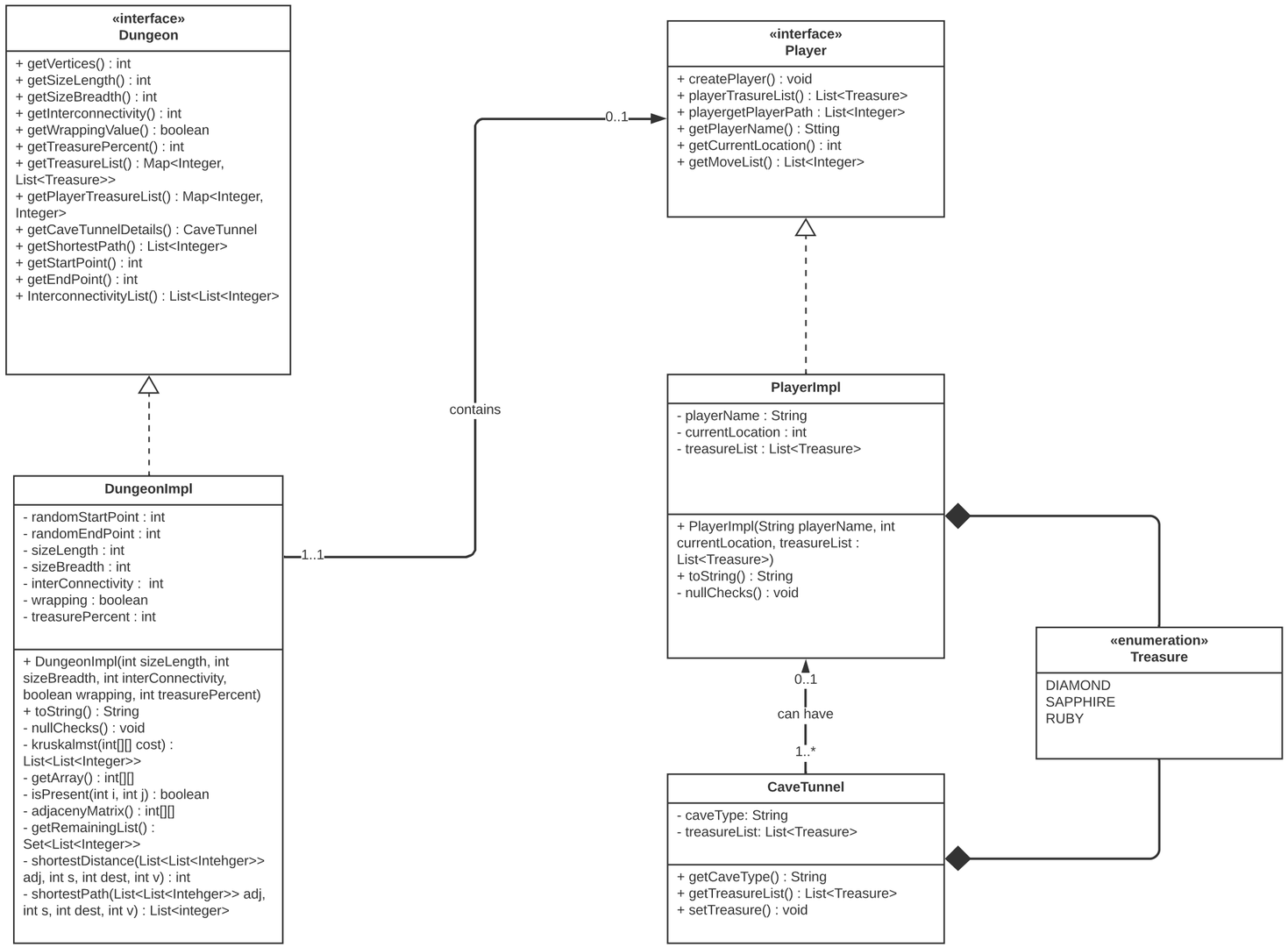
«interface»
Dungeon
+ getVertices() : int
+ getSizeLength() : int
+ getSizeBreadth() : int
+ getInterconnectivity() : int
+ getWrappingValue() : boolean
+ getTreasurePercent() : int
+ getTreasureList() : Map<Integer, List<Treasure>>
+ getPlayerTreasureList() : Map<Integer, Integer>
+ getCaveTunnelDetails() : CaveTunnel
+ getShortestPath() : List<Integer>
+ getStartPoint() : int
+ getEndPoint() : int
+ InterconnectivityList() : List<List<Integer>
«interface»
Player
+ createPlayer() : void
+ playerTrasureList() : List<Treasure>
+ playergetPlayerPath : List<Integer>
+ getPlayerName() : Stting
+ getCurrentLocation() : int
+ getMoveList() : List<Integer>
0..1
contains
1..1
PlayerImpl
- playerName : String
- currentLocation : int
- treasureList : List<Treasure>
+ PlayerImpl(String playerName, int currentLocation, treasureList : List<Treasure>)
+ toString() : String
- nullChecks() : void
DungeonImpl
- randomStartPoint : int
- randomEndPoint : int
- sizeLength : int
- sizeBreadth : int
- interConnectivity : int
- wrapping : boolean
- treasurePercent : int
+ DungeonImpl(int sizeLength, int sizeBreadth, int interConnectivity, boolean wrapping, int treasurePercent)
+ toString() : String
- nullChecks() : void
- kruskalmst(int[][] cost) : List<List<Integer>>
- getArray() : int[][]
- isPresent(int i, int j) : boolean
- adjacenyMatrix() : int[][]
- getRemainingList() : Set<List<Integer>>
- shortestDistance(List<List<Intehger>> adj, int s, int dest, int v) : int
- shortestPath(List<List<Intehger>> adj, int s, int dest, int v) : List<integer>
0..1
can have
1..*
CaveTunnel
- caveType: String
- treasureList: List<Treasure>
+ getCaveType() : String
+ getTreasureList() : List<Treasure>
+ setTreasure() : void
«enumeration»
Treasure
DIAMOND
SAPPHIRE
RUBY

# TESTING PLAN FOR DUNGEON MODEL

1. Check for null values for Player Implement class for values like Player Name, Current Location and Player collected Treasure List.
2. Check for null values for Dungeon Implement class for values like starting Point, Ending Point, size attributes.
3. Check for getter methods for Dungeon.
4. Check for getter methods for Player.
5. Check if Player Details are correctly populated or not.
6. Check toString() function in Dungeon Generation.
7. Check for the validity of move list which the player is making when inside the dungeon.
8. Check for the current location of the player if correct or not.
9. Check if the Player Treasure Collected is correct or not.
10. Check if the Player Location is correctly updated when he moves to new location.
11. Check if the Treasure Populated in the Dungeon is 20% of the available nodes present in it.
12. Check toString() function for player Implementation class.
13. Check if Move Player function to be used by the user is correctly moving player to the expected location or not.
14. Check if the Dungeon nodes are linked from both edges in case of wrapping dungeon type.
15. Check if the Dungeon nodes are not linked to both edges in case of non-wrapping dungeon type.
16. Check if the degree of interconnectivity in dungeon has more than one or more possible paths from start point to end point.
17. Check if Player North move is working correctly or not.
18. Check if Player West move is working correctly or not.
19. Check if Player South move is working correctly or not.
20. Check if Player East move is working correctly or not.
21. Check if 2 or more caves with only 2 entries are formed into Tunnel.
22. Check if the distance between Starting and End point is less than 5 units then it should throw error.
23. Check if the Cave/Tunnel Node can hold more than 1 treasure items.