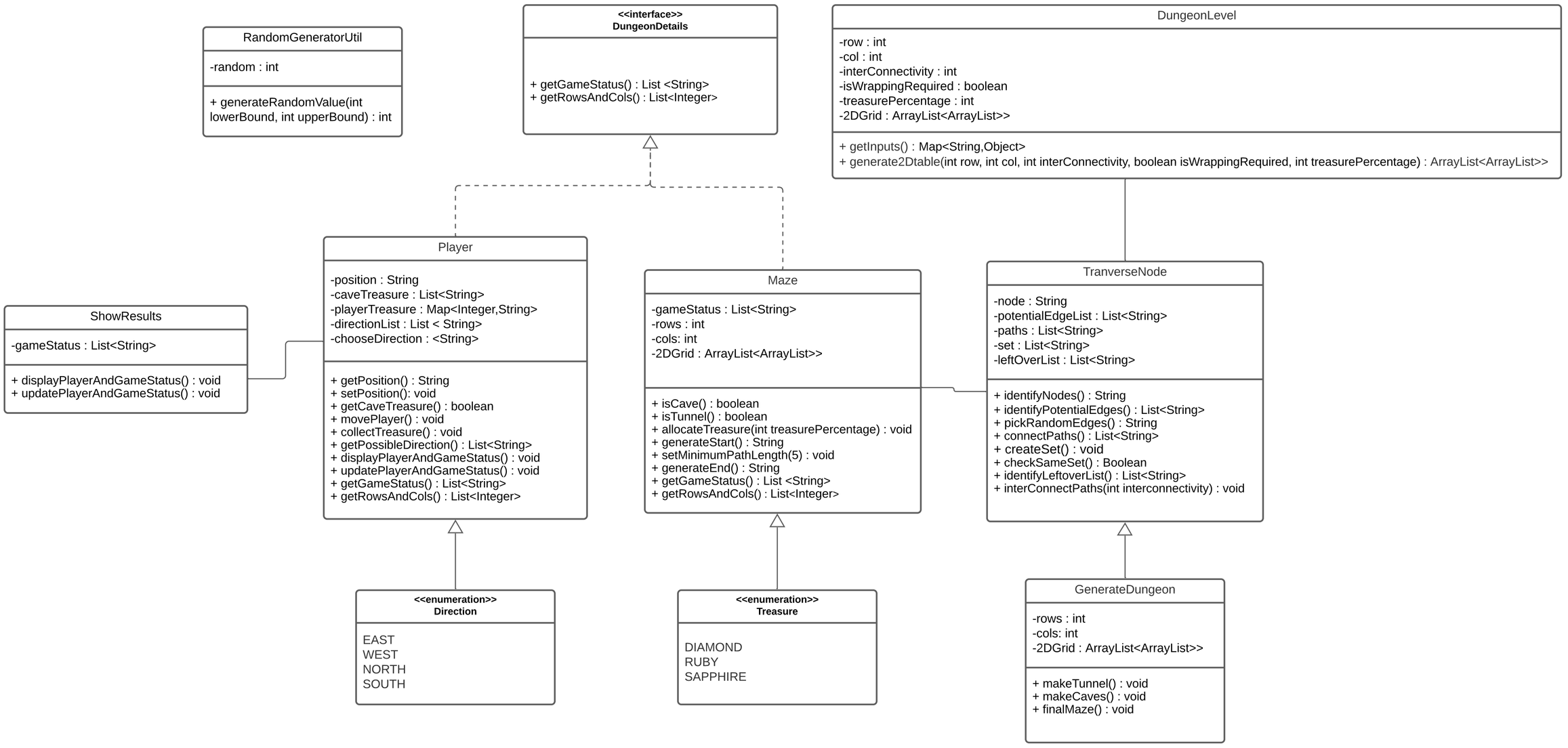
UML DESIGN


RandomGeneratorUtil
-random : int
+ generateRandomValue(int lowerBound, int upperBound) : int
<<interface>>
DungeonDetails
+ getGameStatus() : List <String>
+ getRowsAndCols() : List<Integer>
DungeonLevel
-row : int
-col : int
-interConnectivity : int
-isWrappingRequired : boolean
-treasurePercentage : int
-2DGrid : ArrayList<ArrayList>>
+ getInputs() : Map<String,Object>
+ generate2Dtable(int row, int col, int interConnectivity, boolean isWrappingRequired, int treasurePercentage) : ArrayList<ArrayList>>
Player
-position : String
-caveTreasure : List<String>
-playerTreasure : Map<Integer,String>
-directionList : List < String>
-chooseDirection : <String>
+ getPosition() : String
+ setPosition(): void
+ getCaveTreasure() : boolean
+ movePlayer() : void
+ collectTreasure() : void
+ getPossibleDirection() : List<String>
+ displayPlayerAndGameStatus() : void
+ updatePlayerAndGameStatus() : void
+ getGameStatus() : List<String>
+ getRowsAndCols() : List<Integer>
Maze
-gameStatus : List<String>
-rows : int
-cols: int
-2DGrid : ArrayList<ArrayList>>
+ isCave() : boolean
+ isTunnel() : boolean
+ allocateTreasure(int treasurePercentage) : void
+ generateStart() : String
+ setMinimumPathLength(5) : void
+ generateEnd() : String
+ getGameStatus() : List <String>
+ getRowsAndCols() : List<Integer>
TranverseNode
-node : String
-potentialEdgeList : List<String>
-paths : List<String>
-set : List<String>
-leftOverList : List<String>
+ identifyNodes() : String
+ identifyPotentialEdges() : List<String>
+ pickRandomEdges() : String
+ connectPaths() : List<String>
+ createSet() : void
+ checkSameSet() : Boolean
+ identifyLeftoverList() : List<String>
+ interConnectPaths(int interconnectivity) : void
ShowResults
-gameStatus : List<String>
+ displayPlayerAndGameStatus() : void
+ updatePlayerAndGameStatus() : void
<<enumeration>>
Direction
EAST
WEST
NORTH
SOUTH
<<enumeration>>
Treasure
DIAMOND
RUBY
SAPPHIRE
GenerateDungeon
-rows : int
-cols: int
-2DGrid : ArrayList<ArrayList>>
+ makeTunnel() : void
+ makeCaves() : void
+ finalMaze() : void

TESTING PLAN :

All the following tests are done using JUNIT4 unit tests. It is done using the assert methods in order test the working of every unit individually.

Test Case :	Method Name :	Result :
Check if the input values are valid: • row – nonnegative, within maximum and minimum range • column - nonnegative, within maximum and minimum range • interconnectivity – nonnegative • isWrapping – boolean • treasurePercentage – within 0-100	testInputs()	True or False.
Check if the 2DGrid is plotting with correct number of inputs • rox x col number of nodes must be created	test2DGrid	True or False
Check if nodes are identified and stored for later use	testNodes()	True or False
Check if all potential edges are identified	testPotentialNodes()	True or False
Check if all edges are picked • rol x col number of edges must be picked	testAllEdgesPicked()	True or False
Check if all potential edges are in the list • i.e there must be a path between every two nodes in the grid	testPotentialEdgesList()	True or False
Leftover Node check : • Check if all leftover edges are identified properly • Check if all leftover edges are stored properly	testLeftOverEdges()	True or False
Check if number of interconnections are connected	testInterconnections()	True or False
Check if tunnels are properly generated • Check if tunnels are created without the possibility of having treasure in it	testTunnels()	True or False
Check if caves are properly generated	testTunnels()	True or False

• Check if caves are created with the possibility of having treasure in it		
Check if every node is either tunnel or cave • Every node must be a tunnel or a cave	testNode()	True or False
Verify if treasure allocated properly	testAllocatedTreasure()	True or False
Verify if the % of treasure allocated is equal to the input • Must be allocated according the percentage provided by the user	testPercentageTreasure()	True or False
Verify start point generated as per the algorithm • Must belong to the node list of the 2DGrid	testStart()	True or False
Verify end point generated as per the algorithm • Must belong to the node list of the 2DGrid	testEnd()	True or False
Check minimum distance between the path from start to end • Distance between start and end must be at least 5	testDistance()	True or False
Check if position is valid and returned properly	testPosition()	True or False
Check if the moves of the player is updated properly	testUpdatePosition()	True or False
Check if room treasure exists and displayed properly • Treasure in the particular node is tested	testTreasureInRoom()	True or False
Check if player treasure exists and displayed properly • Treasure from all nodes visited must be added to player and tested	testTreasureOfPlayer()	True or False

Check if the list of direction the player can move in from a particular node is printed properly	testDirectionList()	True or False
Check if the player moves in the right direction to the right node	testDirection()	True or False
Check if player and game status is updated based on player • At every point, the value must be updated properly	testUpdateStats()	True or False

More test cases can be added for the different cases that we come across as we develop the project.