Awais Hussain

UB: 19009475

# Task 1 – Level Generation

The level was generated using an array-type method. The array is constructed as ‘tiles’ and represents the types of tiles that will span across the entire array to create a floor, which is why the array is contained with DUNGEON\_WIDTH and DUNGEON\_HEIGHT, which is the maximum (final) value of the entire array, which essentially creates the entire level. The level is created with a for-loop of type int, taking the value of COLUMN, indicating if it is less than DUNGEON\_HEIGHT/WIDTH, it will keep increasing until it hits maximum value. The Dungeon height is nested in the dungeon width, so an entire height (column) will be filled with the tiles, then once a column is complete, it continues to the next row to fill up the column. This repeats until the array is filled.

An if statement is created within the dungeon width and height, using the random class of next float (since the ENUM variable WALL\_CHANCE holds a double value of 0.15, representing 15%), to then say “if WALL\_CHANCE” is less than the RNG value, it will create tiles that are walls, with a 15% chance of them spawning on every square, creating a randomly generated map with 15% of the map contained with walls. However, there is an ‘else’ statement put after this to say “otherwise, the rest shall remain as the floor tile type” through tiles[ROW][COLUMN] = TileType.FLOOR; .

A final important tile to note is the “stairs tile”, which is set separately under the rest of the code, which grabs a random int of the height and width, putting stairs between them so the player can progress: tiles[rng.nextInt(DUNGEON\_WIDTH)][rng.nextInt(DUNGEON\_HEIGHT)] = TileType.STAIRS;

The tiles will then return once the ‘return tiles’ code is executed.

# Task 2 – Player Spawning

The ‘getSpawns’ method is required to set the data to create spawning points for every change of the level. This is useful for when the levels change and when the game starts, locating enemies and the player at random locations on the floor tile.

The spawnPlayer method consists of using an integer, using the RNG class to create a random number to spawn the player in a location. To take safety measures here, there is an Entity constructor that is used to build an entity **(which is contained in the Entity class**), in this case it is the player. The player is created to have 50 max health, to get the value of a random X axis and random Y axis, to place the entityType of player. Spawns.remove(random); prevents the player from overlapping with monsters or other entities. The value is returned.

# Task 3 – Player Movement

The player entity is called into the method. The player movement is separated into 4 methods, which separates moving into 4 directions. movePlayerLeft, movePlayerRight, movePlayerDown and movePlayerUp. Each of the methods are very similar but have slight differences to match their methods. Each of them consists of a declared variable (x and y) that gets the coordinates of the player, and if the player is spawned less than the 0th value from RNG, the statement returns to prevent any bugs, crashes, or compiler errors. This is to prevent the player or enemies from creeping out of bounds as that will create an error as there is no data to access in those areas. If the player were to move right, they must be less than the max value of X which is the dungeon width. The same concept applies with the Y axis, so the player or enemy does not fall out of bounds from the top or bottom of the array.

The code has a second if statement to prevent the player from walking into the wall, saying if tiles[x][y] is equals to the tileType of wall (so if the player is attempting to go into the wall) then return (stay in the same position).

A final loop is created which allows hurting of the monster, which says “if the monster exists, then get the X and Y axis of all monsters in the monsters array”. Then it hits the monster that is adjacent to the player in the array that attempts to bump into it.

# Task 4 – Monster Generation

Monsters are assigned a value of an array as ‘enemy’ within the spawnMonsters method to keep track of what the enemies are. A loop is created saying “if the number is less than the number of enemies on a level (since it creates a new Entity[depth]), then increment by 1. This will increase the number of monsters per level from the beginning, as depth represents the levels (how far you are into the game).

The entity constructor is called to give the monster 10 health and to create coordinates for it to spawn somewhere **(as it’s created with 3 values through the Entity class),** while fetching the type “MONSTER” to somewhere that is not a wall and that is not itself (or more monsters), which is why ‘spawns.remove(random);’ is included to prevent that issue. Example of constructor of creating an enemy: enemy[i] = new Entity(10, (int)spawns.get(random).getX(), (int)spawns.get(random).getY(), EntityType.MONSTER);

The statement then returns and creates an enemy.

# Task 5 – Monster Movement

Moving the monster required creating a few variables to assign an RNG movement value and the x and y coordinates for the monster, that is indicated as ‘m’ (the signature of the method). The switch statement holds 4 cases which sets the position of the monster, m.setPositions(x++,y); for example, would shift the monster entity to the right in the 2d array, and the other cases will do movements for all directions. Up = y - - Down = y + + Left = x - - and Right = x + +. The switch statement contains rngMove=rng.nextInt(4), saying that any of the 4 statements will execute (with the exception of the monster colliding into walls or the inner bounds of the 2d array).

A set of if statements are created, and similarly like the player movement, there is implemented code to prevent the monster from going out of bounds, as this will cause errors in the compiler. This stays within the X and Y values of the 2d array to prevent the “no data” error from happening.

# Task 6 – Combat

Combat is embedded within the movement of the player; however, the combat is one-sided as the player can only kill the entities. When the player is adjacent to the monster, moving into the same square as the monster does damage to the monster, while keeping the player in the same position to prevent overlapping. This was implemented in all movement types for the player so it would take damage from any sides of movement as hitMonster(monster[i]) (as ‘I’ in the braces of monster indicates the one monster the player is attacking).

# Task 7 – Descending Levels

When the levels descend, in the method of decendLevel, there are variables that do different things when a new level is entered. Depth++ is used in a previous method when generating monsters and completing the level with the stairs, which recreates the RNG of the walls, the spawns for the enemy and the spawn for the player. getSpawns is the method that is used in the method that creates random numbers to create spawn points for the entities (player, monster). The monsters variable would indicate spawnMonsters, which would create the monster entity and place it into the array. The value of int depth that is declared in the class starts at 1, indicating the current dungeon level, which increases at spawnMonsters, as the array “enemy” changes the spawn location and spawn increase of the monsters. Entity[] enemy = new Entity[depth]; So, whenever we refer to enemy, we are referring to the array of monsters that are going to be spawned.