
        this.position = position;
```

- This method passes the icon to represent this game object, and the Position of this game object in the map.

```
public char getIcon() {
```

```
        return icon;
    }
```

- This method returns the icon of this game object.

```
public Position getPosition() {
            return position;
        }
```

- This is an immutable class encapsulating the x-coordinate and y-coordinate of a GameObject in the game map.

```
public int getX() {
            return position.getX();
        }
```

- This method Returns the x-coordinate of this game object's Position

```
void setX(int x) {
        position = new Position(x, position.getY());
    }
```

- This method sets the x-coordinate of this game object to the specified x-coordinate.

```
public int getY() {
        return position.getY();
    }
```

- This method Returns the y-coordinate of this game object's Position

```
void setY(int y) {
       position = new Position(position.getX(), y);
   }
```

- This method sets the y-coordinate of this game object to the specified y-coordinate.

```
public double distanceTo(GameObject obj) {
            return Math.sqrt(Math.pow(((this.getX() - obj.getX()) / 4.0), 2)
                        + Math.pow(((this.getY() - obj.getY()) / 2.0), 2));
            }
```

- Lastly, this method Returns the distance between the this game object and the specified game object.

Stopped @ 12:30PM

=== October 4th, 2020 – started @ 3:00 PM

Today I will be working on my key class. This class will represent a 'key' game object. I will create a method that returns a string object representing the direction hints to this Key object from the specified player's position.

I will use an if … else … statement with the condition that if the x-coordinate of this game object's Position is greater than the position of the player, print an arrow based on that.

The same goes for the y coordinate.

```
public String getDirectionHints(Position playerPosition) {
        StringBuilder sb = new StringBuilder();

        if (this.getX() > playerPosition.getX())
            sb.append("-> ");
        else if (this.getX() < playerPosition.getX())
            sb.append("<- ");

        if (this.getY() > playerPosition.getY())
            sb.append("v ");
        else if (this.getY() < playerPosition.getY())
            sb.append("^ ");

        return sb.toString();
    }
```

4:30PM I will then work on my Movable class.

This interface is implemented by a GameObject such as Player. A Movable is a GameObject that can change its Position in the maze. The move method is invoked to change the Position of the Movable.

```
public interface Movable {
    int DIRECTION_UP = KeyEvent.VK_UP;
    int DIRECTION_DOWN = KeyEvent.VK_DOWN;
    int DIRECTION_LEFT = KeyEvent.VK_LEFT;
    int DIRECTION_RIGHT = KeyEvent.VK_RIGHT;
```

Invoked when the Position of the GameObject has to be changed

```
    void move(int direction);
}
```

5:00PM I will now work on my Player class. This is a class representing the 'player' of this game. I will create multiple variables. A health level variable that stores the health level, a default icon variable that stores the character icon, in this case it will be "J" and initial visibility.

```
public class Player extends GameObject implements Movable {

    public static final int INITIAL_HEALTH_LEVEL = 250;
    private static final char DEFAULT_ICON = 'J';
    private static final int INITIAL_VISIBILITY = 2;
    private int visibility;
```

```
    private int healthLevel;
    private boolean hasTorch;
```

I will then position the Position of this game object

```
public Player(Position position) {
        super(DEFAULT_ICON, position);
        this.visibility = INITIAL_VISIBILITY;
        this.healthLevel = INITIAL_HEALTH_LEVEL;
        this.hasTorch = false;
    }
```

I will then create a method that Returns the visibility of this Player object. Note: Visibility is the number of blocks that is not covered up and is visible to the player.

```
public int getVisibility() {
        return visibility;
    }
```

5:30 PM I will then create a method that returns the healthLevel of this Player object.

```
public int getHealthLevel() {
        return healthLevel;
    }
```

I will then need to create a method that Reduces the healthLevel of this Player object.

```
public void reduceHealthLevelBy(int value) {
        healthLevel -= value;
    }
```

I will then need to create a method that returns true if this Player object's healthLevel is less than or equals to zero or false if otherwise.

```
public boolean isDead() {
        return healthLevel <= 0;
    }
```

7:00 PM I will now create a method that Returns true if the player has reached a torch before.

```
public boolean hasTorch() {
        return hasTorch;
    }
```

I will then create a method that Sets hasTorch to true and increase this Player object's visibility.

```
public void foundTorch() {
        hasTorch = true;
        visibility += Torch.VISIBILITY_BOOST;
    }
```

Lastly, I will create a method that changes the player object position based on the specified direction. This makes the game more realistic instead of having a fixed character orientation when navigating around the maze.

```
public void move(int direction) {
        switch (direction) {
            case DIRECTION_UP:
                this.setY(this.getY() - 2);
                break;
            case DIRECTION_DOWN:
                this.setY(this.getY() + 2);
                break;
            case DIRECTION_LEFT:
                this.setX(this.getX() - 4);
                break;
            case DIRECTION_RIGHT:
                this.setX(this.getX() + 4);
                break;
            default:
                break;
        }
    }
```

Stopped @ 6:00PM

=== October 5th, 2020 – started @ 5:00 PM

Today I will be working on my Position class. This is an immutable class encapsulating the x-coordinate and y-coordinate of a GameObject in the game map.

```
public final class Position {

    private final int x;
    private final int y;

    public Position(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

I will then return the x coordinate as well as y coordinate

```
int getX() {
        return x;
    }

int getY() {
        return y;
    }
```

I will then create a method that compares this Position to the specified object. The result is true if and only if the argument is not null and is a Position object that represents the same x and y coordinate as this object.

```
public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        } else if (!(obj instanceof Position)) {
            return false;
```

```
        } else {
            Position position = (Position) obj;
            return this.x == position.x && this.y == position.y;
        }
    }
```

The last method in this class will return a string object representing this Position's coordinates. The x and y coordinates are separated by a comma and returned as a string.

```
public String toString() {
        return x + ", " + y;
    }
```

I will then work on my Torch class which is the last class in my main.game_objects package. This class represents the 'torch' game object.

```
public class Torch extends GameObject {

    static final int VISIBILITY_BOOST = 2;
    private static final char DEFAULT_ICON = '%';

    public Torch(Position position) {
        super(DEFAULT_ICON, position);
    }

}
```

Stopped @ 6:00PM

=== October 6th, 2020 – started @ 1:00 PM

I will now work on my second to last package which is main.maze_generator. This class encapsulates the algorithm for maze generation. The algorithm used is Aldous-Broder algorithm, a passage-carving algorithm. It is one of the slowest algorithms but it is the simplest one to implement.

I will create a method with two parameters, rows which is the number of rows of the maze defined by the player and columns which is the number of columns of the maze defined by the player

```
public MazeGenerator(int rows, int columns) {
        this.rows = rows;
        this.columns = columns;
        maze = new char[rows * 2 + 1][columns * 4 + 1];
        initAllWalls();
    }
```

I will then create a method that returns a 2D array of characters representing the maze generated by the algorithm.

```
public char[][] generate() {

        boolean[][] isVisited = new boolean[rows][columns];
        int remainingBlocks = rows * columns;
```

```
        int row = 0, column = 0;
        isVisited[row][column] = true;
        remainingBlocks--;

        while (remainingBlocks > 0) {
            int direction = getDirection(row, column);
            int n_row = (direction == UP) ? row - 1 : (direction == DOWN) ? row + 1 : row;
            int n_column = (direction == LEFT) ? column - 1 : (direction == RIGHT) ? column + 1 : column;

            if (!isVisited[n_row][n_column]) {
                carveWall(row, column, direction);
                isVisited[n_row][n_column] = true;
                remainingBlocks--;
            }
            row = n_row;
            column = n_column;
        }

        return maze;
    }
```

carveWall carves a passage from the specified current block to a valid random adjacent block in the given direction. The passage is carved by replacing the 'walls' with 'spaces'. This has 3 Parameters:

- row the row of the current block
- column the column the current block
- direction the valid direction to an adjacent block

I will then create a method that Initialize the maze with all walls.

```
private void initAllWalls() {
        for (int row = 0; row < maze.length; row++) {
            if (row % 2 == 0) {
                for (int column = 0; column < maze[0].length; column++) {
                    if (column % 4 == 0)
                        maze[row][column] = '+';
                    else
                        maze[row][column] = HORIZONTAL_WALL;
                }
            } else {
                for (int column = 0; column < maze[0].length; column++) {
                    if (column % 4 == 0)
                        maze[row][column] = VERTICAL_WALL;
                    else
                        maze[row][column] = EMPTY_SPACE;
                }
            }
        }

    }
```

This method will carve a passage from the specified current block to a valid random adjacent block in the given direction. The passage is carved by replacing the 'walls' with 'spaces'.

```
private void carveWall(int row, int column, int direction) {
        switch (direction) {
            case UP:
                for (int i = column * 4 + 1; i <= column * 4 + 3; i++)
                    maze[row * 2][i] = EMPTY_SPACE;
                break;
            case DOWN:
                for (int i = column * 4 + 1; i <= column * 4 + 3; i++)
                    maze[row * 2 + 2][i] = EMPTY_SPACE;
                break;
            case LEFT:
                maze[row * 2 + 1][column * 4] = EMPTY_SPACE;
                break;
            case RIGHT:
                maze[row * 2 + 1][column * 4 + 4] = EMPTY_SPACE;
                break;
        }
    }
```

Lastly, this method returns an integer representing a valid random direction based on the specified current block position.

```
    private int getDirection(int row, int column) {
        Random r = new Random();
        int direction;

        if (column == 0) {
            if (row == 0) {
                // down or right
                direction = DOWN + r.nextInt(2);
            } else if (row == rows - 1) {
                // up or right
                direction = RIGHT * r.nextInt(2);
            } else {
                // up or down or right
                direction = r.nextInt(3);
                direction = (direction == LEFT) ? RIGHT : direction;
            }
        } else if (column == columns - 1) {
            if (row == 0) {
                // down or left
                direction = LEFT + r.nextInt(2);
            } else if (row == rows - 1) {
                // up or left
                direction = r.nextInt(LEFT + 1);
            } else {
                // up or down or left
                direction = r.nextInt(3);
            }
        } else if (row == 0) {
            // left or right or down
            direction = LEFT + r.nextInt(RIGHT);
        } else if (row == rows - 1) {
            // left or right or up
            direction = r.nextInt(3);
```

```
            direction = (direction == DOWN) ? RIGHT : direction;
        } else {
            // any directions
            direction = r.nextInt(4);
        }

        return direction;
    }
}
```

Stopped @ 7:00PM

=== October 7th, 2020 – started @ 2:30 PM

Today I will be developing my last package which will be my main class which will serve as the entry point to my application. I will use a try… catch… condition to clear the screen, and will catch common errors if they do ever occur

```
public class Launcher {
    public static void main(String[] args) {
        try {
            new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
        } catch (InterruptedException | IOException e) {
            e.printStackTrace();
        }
        System.out.println("-----------Welcome to the Maze Runner Game-----------
");
        new Game();
    }
}
```