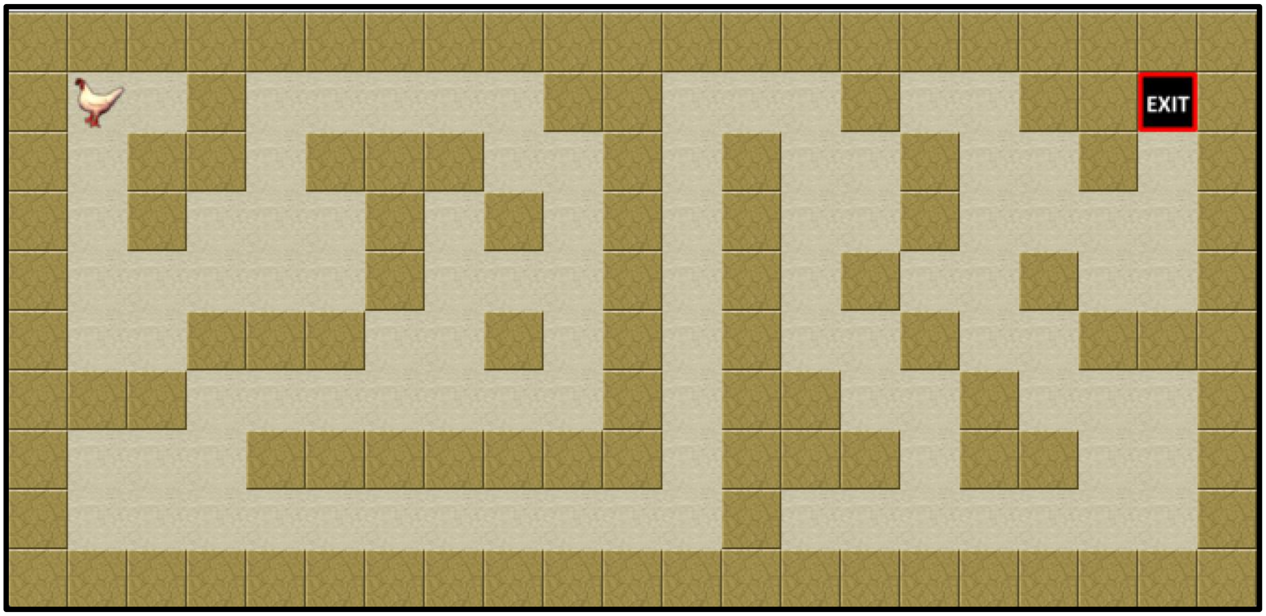
University of New Orleans - Department of Computer Science

Maze Game

An Exploration Adventure Game



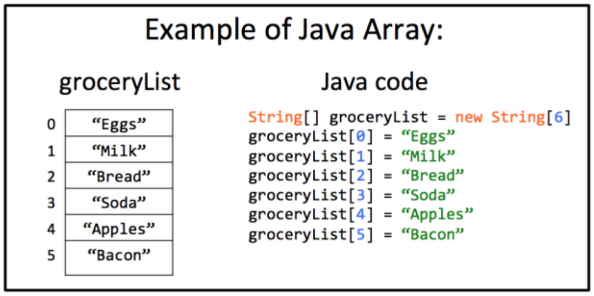
Homework Summary

Iterations	Overview	Expected Time	Points per Goal (Sum)
Goal 01	**Design the Game Class & Stub out methods**	10 minutes	+5 points (05)
Goal 02	**Read in the Config File & Display it to Console**	10 minutes	+5 points (10)
Goal 03	**Save Level data & Load a Scene**	10 minutes	+5 points (15)
Goal 04	**Identify Walls and Floors**	10 minutes	+5 points (20)
Goal 05	**Draw the Scene with graphics**	10 minutes	+5 points (25)
Goal 06	**Draw the player**	10 minutes	+5 points (30)
Goal 07	**Player Controls**	10 minutes	+5 points (35)
Goal 08	**Collision Detection for walls**	10 minutes	+5 points (40)
Goal 09	**Draw Exit**	10 minutes	+5 points (45)
Goal 10	**Game Over and Additional levels**	10 minutes	+5 points (50)
Final	**Homework**: *Add custom features*	∞ minutes	+50 points (100)

Introduction:

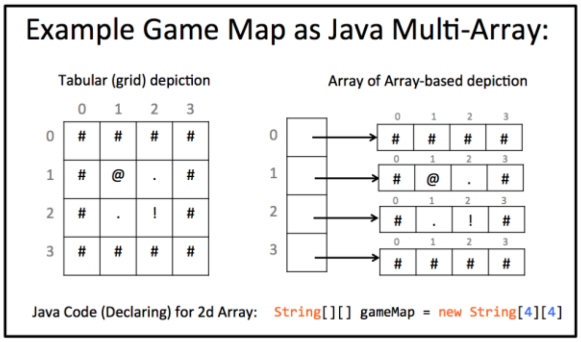
In this lab, you will build a multi-level maze game. The player must navigate the two dimensional maze to reach the exit. Exiting one maze may bring the player into another maze until the game is complete. To implement such a game, you must use 2d array to store the level data (i.e. the mazes). The mazes are provided into the game application from a configuration file.

Quick Explanation of Arrays:
Arrays are special type of storage operation where we can store more than one value in it. Arrays are very versatile as they can store any data type including other arrays of values.



	groceryList
0	"Eggs"
1	"Milk"
2	"Bread"
3	"Soda"
4	"Apples"
5	"Bacon"

```
String[] groceryList = new String[6]
groceryList[0] = "Eggs"
groceryList[1] = "Milk"
groceryList[2] = "Bread"
groceryList[3] = "Soda"
groceryList[4] = "Apples"
groceryList[5] = "Bacon"
```


Example Game Map as Java Multi-Array:
Tabular (grid) depiction
Array of Array-based depiction

	0	1	2	3
0	#	#	#	#
1	#	@	.	#
2	#	.	!	#
3	#	#	#	#

Java Code (Declaring) for 2d Array: `String[][] gameMap = new String[4][4]`

Data in Configuration file:
First line contains the number of levels in this world file
Second line contains two integers:

1. the number of rows in game map
2. the number of columns in game map

Remaining lines contain the symbols that represent the game tile for each cell in map

Symbols for Configuration File:

#	Wall
.	Floor
@	Player
!	Exit

maps.txt *(configuration file)*

```
1
10 10
# # # # # # # # # #
# @ . . . . . . . #
# . # # . # # # . #
# . # . . . # . . #
# . # . . . # . . #
# . . # # # . . # #
# # . . . . . . . #
# . . # # # # # # #
# . . . . . . . ! #
# # # # # # # # # #
```

Goal 0: Goal: Designing the Game classes.

`MazeGame`	Manages Game data and game methods	main, start, update, render
`Scene`	Manages Scene data and Scene methods	start, draw
`Player`	Manages Player data and Player methods	start, draw, update, getters
`Exit`	Manages Exit data and Exit methods	start, draw, getters
`WorldData`	Manage World Data and methods to access world data	start, getters
`StdDraw`	Class for drawing graphics	

Starting this Lab *(Gitlab)*	*See moodle.*

Goal 1: Make Game class responsible for managing the game rules.

Planning:

Architect the general workflow of the application. This game application architecture will be similar to the previous project.

Implementation:

MazeGame.java → class name & class data

```
public class MazeGame {
    private static boolean gameOver;
}
```

MazeGame.main()

```
public static void main(String[] args) {
    start();
    while (gameOver == false) {
        update();
        render();
    }
}
```

MazeGame.start()

```
public static void start() {
    gameOver = false;
}
```

MazeGame.update()

```
public static void update() {

}
```

MazeGame.render()

```
public static void render() {

}
```

Testing:

Ensure that the code compiles, at this point there is no logic to test.

Goal 2: Read the configuration file and display it on the console.

Planning:

Create a World class responsible for holding all the level data in the game. The world data will get its info from the configuration file and then save it in the program. This class needs a start method.

Implementation:

World.java

```java
import java.util.Scanner;

public class World {

}
```

World.java

```java
public static void start() {
    /get all map data and save it for later
    Scanner input = new Scanner(System.in);
    int count = input.nextInt();
    for (int lvl=0; lvl<count; lvl++) {
        int rows = input.nextInt();
        int cols = input.nextInt();
        setLevel(rows, cols, input);
    }
}
```

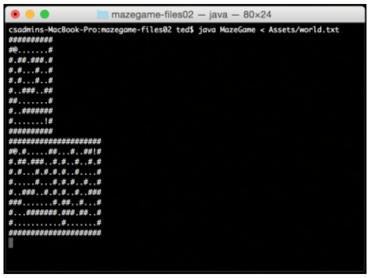
World.java → setLevel method

```java
public static void setLevel(int rows, int cols, Scanner input) {
    for (int y=0; y < rows; y++) {
        for (int x=0; x < cols; x++) {
            String tile = input.next();
            System.out.print(tile);
        }
        System.out.print("\n");
    }
}
```

MazeGame.java → start()

```java
//Start game algorithm:
public static void start() {
        gameOver = false;
        World.start();
}
```

Testing:



```
csadmins-MacBook-Pro:mazegame-files02 ted$ java MazeGame < Assets/world.txt
##########
#@.......#
#.##.###.#
#.#...#..#
#.#...#..#
#..###..##
##.......#
#..#######
#.......!#
##########
#####################
#@.#.....##...#..##!#
#.##.###..#.#..#..#.#
#.#...#.#.#.#..#....#
#.....#...#.#.#..#..#
#..###..#.#.#..#..###
###.......#.##..#...#
#...#######.###.##..#
#...........#.......#
#####################
```

To execute your program:

```
java MazeGame < Assets/world.txt
```

The console should print both level maps to the screen

Goal 3: Save World data into a Multidimensional Array & Setup a Scene

Planning:

Scene —> row, col data, start method that prints the scene

World —> save data into 3 dim array
MazeGame —> add level variable, start Scene with level number

Implementation:

World.java → new class data

```
private static String[][][] levels;
```

World.java → start() method

```
public static void start() {
    //get all map data and save it for later
    Scanner input = new Scanner(System.in);
    int count = input.nextInt();

    levels = new String[count][][];

    for (int lvl=0; lvl<count; lvl++) {
        int rows = input.nextInt();
        int cols = input.nextInt();
        setLevel(lvl, rows, cols, input);
    }
}
```

World.java → setLevel method

```
public static void setLevel(int lvl, int rows, int cols, Scanner input) {
    levels[lvl] = new String[rows][cols];
    for (int y=0; y < rows; y++) {
        for (int x=0; x < cols; x++) {
            String tile = input.next();
            levels[lvl][y][x] = tile;
            ~~System.out.print(tile);~~
        }
        ~~System.out.print("\n");~~
    }
}
```

World.java → new method

```
public static String[][] getLevel(int level) {
    return levels[level];
}
```

Scene.java

```
public class Scene {
    private static int rows;
    private static int cols;
}
```

Scene.java

```
public static void start(int level) {
    String[][] map = World.getLevel(level);
    rows = map.length;
    cols = map[0].length;

    for (int y=0; y < rows; y++) {
        for (int x=0; x<cols; x++) {
            String tile = map[y][x];
            System.out.print(tile);
        }
    System.out.print("\n");
   }
}
```

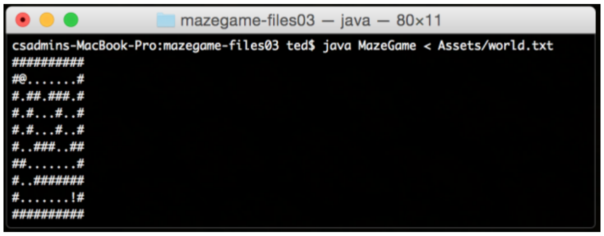
MazeGame.java → new class data

```
private static int level;
```

MazeGame.java → start() method

```
public static void start() {
    gameOver = false;
    level = 0;
    World.start();
    Scene.start(level);
}
```

Testing:



mazegame-files03 — java — 80×11

```
csadmins-MacBook-Pro:mazegame-files03 ted$ java MazeGame < Assets/world.txt
##########
#@.......#
#.##.###.#
#.#...#..#
#.#...#..#
#..###..##
##.......#
#..#######
#.......!#
##########
```

To execute your program:

```
java MazeGame < Assets/world.txt
```

The console should print the first level map to the screen

Goal 4: Determine wall tiles vs. floor tiles

Planning:

[image showing the mapping to a boolean 2d array for mapping where walls are]

Scene —> walls array

start —> set booleans in walls array

draw —> iterate over wall array & display true/false

Game —> call draw method

Implementation:

Scene.java → class data

```
private static int rows;
private static int cols;
private static boolean[][] walls;
```

Scene.java → setTile() method

```
public static void setTile(int x, int y, String tile) {
    if ( tile.equals("#") ) {
        walls[y][x] = true;
    }
}
```

Scene.java → start() method

```
public static void start(int level) {
    String[][] map = World.getLevel(level);
    rows = map.length;
    cols = map[0].length;

    walls = new boolean[rows][cols];
    for (int y=0; y < rows; y++) {
        for (int x=0; x < cols; x++) {
            String tile = map[y][x];
            setTile(x,y,tile);
            ~~System.out.print(tile);~~
        }
        ~~System.out.print("\n");~~
    }
}
```

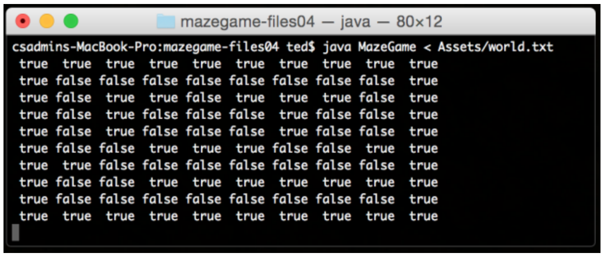
Scene.java → draw() method

```
public static void draw() {
    for(int y=0; y < rows; y++) {
        for (int x=0; x < cols; x++) {
            System.out.printf("%5s ", walls[y][x] );
        }
        System.out.print("\n");
    }
}
```

MazeGame.java → render() method

```
public static void render() {
    Scene.draw();
    StdDraw.show(100);
}
```

Testing:



```
csadmins-MacBook-Pro:mazegame-files04 ted$ java MazeGame < Assets/world.txt
 true  true  true  true  true  true  true  true  true  true
 true false false false false false false false false  true
 true false  true  true false  true  true  true false  true
 true false  true false false false  true false false  true
 true false  true false false false  true false false  true
 true false false  true  true  true false false  true  true
 true  true false false false false false false false  true
 true false false  true  true  true  true  true  true  true
 true false false false false false false false false  true
 true  true  true  true  true  true  true  true  true  true
```

To execute your program:

```
java MazeGame < Assets/world.txt
```

*The console should print **true** for wall tiles & **false** for floor tiles*

Goal 5: Draw the Scene with StdDraw

Planning:

Scene —> add floor Image, wall Image, TILE_SIZE, width, height
start —> set values to new variables
draw —> iterate over array and draw each picture

Implementation:

Scene.java → class data

```java
private static final int TILE_SIZE = 32;

private static int rows;
private static int cols;
private static boolean[][] walls;
private static int width;
private static int height;
private static String floorImage;
private static String wallImage;
```

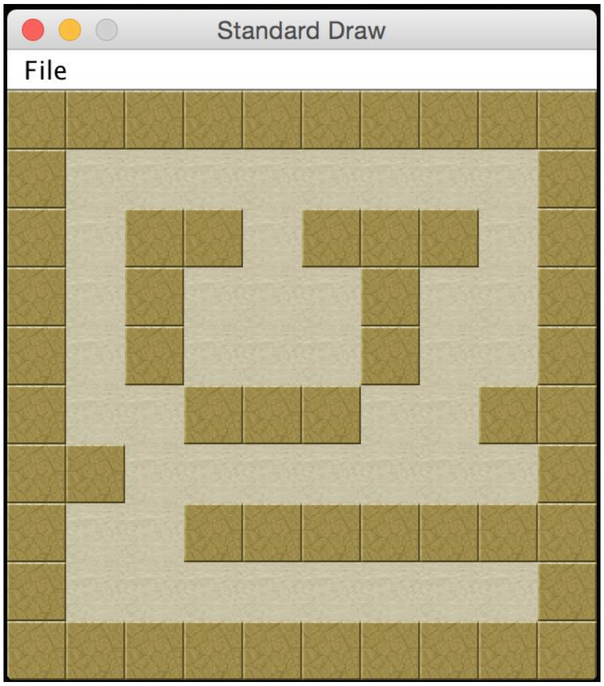
Scene.java → start() method

```java
public static void start(int level) {
    floorImage = "Assets/tile-passage.png";
    wallImage = "Assets/tile-brickwall.png";

    String[][] map = World.getLevel(level);
    rows = map.length;
    cols = map[0].length;

    width = cols * TILE_SIZE;
    height = rows * TILE_SIZE;

    walls = new boolean[rows][cols];
    for (int y=0; y<rows; y++) {
        for (int x=0; x<cols; x++) {
            String tile = map[y][x];
            setTile(x,y,tile);
        }
    }

    //setup canvas data (size & scale)
    StdDraw.setCanvasSize(width, height);
    StdDraw.setXscale(0.0, width);
    StdDraw.setYscale(height, 0.0);
}
```

Scene.java → draw() method

```java
public static void draw() {
    for(int y=0; y < rows; y++) {
        for (int x=0; x < cols; x++) {
            int tileX = x * TILE_SIZE + TILE_SIZE/2;
            int tileY = y * TILE_SIZE + TILE_SIZE/2;

            if( walls[y][x] == true) {
                StdDraw.picture(tileX, tileY, wallImage );
            }
            else {
                StdDraw.picture(tileX, tileY, floorImage );
            }
            }
        }
    }
```

Testing:

	**To execute your program:** `java MazeGame < Assets/world.txt` *Graphical window should display level 0*

Goal 6: Draw the Player

Planning:

Player —> create player class
Scene —> start the player
Game —> player.draw()

Implementation:

Player.java

```
public class Player {
    public static final int TILE_SIZE = 32;
    private static int x;
    private static int y;
    private static String image;
}
```

Player.java

```
//Start player data
public static void start(int x, int y) {
    Player.x = x;
    Player.y = y;
    image = "Assets/character-chicken.png";
}
```

Player.java

```
public static void draw() {
    int tileX = x * TILE_SIZE + TILE_SIZE/2;
    int tileY = y * TILE_SIZE + TILE_SIZE/2;
    StdDraw.picture(tileX, tileY, image);
}
```

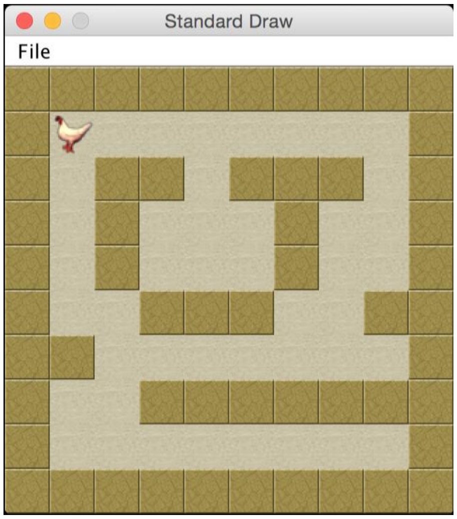
Scene.java → setTile() method

```
public static void setTile(int x, int y, String tile) {
    if ( tile.equals("#") ) {
        walls[y][x] = true;
    }
    else if ( tile.equals("@") ) {
        Player.start(x,y);
    }
}
```

MazeGame.java → render() method

```
public static void render() {
    Scene.draw();
    Player.draw();
    StdDraw.show(100);
}
```

Testing:

<table>
<tr>
<td> </td>
<td><b>To execute your program:</b>
<code>java MazeGame < Assets/world.txt</code>
<i>Graphical window should display level 0 with Player</i></td>
</tr>
</table>

Goal 7: Move the Player

Planning:

Player —> add an update method

Game —> invoke player's update method

Implementation:

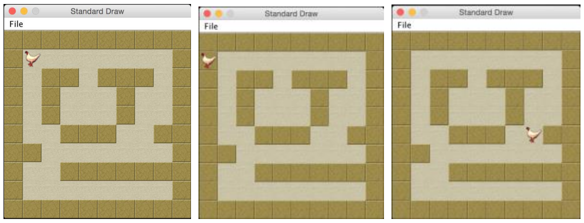
Player.java → update() method

```
public static void update() {
    if ( StdDraw.hasNextKeyTyped() == true ) {
        char key = StdDraw.nextKeyTyped();
        if ( key == 'w') {
            y--;
        }
        else if ( key == 's') {
            y++;
        }
        else if (key == 'a') {
            x--;
        }
        else if (key == 'd') {
            x++;
        }
    }
}
```

MazeGame.java → update() method

```
public static void update() {
    Player.update();
}
```

Testing:



To execute your program:

```
java MazeGame < Assets/world.txt
```

Move the chicken character around with WASD controls
NOTE: Walls are not yet implemented

Goal 8: Check for obstacles

Planning:

Player —> refactor move/update method
Scene —> canMove method

Implementation:

Scene.java → canMove() method

```java
public static boolean canMove(int x, int y) {
    return !walls[y][x];
}
```

Player.java → update() method

```java
public static void update() {
    if ( StdDraw.hasNextKeyTyped() == true ) {
        char key = StdDraw.nextKeyTyped();
        if ( key == 'w' && Scene.canMove(x,y-1) ) {
            y--;
        }
        else if ( key == 's' && Scene.canMove(x,y+1) ) {
            y++;
        }
        else if (key == 'a' && Scene.canMove(x-1,y) ) {
            x--;
        }
        else if (key == 'd' && Scene.canMove(x+1,y) ) {
            x++;
        }
    }
}
```

Testing:

To execute your program:

```
java MazeGame < Assets/world.txt
```

Move the chicken character around with WASD controls. Walls block movement.

Goal 9: Draw Exit

Planning:

Exit —> create Exit class
Scene —> start Exit
Game —> draw Exit

Implementation:

Exit.java

```java
public class Exit {
    public static final int TILE_SIZE = 32;
    private static int x;
    private static int y;
    private static String image;
}
```

Exit.java

```java
public static void start(int x, int y) {
    Exit.x = x;
    Exit.y = y;
    image = "Assets/tile-exit.png";
}
```

Exit.java

```java
public static void draw() {
    int tileX = x * TILE_SIZE + TILE_SIZE/2;
    int tileY = y * TILE_SIZE + TILE_SIZE/2;
    StdDraw.picture(tileX, tileY, image);
}
```

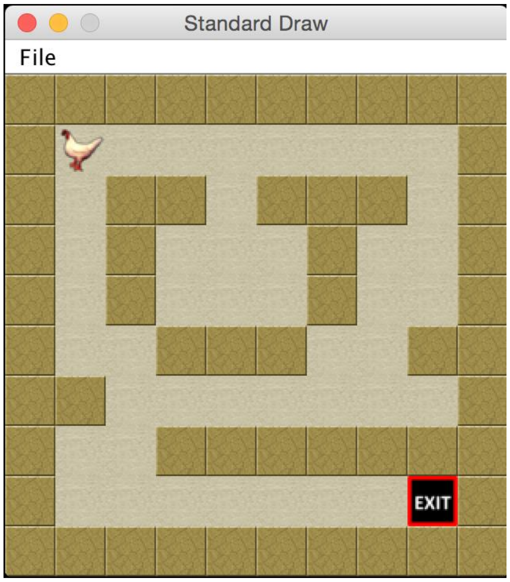
Scene.java → setTile() method

```java
public static void setTile(int x, int y, String tile) {
    if ( tile.equals("#") ) {
        walls[y][x] = true;
    }
    else if ( tile.equals("@") ) {
        Player.start(x,y);
    }
    else if ( tile.equals("!") ) {
        Exit.start(x,y);
    }
}
```

MazeGame.java → render() method

```java
public static void render() {
    Scene.draw();
    Exit.draw();
    Player.draw();
    StdDraw.show(100);
}
```

Testing:

	**To execute your program:** `java MazeGame < Assets/world.txt` *Graphical window should display level 0 with Player & Exit*

Goal 10: Add win condition and move between levels

Planning:

Game —> refactor update method

Implementation:

MazeGame.java → update() method

```java
public static void update() {
    Player.update();
    if (Player.getX() == Exit.getX() && Player.getY() == Exit.getY() ) {
        level++;
        if (level == World.getLength() ) {
            gameOver = true;
        }
        else {
            Scene.start(level);
        }
    }
}
```

Player.java → getX() and getY() methods

```java
public static int getX() {
    return x;
}

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

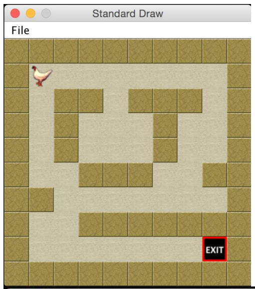
Exit.java → getX() and getY() methods

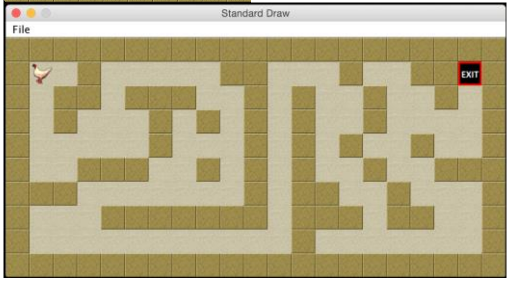
```java
public static int getX() {
    return x;
}

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

World.java → getLength() methods

```java
public static int getLength() {
    return levels.length;
}
```

Testing:





To execute your program:

```
java MazeGame < Assets/world.txt
```

Graphical window should display level 0 with Player & Exit.
When player touches exit, it should load level 1.
When player touches exit again, the game should be over.

Homework 5: Maze Game Homework

Part 1 (45 points):
Add three features to the game, any additional features count towards bonus points. At least one feature must include the use of an Array or an ArrayList.

1. Collect gems in a certain amount of time

2. Collect food to increase food counter, each move decrements the food counter
   (+1 more if you have different floor tiles that cost more food to traverse)

3. Add multiple enemies into the level that hurt the player
   (+1 more if the enemies move)
   (+1 more if player can attack the enemies)

4. Add a key item to level that player must collect to unlock the exit door first

5. Add more levels to the game

6. Any other features of your own design that make use of an Array or an ArrayList.

Part 2 (5 Points):
Write a readme.txt file detailing the custom features you added into your version of the game, explain how it works, and how to play.

Submitting:

Submit all homework files to both the class's gitlab repo and Moodle (zipped file):

Appendix A: Configuration file (*world.txt*)

world.txt

```
2
```

```
10 10
# # # # # # # # # #
# @ . . . . . . . #
# . # # . # # # . #
# . # . . . # . . #
# . # . . . # . . #
# . . # # # . . # #
# # . . . . . . . #
# . . # # # # # # #
# . . . . . . . ! #
# # # # # # # # # #
10 21
# # # # # # # # # # # # # # # # # # # # #
# @ . # . . . . . # # . . . # . . # # ! #
# . # # . # # # . . # . # . . # . . # . #
# . # . . . # . # . # . # . . # . . . . #
# . . . . . # . . . # . # . # . . # . . #
# . . # # # . . # . # . # . . # . . # # #
# # # . . . . . . . # . # # . . # . . . #
# . . . # # # # # # # . # # # . # # . . #
# . . . . . . . . . . . # . . . . . . . #
# # # # # # # # # # # # # # # # # # # # #
```