**Tawseef Abdullah**

**UB: 22071169**

**Faculty of Engineering and Informatics**

**BEng Software Engineering**

Task 1 – Adding a player

The code creates a player object using the player class constructor and sets the parameter for the maximum energy the player can have and the XY coordinate in which the player will spawn. This enables the GUI to load the player model in the game.

Task 2 – Player movement

The code changes the position of the player corresponding to the input key from the user. Every time the player presses a key, the keyPressed() method in the InputHandler class calls the movePlayer() method and passes a direction parameter according to the input.

In the movePlayer() method, 2 integer variables playerX and playerY are declared that calls the getX() and getY() methods for the player object and stores the coordinate of the player. Every switch statement calls the setPosition() method from the Entity class and passes different parameters depending on the input from the user. Case ‘N’ and ‘S’ moves the player to the upper or lower tile by calling the setPosition() method and changing the Y coordinates of the player by 1. Case ‘E’ and ‘W’ moves the player to the right or left tile by calling the setPosition() method and changing the X coordinates of the player by 1.

Task 3 – Level generation

The generateLevel() method instantiates the 2D array called level and sets its array lengths to the level width and height of the game. Then a nested for loop is used to iterate over every array index of level. With each iteration, a new Tile object is created with the TileType EMPTY from the TileType enumeration and assigned to the index. The GameGUI class then renders the EMPTY tiles in the game.

Task 4 – Better player movement

An if statement is added to every switch statement in the movePlayer() method to check if the setPosition() method would not move the player outside the level. If it does, the setPosition() command will not run and the user input would essentially be ignored.

Task 5 – Better level generation

Inside the nested for loop in the generateLevel() method, an integer n is declared and assigned a random number between 0 and 99 using the random number generator rng. An if-else statement is created which creates a new Tile object with a specific tiletype and assigns it to the level index the loop is accessing. If n is between 0 and 79, the tile becomes EMPTY which means for every tile there is a 80% chance the tile will be EMPTY. If n is between 80 and 99, the tile becomes DIRT which means for every tile there is a 20% chance the tile will be DIRT. This makes sure the level tiles will have more diversified tiletypes.

Task 6 – Even better player movement

In the movePlayer() method, an if statement is added to each switch cases which checks if the tile that the player is about to move to is an EMPTY tile or not. If it is empty, the setPosition() method is called accordingly. If it is not empty, the player will be unable to move into the tile.

Task 7 – Mining

In the movePlayer() method, an else statement is added to the nested if statement. If the tile the player is about to is not an empty tile, the mine() method is called from the Tile class and is passed the miningStrength value for the parameter strength. This makes the player mine the tile until its durability is depleted. A Tile variable is declared at the beginning of each case to store the coordinate of the tile the player is about to move to or mine.

Task 8 – Adding moles

The addMoles() method instantiates the moles array with a fixed index length. A for loop is created to access every index in the moles array. For every index, 2 integer variables xPos and yPos are declared. A randomly generated number between 1 and 33 is assigned to xPos and a number between 1 and 16 is assigned to yPos. Then, for each moles index, a new Mole class object is created and passed a specific value for its maxfullness parameter and the xPos and yPos variables for the initial position of the moles. This makes sure the moles are randomly spawned in the level.

Task 9 – Mole movement

In the moveAllMoles() method, a for loop is created to iterate over each index in the moles array. For each index, an if statement is added to check if the index is null or not. If it is not null, the code calls the moveMole() method and passes the current index it is accessing as its parameter. This makes sure the moveMole() method is called for each non-null mole in the array.

The moveMole() method works with all the moles one at a time. In the method, 2 integer variables moleX and moleY are declared which stores the coordinates of the mole. An integer variable n is declared and is assigned a number between 0 and 3. A switch statement is created which calls the setPosition() method and is passed different parameters depending on the value of n. If n = 0, the mole moves up. If n = 1, the mole moves down. If n = 2, the mole moves right. If n = 3, the mole moves left. This makes sure the mole movement is random and unpredictable.

Task 10 – Improved level generation

The generateLevel() method is modified to spawn all types of tile from the TileType enumeration. The if-else statement inside the nested for loop is modified so that every tiletype has a small chance of spawning in the level. The chance of tile spawning depends on the value of n. Uranium is considered the rarest ore so it has a chance of only 5% to spawn while dirt is considered the most common so it has a chance of 20% to spawn.

Task 11 – Better mole movement

In the moveMole() method, 2 integer variables playerX and playerY is declared to store the current coordinates of the player. In the beginning of each case of the switch statement, an if statement is added that checks if the tile the mole is about to move to is outside the level and doesn’t have the player in it by checking the current position of the player. If both cases are true, the setPosition() method is executed.

Task 12 – Mining moles

The switch cases are modified so that before the mole moves to a tile, the code checks if the tile is empty. If it is empty, the mole will move to the tile. If it is not, the code calls the mine() method on the tile and passes the miningStrength value in the parameter. The mole will not be able to move to the tile until it is fully mined.

Task 13 – EXPLODING MOLES

In every case in the moveMole() method, before each time the mole mines a tile, the current durability of the tile is stored in an integer variable and after it mines, its fullness is increased by the durability the tile had. This is done by calling the changeFullness() method from the Mole class and passing the variable as its parameter.

In the explode() method, 2 integer variables moleX and moleY are declared to store the current coordinates of each moles. A Tile variable is declared for each tile surrounding a mole including the tile the mole is on. Then, the mine() method is called on each of these tile variables and passed 100 as the strength parameter to make sure the tiles are mined instantly. This somewhat simulates an explosion at the location of a mole whose fullness reached max fullness and destroys all tiles around it. The clearExplodedMoles() method is called in the doTurn() method and executed every turn. It uses a for loop to iterate over every mole in the array and uses an if statement to check if a mole’s fullness reached maxfullness and then changes the mole index to null. Another if statement is added at the beginning of the for loop to make sure the loop does not access a null index. In the moveAllMoles() method, an if statement has been added inside the for loop to prevent the loop from accessing a null index in the moles array. Everytime the moveMole() method is called, the code also checks if a mole’s fullness reached max fullness and calls the explode() method on the mole that reached max fullness.

Task 14 – Player energy

In the movePlayer() method, before the player can mine a tile, an if statement is used to check if the player’s current energy is equal to or greater than the tile’s current durability. Then, the player mines the tile and its energy is reduced by the durability of the tile by calling the changeEnergy() method. An if statement is added outside the switch statement that checks if a player is standing on a BASE tile and has its energy increased to max energy.

In the doTurn() method, the code has been modified so that the player can regain some energy every turn.

Task 15 – New levels

The allOreMined() method declares a boolean value oreMined and sets it to true by default. Then, it uses a nested for loop to iterate over every index in the level array and uses an if statement to check if there are any ores left in the level. If there are, it changes the value of oreMined to false. If there are no ores left, oreMined remains true. Overall, the method returns the boolean value of oreMined. In the doTurn() method, an if statement checks if the allOreMined() method returns a boolean value true and if the current position of the player is in a BASE tile and then calls the nextLevel() method if so. In the nextLevel() method, the integer class attribute levelNumber is incremented by one and the methods generateLevel(), addMoles(), placePlayer() are called. The placePlayer() method calls the setPosition() method and resets the position of the player to a specific position. The addMoles() method has been modified so that the number of moles increase with levelNumber and the max fullness of each mole is proportional to levelNumber so that the levels would become more difficult gradually.

Task 16 – Better player and mole placement

For this task, 2 integer class attributes baseX and baseY are declared. In the generateLevel() method, a second nested for loop is added that iterates over the level array and stores the X and Y coordinate of the first BASE tile it finds. Then, baseX and baseY variables are used in createPlayer() and placePlayer() methods to always set the initial position of the player in a BASE tile.

Task 17 – Power ups

In the movePlayer() method, if a player completely mines a tile, its tiletype is stored in a TileType object variable t. If t is equal to URANIUM, the integer class attribute miningStrength is permanently set to 25. The nextLevel() method includes a code to make sure the value of miningStrength goes back to 5 when the level changes.