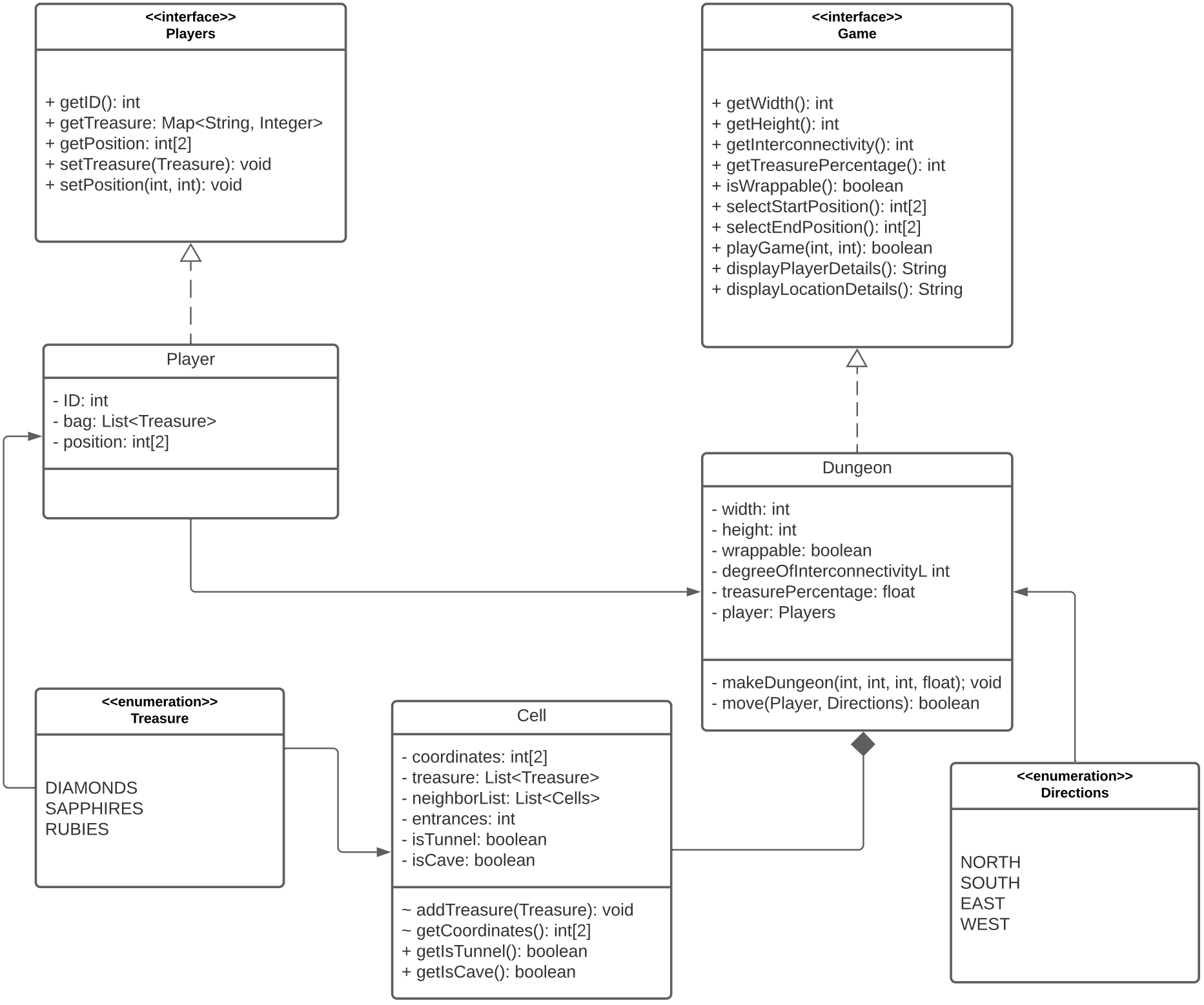
<<interface>>
Players
+ getID(): int
+ getTreasure: Map<String, Integer>
+ getPosition: int[2]
+ setTreasure(Treasure): void
+ setPosition(int, int): void
<<interface>>
Game
+ getWidth(): int
+ getHeight(): int
+ getInterconnectivity(): int
+ getTreasurePercentage(): int
+ isWrappable(): boolean
+ selectStartPosition(): int[2]
+ selectEndPosition(): int[2]
+ playGame(int, int): boolean
+ displayPlayerDetails(): String
+ displayLocationDetails(): String
Player
- ID: int
- bag: List<Treasure>
- position: int[2]
Dungeon
- width: int
- height: int
- wrappable: boolean
- degreeOfInterconnectivityL int
- treasurePercentage: float
- player: Players
- makeDungeon(int, int, int, float); void
- move(Player, Directions): boolean
<<enumeration>>
Treasure
DIAMONDS
SAPPHIRES
RUBIES
Cell
- coordinates: int[2]
- treasure: List<Treasure>
- neighborList: List<Cells>
- entrances: int
- isTunnel: boolean
- isCave: boolean
~ addTreasure(Treasure): void
~ getCoordinates(): int[2]
+ getIsTunnel(): boolean
+ getIsCave(): boolean
<<enumeration>>
Directions
NORTH
SOUTH
EAST
WEST

TESTING PLAN FOR PROJECT 3 (DUNGEON MODEL)

Player Class:

---

1. Check for uniqueness of the ID for the player.
2. Check for null value while creating the copy of the player object.
3. Check for null/invalid values while adding newly found treasure to the bag of treasure that player has.
4. Check for invalid values for position arguments while updating the position of the player during the game.
5. Check if getTreasure() returns the bag of treasure that the user has as expected.
6. Check if the ID of the player is generated and returned as expected.
7. Check if the position of the player is updated and returned as expected.

Dungeon Class:

---

1. Check for illegal arguments given for creating the dungeon.
    1. The width and height of the dungeon should not be negative or zero.
    2. The degree of interconnectivity should not be negative.
    3. The treasure percentage should not be zero or negative.
    4. The player object should not be null.
2. Check that the makeDungeon method creates the dungeon as expected or not.
3. Check that non-wrapping dungeons do not have wrapping edges.
4. Check that wrapping dungeons have wrapping edges below the interconnectivity.
5. Check if the startPosition and endPosition are valid. (Have a distance of at least 5 between them).
6. Check if the player details are being printed correctly or not.
7. Check if the location details are being printed correctly or not.
8. Check if playGame method is handling the game according to the rules or not.
9. Check if the game ends when you reach the end position.
10. Check if cave's treasure becomes empty when the player picks it.
11. Check if the playGame method is asking directions to the user and if the user input is valid or not.
12. Check if move method is working as expected.
    1. If the dungeon is wrappable, it should allow for player to move to the other end should the choice be available.

    2. It should throw an error saying that the desired path is not possible because there is no entrance.
13. Check if the direction given to the move method is valid.
14. Check if the treasure percentage is satisfied.
15. Check if all the available treasures are generated in that treasure percentage.
16. Check if the interconnectivity is satisfied while creating the dungeon.
    1. Can be tested by taking the maximum possible paths including the interconnectivity and checking if the dungeon satisfies the condition.

Cell Class:

---

1. Check if the addTreasure method is adding the treasure as expected.
2. Check that the treasure input is not null.
3. Check if the getEntrances method is returning the number of entrances as expected.
4. Check if the isTunnel and isCave methods are returning boolean as expected.