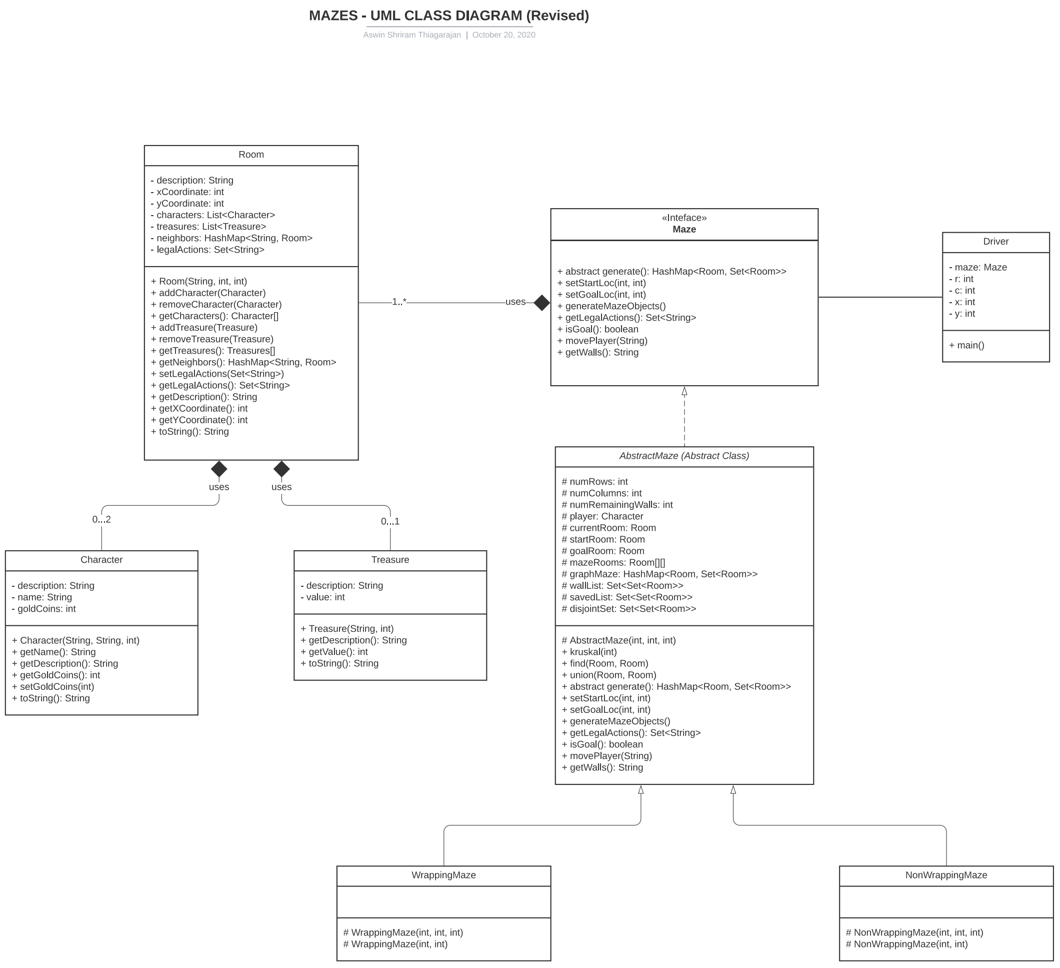
- **UML Class Diagram (Updated):**



- **Design Explanation:**

    - **Package Name:** maze
    - **Maze Interface:**
        - The Maze interface consists of all the methods that needs to be implemented by the maze class.
        - The Setter methods **setStartLoc(int, int)** and **setGoalLoc(int, int)** are used to set the locations of starting point and goal point by the user.
        - The **generate()** and **generateMazeObjects()** are used to generate the maze of required type and are used to instantiate the objects that are present in the maze.
        - The **getLegalActions()** method will generate the possible actions that a player can take at that current location.
        - The **movePlayer(actions)** method will move the player in that specified direction.
        - The **isGoal()** method is used to check if the current room is the goal location or not.

    - The **getWalls()** will provide the list of walls present in the maze.
- **AbstractMaze Class:**
    - The AbstractMaze class implements the Maze interface and is defined as abstract to offer encapsulation and enable DRY (Don't Repeat Yourself) for code refactoring.
    - The disjoint set consists of all the set of rooms.
    - The method **kruskal(number of walls)** uses a modified version of kruskal's algorithm for maze generation. It has helper methods **union(room1, room2)** and **find(room1, room2)** to help in combining the rooms that don't have wall between them and checking whether the two rooms belong to the same disjoint set respectively. This maze generator uses union find disjoint set method for generation.
- **WrappingMazeClass:**
    - This class inherits the AbstractMaze class. This class is defined for wrapping mazes.
    - This class has two constructors to deal with Perfect Maze and Non Perfect Maze types.
    - The perfect maze takes just number of rows and columns as input, whereas non perfect maze takes additional parameter number of remaining walls.
- **NonWrappingMaze Class:**
    - This class inherits the AbstractMaze class. This class is defined for Non wrapping mazes.
    - This class has two constructors to deal with Perfect Maze and Non Perfect Maze types.
    - The perfect maze takes just number of rows and columns as input, whereas non perfect maze takes additional parameter number of remaining walls.
- **Room Class:**
    - The rooms are the building blocks of the maze.
    - The room class consists of fields description – description about the room, location of the room (defined by X and Y Co-ordinate), List of characters present in the room, List of treasures present in the room, neighbor list – list of neighbor rooms, legal actions – actions that a player can take in the current room.
    - The room can either be a starting room which can only have players and no treasures or thieves, or a goal location which is the termination state of the game and also can't have treasures or thieves. Other rooms can have player, thief, treasure or all.
    - The getter methods of the room class are used to get the information about the room and query them by the maze class.
    - The setter methods are used to instantiate the player, treasures, and thieves all around the maze.
    - The **toString()** method is overwritten to display the stats and summary of a character object.
    - The room class has '***has-a***' relationship with the Maze class via the Maze interface. There should be atleast one room in a maze. The rooms cannot exist if there is no maze.
- **Character Class:**
    - The character class consists of fields (Like Name of the character, description about the character, and amount of gold coins they have) that describe characters that are part of the maze. There are two types of characters in this game. The player and thieves. The player's aim is to collect the gold coins and finish the maze, whereas the thief is present to rob the player and vanish.

        - The character has getter methods like **getName()**, **getDescription()** and **getGoldCoins()** to get the respective stats.
        - The character has setter method like **setGoldCoins(goldCoins)** which can be used when a player picks up the gold.
        - The **toString()** method is overwritten to display the stats and summary of a character object.
        - The character class has '***has-a***' relationship with the room class. There can be max 2 characters in a room at a time (one player and one thief). The characters cannot exist if there are no rooms.
    - **Treasure Class:**
        - The treasure class consists of fields (Like Name of the treasure, and value of gold coins) that describe treasures that are part of the maze. There is only one type of treasure in this game and they are gold coins.
        - The treasure has getter methods like **getName()** and **getValue()** to get the respective stats of the treasure.
        - The **toString()** method is overwritten to display the stats and summary of a treasure object.
        - The treasure class has '***has-a***' relationship with the room class. There can be max 1 treasure in a room at a time. The treasures cannot exist if there are no rooms.
    - **Driver Class:**
        - The driver class is where the main function resides.
        - The main function is where a quick game demo is played.
        - The demo is interactive. The user can either run the driver.java file and the program is intuitive enough and will ask for commands from the user, so they can play the game.
        - The demo can be run from a .jar file and the instructions are specified in the README.md file.

- **Testing Plan:**

    The classes to be tested are tested in separate test classes, to enable modularity in testing and also to cover all the functionalities of the respective classes.

    - **Test 1:** To check if an object of the class maze is created properly.
    - **Test 2:** To check if an object of the class room is created properly.
    - **Test 3:** To check if an object of the class character is created properly.
    - **Test 4:** To check if an object of class treasure is created properly.
    - **Test 5:** To check if the maze is created correctly based on the inputs number of rows and columns, number of remaining walls.
    - **Test 6:** To check if the walls are properly deleted.
    - **Test 7:** To check if the kruskal's algorithm works properly.
    - **Test 8:** To check if the gold coins are correctly spawned in the maze.
    - **Test 9:** To check if the characters (player and thieves) are correctly spawned in the maze.
    - **Test 10:** To check if an exception is thrown if an invalid input is given to the maze.
    - **Test 11:** To check the number of walls is correctly remaining for each type of maze.
    - **Test 12:** To check if the treasure value of the player is correctly modified after picking up gold and getting robbed by a thief.
    - **Test 13:** To check if the gold vanishes after the player has picked up the gold.
    - **Test 14:** To check if the thief vanishes after the player has been robbed.
    - **Test 15:** To check if the player has reached the goal.
    - **Test 16:** To check if player is in the same location after an invalid move.

- **<u>Test 17 – 30:</u>** To check for other individual public methods of the classes Treasure, Room, Character including a wider coverage of testing.

- **Assumptions Made:**
    - The gold and the thief can be in the same room.
    - When a player visits a room such as above, they first pick up the gold coins and then get robbed by the thief.
    - The player can’t be robbed below zero, in that case their gold coin value is set to 0.
    - The gold and thief locations are spawned randomly and no seed is set in the program.
    - The player remains in the same location after an invalid move.

- **Challenges Faced:**
    - The biggest challenge I faced is when choosing datastructures to represent walls, and rooms of the maze. It took a lot of trial and error to finalize on the Set data structure.
    - Use of relationships between classes were a challenge. That is, whether to use “Association” or “Composition” relationship between the room and the maze.
    - The design decision of keeping the generate game into 2 different components generate() and generateMazeObjects().