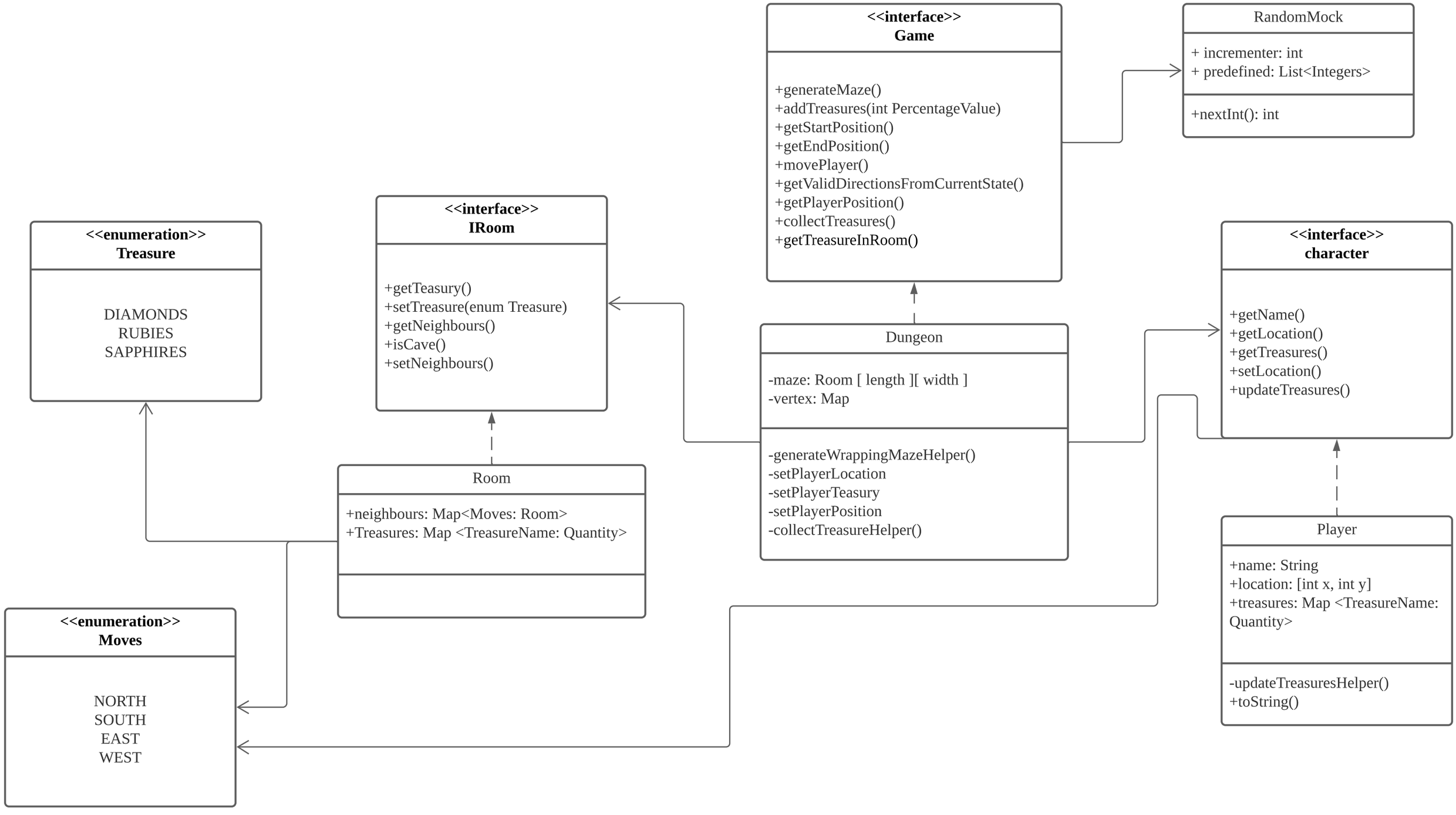
<<interface>>
Game
+generateMaze()
+addTreasures(int PercentageValue)
+getStartPosition()
+getEndPosition()
+movePlayer()
+getValidDirectionsFromCurrentState()
+getPlayerPosition()
+collectTreasures()
+getTreasureInRoom()
RandomMock
+ incrementer: int
+ predefined: List<Integers>
+nextInt(): int
<<enumeration>>
Treasure
DIAMONDS
RUBIES
SAPPHIRES
<<interface>>
IRoom
+getTeasury()
+setTreasure(enum Treasure)
+getNeighbours()
+isCave()
+setNeighbours()
Dungeon
-maze: Room [length][width]
-vertex: Map
-generateWrappingMazeHelper()
-setPlayerLocation
-setPlayerTeasury
-setPlayerPosition
-collectTreasureHelper()
<<interface>>
character
+getName()
+getLocation()
+getTreasures()
+setLocation()
+updateTreasures()
Room
+neighbours: Map<Moves: Room>
+Treasures: Map <TreasureName: Quantity>
Player
+name: String
+location: [int x, int y]
+treasures: Map <TreasureName: Quantity>
-updateTreasuresHelper()
+toString()
<<enumeration>>
Moves
NORTH
SOUTH
EAST
WEST

Test Plan for Project 3 - Dungeon		
Testing Player	**Input**	**Expected Output**
Get Player name	Player1	"Player1"
Get Player Location		[2,4]
Get Player Treasure		{Diamonds: 2, Rubies: 1, Sapphires: 0}
Set Player Location, Get Player Location	2,5	[2,5]
Update Player Treasure, Get Player Treasure	Sapphires	{Diamonds: 2, Rubies: 1, Sapphires: 1}
Player toString		"Player 1 -> Treasures: {Diamonds: 2, Rubies: 1, Sapphires: 1}"
Testing Room	**Input**	**Expected Output**
Get Treasury		{Diamonds: 1, Rubies: 0, Sapphires: 0}
Set Treasury -> Get Treasury	Rubies	{Diamonds: 1, Rubies: 1, Sapphires: 0}
Set Neighbour	Room1, NORTH	
Get Neighbours		{NORTH: Room1, SOUTH: null, EAST: null, WEST: null}
Is Room a Cave		FALSE
Testing Dungeon	**Input**	**Expected Output**
Test Grid height	6	8
Test Grid width	8	8
Test invalid Grid Height	-2	Illegal Argument Exception
Test invalid Grid Width	0	Illegal Argument Exception
Test Grid Dimension -> get grid height	6,8	6
Test invalid Grid Dimension	0,4	Illegal State Exception
Test interconnectivity	-2	Illegal Argument Exception
Test add tresures		
Test minimum path length between start and end	[0,0], [6,4]	TRUE
Test get Valid Directions from State	State: Room 7	{NORTH: Room1, SOUTH: null, EAST: null, WEST: null}
Test get player position		[2,5]
Test Collect Tresure -> getTreasureInRoom	Room1	{Diamonds: 0, Rubies: 0, Sapphires: 0}
Test Wrapping dungeon -> get valid direction from state	Room1	{NORTH: Room41, SOUTH: Room9, EAST: Room2, WEST: Room8}
Test Non wrapping dungeon -> get valid direction from state		{NORTH: null, SOUTH: Room9, EAST: Room2, WEST: null}
Testing MockRandom	**Input**	**Expected Output**
Test next int	[4,2,1,3]	4
Test next int	[4,2,1,3]	2