Arkham Asylum

Killing the joker

MovePlayerCommand
-scanner: Scanner
-out:Appendable

+MovePlater(roomName: string)
+execute(world):void

CreateMapWorldCommand
-scanner: Scanner
-out:Appendable

+CreateMapWorldCommand(world)
+execute(worldString):void

GetNeighbourInfoCommand
-scanner: Scanner
-out:Appendable

+GetNeighbourInfoCommand(
neighbourInfostring)
+execute(world):void

PickUpItemCommand
-scanner: Scanner
-out:Appendable

+PickUpItemCommand(
pickUpItemCommandString : String)
+execute(world):void

DisplayWorldInfoCommand
-scanner: Scanner
-out:Appendable

+DisplayInfoCommand(world :
World)
+execute(world):void

AddPlayerCommand
-scanner: Scanner
-out:Appendable

+AddPlayerCommand(world,
Player)
+execute(world)

DisplayInfoOfPlayerCommand
-scanner: Scanner
-out:Appendable

+DisplayInfoOfPlayerCommand(world,
playerString)
+execute(world)

LookAroundCommand
-scanner: Scanner
-out:Appendable

+LookAroundCommand(world,
playerString)
+execute(world)

<<interface>>
ArkhamAsylumGameCommand

+execute(ArkhamAsylumWorld world)

DisplayInfoOfSpecificPlace
-scanner: Scanner
-out:Appendable

+DisplayInfoOfPlayerCommand(world,
placeString)
+execute(world)

WorldControllerImpl
-appendable:Appendable
-readable: Readable
-maximumNoOfPlayer: Integer
-command:Map<String, Function<Scanner, command>>

+WorldController(appendable:Appendable,
readableReadable)
+playGame(worldModel: ArkhamAsylumWorld)

<<interface>>
WorldCreator

+playGame(readable: Readable)

<<interface>>
ArkhamAsylumGameCreatorInterface

+parseAllData()
+getDrLuckyWorld()

MovePetCommand
-scanner: Scanner
-out:Appendable

+DisplayInfoOfPlayerCommand(world,
placeString)
+execute(world)

AttackCommand
-scanner: Scanner
-out:Appendable

+AttackCommand(world, placeString)
+execute(world)

ArkhamAsylumGameWorldCreator
- numberOfSpaces:int = 0
- numberOfItems:int = 0
-drLuckyWorldImpl:DrLuckyWorldImpl
-spaceList:ArrayList<SpaceImpl>
-weaponList:ArrayList<Weapon>
-rowCount:int = 0
-colCount:int = 0
-fileContect:String

+parseAllData()
+getDrLuckyWorld()
-parseWeaponData()
-parseSpaceData()
-parseTargetData()
-createDrLuckyWorld()

<<interface>>
ArkhamAsylumWorld

+checkWhoCanSeeTheSpace(
spaceName:String)
:ArrayList<String>
+getWorldName():String
+getWorldDetails():String
+getTargetCharacterInfo():Target
+getTargetLocation():String
+getItemsInsInRoom():ArrayList<String>
+movePlayer():boolean
+getPlayerTurn():Player
+addPlayer(playerName: String,
roomId : Integer):boolean
+getPlayerDetails(playerName:
String):String
+getSpaceDetails(placeName:
String):String
+attack(itemName:String):Boolean
+getClue():String
+isGameOver():Boolean

ArkhamAsylumWorldImpl
-spacesList: List<Space>
-tagetCharacter:Taget
-itemsList:List<Items>
-playerList:List<Player>
-rowCount: Int
-colCount:Int

-getNumberPlayers():ArrayList<String>
-getNeighnoringSpanceLocation():
ArrayList<ArrayList<Integer>[0..3>
-getNumberOfNeighbors():Integer
-ifAnyNeighbours():Boolean
+checkWhoCanSeeTheSpace(ispaceId:int)
:ArrayList<String>
-createGameWorldMap()
-isOverLapping():Boolean
-isXinRange():Boolean
-checkXaxisForNeighbors(x1:int, x2:int, x3:int,
x4:int):Boolean
-checkYAxisForNeighbors(int y1, int y2, int y3, int y4)
+getWorldName():String
+getWorldDetails():String
+getTargetCharacter():Target
+getTargetLocation():String
+getItemsInRoom():ArrayList<String>
+movePlayer():boolean
-getItemsWithPlayer(playerName):
ArrayList<Items>
+addPlayer((playerName: String,
roomId : Intege) :boolean
+getPlayerDetails(playerName: String):String
+getSpaceDetails(placeName: String):String
+attackTarget(itemName:String):Boolean
+movePet(spaceName):String
-isPlaceVisible(spaceName:String):Boolean
+getClue():String
+isGameOver():Boolean

<<interface>>
PlayerCharacter

+ getCharacterType() :CharacterType
+ getPlayerPosition() : String
+getPlayerName():String
+getPlayerItems():ArrayList<Item>
+addItemsToPlayer(item:Items)
+getMaximumCount(): int

<<interface>>
TargetCharacter

+ getCharacterType(): CharacterType
+ getCharacterPosition():
ArrayList<Integer>[0..3]
+decreaseTargetHealth
(decreaseHealthBy:Integer):Integer
+moveTargetCharacter():Integer
+getTargetHealth():Integer

<<interface>>
Items

+ getItemsType(): Pair<integer, String>
+ getDamage(): Integer
+ getItemRoomLocation(): List<Integer>[0..3]

<<interface>>
Space

+ getSpaceType(): SpaceType
+ getSpaceLocation(): Arraylist<Integer>[0..3]
+ getRoomId(List<Integer>[..3]):Integer

<<interface>>
Pet

+getLocation():String
+getName():String
+movePet():String

PlayerImpl
-characterType:CharacterType
-playerPosition:ArrayList<Integer>[0..3]
-nameOfPlayer:String
-itemsWithThePlayer:ArrayList<Items>
-maximumItemCount:int

+player(playerName:String, playerPosition
:ArrayList<Integer>[0..3], itemsWithThePlayer)
+ getCharacterType() :CharacterType
+ getPlayerPosition() : ArrayList<Integer>[0..3]
+moveToNextSpace():Integer
+getPlayerName():String
+getPlayerItems():ArrayList<Items>
+addItemsToPlayer(item:Items)

Target
- characterType:CharacterType
- targetPosition:ArrayList<Integer>[0..3]
- tagetHealth:Integer = 0

+ Target(targetPosition:ArrayList<Integer>[0..3],
targetHealth:Integer)
+ getTargetHealth(): Integer
+ getCharacterType(): CharacterType
+ getCharacterPosition(): Integer
+decreaseTargetHealth(decreaseHealthBy):Integer
+toString():String
+ moveToNextSpace():Integer
-checkTargetLocationForOutOfBound():Boolean
+getTargetHealth():Integer

Weapon
-weaponType:Pair<integer, String>
-damageValue:Integer = 0
-weaponUsableLocation:Integer

+Weapon(weaponName:String, damagaeValue:
Integer, ItemRoomLocation: List<Integer>[0..3])
+ getItemsType(): Pair<Integer, String>
+ getDamage(): Integer
+ getItemRoomLocation(): Integer

Room
-roomType:String
-roomPosition:ArrayList<Integer>[0..3]
-numberOfPlayers:Integer = 0
-neighborLocation:ArrayList<ArrayList<Integer>[0..3]
-numberOfNeightbors:Integer = 0
-ifNeighbourExist:Boolean = false
-numberOfPlayers:Integer=0
-roomId
-roomILocationToIDMap:HashMap<Integer,
list<Integer>[0..3]>

+ getSpaceId(): Integer
+ getSpaceLocation(): Arraylist<Integer>[0..3]
+ getNeighnoringSpanceLocation():
ArrayList<ArrayList<Integer>[0..3]>
+getNumberOfNeighbors():Integer
+ifAnyNeighbours():Boolean
+room(roomId:Integer,
roomPostion:List<Integer>[0..3])

PetImpl
-name:string
-location:String

+PetImpl(name: String, location:String)
+getLocation():String
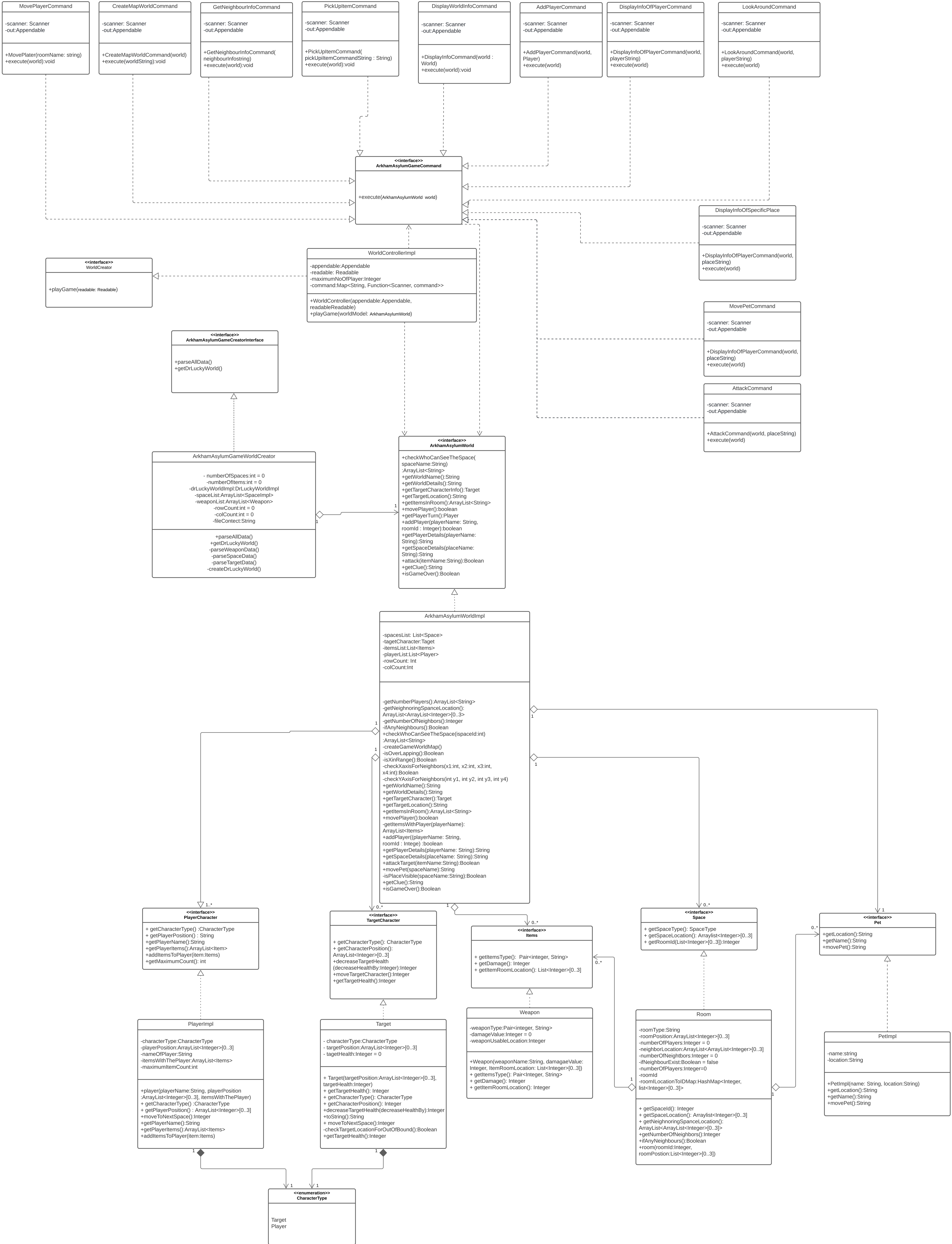+getName():String
+movePet():String

<<enumeration>>
CharacterType

Target
Player

PetImpl Test:

Note: The test cases of the Pet class can be broken down into 2 categories. First is the test of the constructor and second is the test of the functions of the PetImp class.

Constructor test
The constructor of the test class accepts 2 values:
String name: denotes the name of the pet.
String roomName: denotes the initial space location of the pet

Scenario	Function: PetImpl(String name, String roomName) Testing for ?	Input Value	Expected Value
Initialise the pet's name with an empty String. In this case the it should not accept the value and ask the user to input the name of the pet in correct format	New Pet(name, roomName)	Name = "" roomName = target.getRoomLocation()	**Expected**: IllegalArgumentException **Message**: Name should not be blank.
Initialise the pet's name with a valid string but the pet's roomName is not valid.	New Pet(name, roomName)	Name= "Fortune the Cat" roomName = ""	**Expected**: IllegalArgumentException **Message**: Room name should not be blank.
Initialise the pet's name with a valid String.	New Pet(name, roomName)	Name="Fortune the Cat" roomName = target.getRoomLocation()	Void Note: "As the expected values are correct the constructor won't throw any error"

PetImpl function: getLocation()
Note:
The constructor of the class is initialised with the following values:
name = **"**Fortune the Cat"

roomName=target.getRoomLocation()
The value of target.getRoomLocation = Armory

Scenario	Function: getLocation()	Input Value	Expected Value
Testing the initial location of the pet. Note: At first the initial location of the Target character is Armory as that's the index location of the first Room.	pet.getRoomLocation()	Void Note: No input parameter required for this function	"Armory"
Test pet's location after moving the pet to the neighbouring location. Note: In the starting of the game the pet is in the Armory, then the player do movePet("Kitchen") this will move the pet to the kitchen.	pet.getRoomLocation()	Void Note: No input parameter required for this function	"Kitchen"
Test pet's location after moving the pet to the invalid neighbouring location. Note: After moving the pet to the invalid space movePet("1234") function will throw an illegalArgumentException. Hence the pet will remain in the previous specified space that is Kitchen.	pet.getRoomLocation()	Void Note: No input parameter required for this function	"Kitchen" Note: "The location of the pet is not changed"

Function: movePet(String spaceLocation)

Note:

The function takes a string location. It validates if the location is valid or not and then sets the room location of the pet to the specified room.

In world implementation, when the user will input the command to move the pet to an invalid space. The function will check if the room is valid or not. If the room is not valid then the function throws an IllegalArgumentException error.

Scenario	Function: world.movePet(String room location)	Input Value	Expected Value
Move the pet to an invalid room location.	world.movePet(spaceLocation)	spaceLocation = "!@#$%"	**Expected**: IllegalArgumentException **Message**: The space name should be valid
Move the pet to an invalid room location.	world.movePet(spaceLocation)	spaceLocation = ""	**Expected**: IllegalArgumentException **Message**: The space name should be valid
Move the pet to the same room he is already in. Note: The current location of the pet is in Armory	world.movePet(spaceLocation)	spaceLocation = "Armory"	**Expected**: IllegalArgumentException **Message**: Pet is already in the location.
Move the pet to a valid location.	world.movePet(spaceLocation)	spaceLocation="Kitchen"	Void Note: As the location is valid it returns a void hence accepting the parameter

Function: movePetDFT()

Note:

The function moves the pet automatically from one space to another. The function uses depth first search algorithm to move the pet to the adjacent neighbour.

In the Arkham Asylum world, the space are declared in the following order:

1 Armory
2 Gaming Room
3 Gym
4 Dining Hall
5 Cell
6 Intense Treatment Room
7 Medical
8 Maze
9 Kitchen
10 Pool
11 Library
 12 Batman Cave
 13 Jailer Room
14 Watch Tower
15 Guard Room
16 Pantry
17 Servants' Quarters
18 Gordan Room
19 CCTV Room
20 Playing Area
21 Garden

After each player's turn the function will be called the pet will move the space which is in the next order from the list.

Scenario	Function: **world.movePetDFT( )**	Input Value	Expected Value
After each turn of the player or computer, if movePet is not explicitly called, world.moveDFT() will be called.

Suppose the starting location of the pet is Armory and the current player plays a turn to move to | world.movePetDFT() | Void

Note: Here no input is required. | Expected output of world.movePet():
Void

Note: We can confirm if the pet moved to the correct location by testing pet.getLocation:
"Gaming Room" |

neighbourhood, after the turn movePetDFT() will be called.			
After each turn of the player or computer, if movePet is not explicitly called, world.moveDFT() will be called. Suppose the starting location of the pet is "Gaming Room" and the current player plays a turn to move to neighbourhood, after the turn movePetDFT() will be called.	world.movePetDFT()	Void Note: Here no input is required.	Expected output of world.movePet(): Void Note: We can confirm if the pet moved to the correct location by testing pet.getLocation: "Gym"
Testing when the pet has reached the last room in the list, i.e Garden Now, the pet should go back to the first room in the list, i.e Armory	world.movePetDFT()	Void Note: Here no input is required.	Expected output of world.movePet(): Void Note: We can confirm if the pet moved to the correct location by testing pet.getLocation: "Armory"
If the makes a command to movePet and specify explicitly where the pet should move then this function should not be called.	world.movePet(String spaceName)	spaceName = "Kitchen"	Void Note: As the location is valid it returns a void hence accepting the parameter We can confirm if the pet moved to the correct location by testing pet.getLocation: "Kitchen"

Function: attackTarget(String itemName)

Note:

The function takes an item as an input. By default Poke will be available with the users to use.

If the item entered by the user is invalid then the function will throw an IllegalArgumentException error. If another player can see the player attacking from the neighbouring room then the function throws IllegalStateExcpeption

Scenario	Function: **world.attackTarget (String itemName)**	Input Value	Expected Value
Player attacks the target with an Invalid item. Note: Assume the player has following items with himself: "Item: Knife, Damage Value: 50", "Item: Gun, Damage value: 80" and the total health value of the Target is 100	world.attackTarget(String itemName)	itemName= "!@$"	**Expected**: IllegalArgumentException **Message**: The user does not have the specified item
Player attacks the target with an item which he has already used. Note: The player has already used knife, and the player has following items with himself: "Item: Gun, Damage value: 80" and the total health value of the Target is 100	world.attackTarget(String itemName)	itemName= "knife"	**Expected**: IllegalArgumentException **Message**: The player has already used the following item.
Player attacks the target without any	world.attackTarget(String itemName)	itemName= "poke"	Void

item. In this case the player can POKE the target, with damage value of 10 Note: The total health value of the Target is 100			Note: As the item is valid it returns a void hence accepting the parameter We can confirm if the health of the target decreased by using target,get Health() function: 90
Player attacks the target with a valid item. Note: Assume the player has following items with himself: "Item: Gun, Damage value: 80" and the total health value of the Target is 90	world.attackTarget(String itemName)	itemName= "gun"	Void Note: As the item is valid it returns a void hence accepting the parameter We can confirm if the health of the target decreased by using target,get Health() function: 10
Player attacks the target with a valid item but another player can see the player from the neighbouring room. Note: Assume the player has following items with himself: "Item: Knife, Damage Value: 50", "Item: Gun, Damage value: 80" and the total health value of the Target is 100	world.attackTarget(String itemName)	itemName= "gun"	**Expected**: IllegalStateException **Message**: Someone can see you while attacking. Note: We can check if the items is used by the player by using the following player.getItems: "Gun"

Function: isGameOver()

Note: After every turn the controller will check if the game is over or not. The game can only be over in 2 ways:
1. If the target is killed.
2. If the total number of turns gets over.

Scenario	Function: **world.attackTarget (String itemName)**	Input Value	Expected Value
Player attacks the target with a valid item. Note: Assume the player has following items with himself: "Item: Gun, Damage value: 80" and the total health value of the Target is 80	world.isGameOver()	Void Note: Here no input is required.	true
Player attacks the target with a valid item. Note: Assume the player has following items with himself: "Item: Gun, Damage value: 80" and the total health value of the Target is 90	world.isGameOver()	Void Note: Here no input is required.	false Note: The target health is 10 hence the game is not yet over.
Player makes a turn. Remaining turns: 0	world.isGameOver()	Void Note: Here no input is required.	True Note: As the remaining turn is over then the game is over.
Player makes a turn. Remaining turns: 2	world.isGameOver()	Void Note: Here no input is required.	false Note: As the remaining turn is not over then the game is not over.

Controller Test:

In the testing of the controller we need to test with mock models to check if our parameters are correctly passed or not. The mock models are also used to check if the function is being called correctly or not. We are going to use 2 mock model :

1. WorldMockModel: This mock model will be used to test all the valid test cases.

2. WorldInvalidMockModel: This mock model will be used to test all the invalid test cases.

The constructor of the mock model accepts 3 parameters:

1. Readable readable

2. StringBuffer log

3. Int uuid

We can use different mocks for failure cases when world or space objects are supposed to throw exceptions.

If the user has entered an invalid move then the user is asked to enter again the same command.

1. **AttackTarget**

Note: The sequence of the AttackTarget is as follows:

A. Prompt: "Enter Command"

B. User: "Attack target"

C. Prompt "Please enter the item to be used"

D. User: <item name>

E. Prompt: "Item used successfully" (if nobody sees the player attacking)

"Please enter a valid item" (if the item is not valid)

"Unable to attack as someone is watched you attack" (if someone watched)

After the turn if the game is over:

D. Prompt: "GAME OVER, <player name> won "

Scenario	Input	Expected Output (in out stream)
Player attacks with a valid Assume the player has the following items with himself: "Item: Knife, Damage Value: 50", "Item: Gun, Damage value: 80" and the total health value of	-> "Attack target" -> "Knife"	"Item used successfully" "Please enter a new command"

the Target is 100.		
Player attacks with an invalid Assume the player has the following items with himself: No items and the total health value of the Target is 100.	**->** "Attack target" -> "Knife"	"Please enter a valid item"
Player attacks without any item Assume the player has the following items with himself: No items and the total health value of the Target is 100.	**->** "Attack target" -> "poke"	**"Item used successfully"** "Please enter a new command"
Player attacks with a valid Assume the player has the following items with himself: "Item: Knife, Damage Value: 50", "Item: Gun, Damage value: 80" and the total health value of the Target is 50. The current player is Subhankar	**->** "Attack target" -> "Knife"	**"Item used successfully"** "GAME OVER, Subhankar won the game!"

2. **MovePetCommand**

   Note : The sequence of the MovePetCommand is as follows:
   A. Prompt: "Enter command"
   B. User: "move pet"
   C. Prompt: "Please enter a space"
   D. User: <space name>
   E. Prompt: "Pet moved to the space successfully" (if valid space)
              "Please enter valid space to move" (if invalid space)

Scenario	Input	Expected Output (in out stream)
Moving the pet from Kitchen to Armory, which is a valid space.	-> "Move Pet" -> "Armory"	"Pet moved to the space successfully " "Please enter a new command"
Moving the pet from Kitchen to Kitchen. Note: Since the pet is moved to the same room this is an invalid move	-> "Move Pet" -> "Kitchen"	"Please enter valid space to move" "Please enter a valid space"
Moving the pet from Kitchen to !@$%. The user has entered an invalid room hence this is an invalid move	-> "Move Pet" -> "!@$%"	"Please enter valid space to move" "Please enter a valid space"

3. playGame()

playGame(ArkhamAsylum WorldImpl model)	Input(World worldModel)	Expected Value
PlayGame if the model is null	null	IllegalArgumentException(" Null exception")

4. AddPlayerCommand

addPlayer(String playerName, Integer roomId)	Input (String playerName, Integer roomId)	Expected Value
Add a valid player name and valid roomId	("Add player","Subhankar", 1)	"Added player Subhankar in the room Kitchen"
Add a player in a room which has player Room: Kitchen, Player: {"Josh", "Tom"}	("Add player", "Subhankar", 1)	"Added player Subhankar in the room Kitchen"

Check if the controller is accepting command after adding a new valid player	("Add player", "Harry", 2)	"Added player Subhankar in the kitchen. Input:"
Check if the controller is accepting input by adding a player which has already been added. Player: {"Josh", "Tom"}	("Add player", "Josh", 1)	"Player already exists. Add player name and space id."
Check if the controller is accepting input by adding a player with invalid string value Player: {"Josh", "Tom"}	("Add player", "!@#$", 1)	"Player name not valid. Add player name and space id"
Check if the controller is accepting input by adding a player with invalid room id Player: {"Josh", "Tom"}	("Add player", "Harry". -1)	"Room id not valid. Add player name and space id"
Check if the controller is accepting input by adding a player with invalid room id and invalid player name Player: {"Josh", "Tom"}	("Add player", "!@#$". -1)	"Room id and player name not valid. Add player name and space id"

5. Move Player Command

movePlayerCommand(Integer roomId)	Input(String command, Integer roomId)	Expected Output
Move the current player to the neighbouring room. Player: {"Josh", "Tom"} Current Player: Name: Josh Room: Kitchen Neighbours:Bathroom(1), Dining Room (2)	("move", 2)	"Moved Josh to Dining Room Next turn Tom:"
Move the current player to the neighbouring room. Player: {"Tom", "Josh"} Current Player:	("move", 2)	"Moved Tom to Dining Room Next turn Josh:"

Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2)		
Check if the controller is accepting input after Moving the current player to the neighbouring room. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2)	("move", 1)	"Moved Tom to Dining Room Next turn Josh:"
Check if the controller is accepting input after Moving the current player to the invalid neighbouring room. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2)	("move", -1)	"No neighbours found Next turn Tom:"
Check if the controller is accepting input after Moving the current player to the room which is not a neighbour. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2)	("move", 10)	"No neighbours found Next turn Tom:"
Check if the controller is accepting input after Moving the current player to the room after the game is over. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1),	("move", 1)	"Game is over. Josh killed the Joker! Input command:"

Dining Room (2)		

6. LookAroundCommand

LookAroundCommand()	Input (String command)	Expected Output
Look Around for the current player. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items: Gun, Stick	("Look Around")	"Room: Kitchen Neighbours:Bathroom(1), Dining Room (2). Items in Room: "Gun". "Stick" Next turn Josh: "
Check if the controller is still working after looking around the command. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items: Gun, Stick	("Look Around")	"Room: Kitchen Neighbours:Bathroom(1), Dining Room (2). Items in Room: "Gun". "Stick" Next turn Josh:"
Check if the controller is still working after looking around the command. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items: Gun, Stick	("Look Around")	"Room: Kitchen Neighbours:Bathroom(1), Dining Room (2). Items in Room: "Gun". "Stick" Next turn Josh:"
Check if the controller is still working after looking around the command when the game is over. Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2)	("Look Around")	"Game is over. Josh killed the Joker! Input command:"

Items: Gun, Stick		

7. PickUpItemCommand

PickUpItemCommand()	Input (String command, String itemName)	Expected Output
Pick up items by the current player. Current Player: Tom Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items: Gun, Stick	("Pick Up Item", "Gun")	Gun Picked by Tom. Next turn Josh
Check if the controller is working after picking up an already picked item by the current player. Current Player: Tom Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items: Gun, Stick	("Pick Up Item", "Gun")	Gun is already been picked. Next turn Tom
Check if the controller is working after picking up an already picked item by the current player. Current Player: Tom Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items: Gun, Stick	("Pick Up Item", "Gun")	Gun is already been picked. Next turn Tom

	("Pick Up Item", "Gun")	"Gun is no longer available
Check if the controller is working after picking up an item which has already been used by the current player. Current Player: Tom Player: {"Tom", "Josh"} Current Player: Name: Tom Room: Kitchen Neighbours:Bathroom(1), Dining Room (2) Items:Stick		Next turn Tom"

8. **DisplayInfoOfPlayerCommand**

DisplayInfoOfPlayerCommand()	Input(String command, String playerName)	Expected Output
Display info about a valid specific player. Current Player: Tom Player: {"Tom", "Josh"}	("Display player info", "Josh")	"Player: Josh, Room: Kitchen. Items: Gun, Sticker. Joker Location: Drawing Room Next turn Tom"
Check if the controller is working after displaying info about a valid specific player after the game is over. Current Player: Tom Player: {"Tom", "Josh"}	("Display player info", "Josh")	"Player: Josh, Room: Kitchen. Items: Gun, Sticker. Joker is dead Game is over. Start new game?:"
Display info about an invalid player. Current Player: Tom Player: {"Tom", "Josh"}	("Display player info", "Curry")	"Player Not Found. Next Turn: Tom"
Display info about an invalid string name.	("Display player info", "!@#$%")	"Player Not Found. Next Turn: Tom"

Current Player: Tom Player: {"Tom", "Josh"}		

9. **DisplayInfoOfSpecificPlace Command:**

DisplayInfoOfSpecificPlace()	Input(String command, String placeName)	Expected Output
Display info about a valid specific place. Current Player: Tom Place: {"Kitchen", "Bathroom"}	("Display place info", "Bathroom")	"Player in the room: "Josh, Tom, Subhankar. Items in the room: Knife, Ball, Bat" Next turn Tom"
Display info about a valid specific place after picking up an item from the place. Picked Bat. Current Player: Tom Place: {"Kitchen", "Bathroom"}	("Display place info", "Bathroom")	"Player in the room: "Josh, Tom, Subhankar. Items in the room: Knife, Ball" Next turn Tom"
Display info about a valid specific place where the target is present. Current Player: Tom Place: {"Kitchen", "Bathroom"}	("Display place info", "Bathroom")	"Player in the room: "Josh, Joker. Items in the room: Knife, Ball, Bat" Next turn Tom"
Check if the controller is taking input after display info about a valid specific place after the game is over Current Player: Tom Place: {"Kitchen", "Bathroom"}	("Display place info", "Bathroom")	"Player in the room: "Josh, Tom, Subhankar. Items in the room: Knife, Ball, Bat" The game is over. Tom killed Joker:"
Check if the controller is taking input after display info about a valid specific place Current Player: Tom	("Display place info", "Bathroom")	"Player in the room: "Josh, Tom, Subhankar. Items in the room: Knife, Ball, Bat"

Place: {"Kitchen", "Bathroom"}		Next turn Tom:"

10. AddComputerPlayer Command

AddComputerPlayer	Input (String command, String playerName)	Expected Output
Check if the controller is accepting input after adding a computer player before the game is starting	"Add computer player, Android"	"Android added as a computer player. Add another player."
Check if the controller is accepting input after adding a computer player in the middle of the game	"Add computer player, Android"	"Cannot add a player in the middle of the game. Next turn Tom:"
Check if the controller is accepting input after adding a computer player in the game is over.	"Add computer player, Android"	"Cannot add a player after the game is over". Game is over. Tom killed Joker: Start a new game:"

11. Restrict The number of Turn Command

restrictTurn()	Input	Expected
Check if the controller is accepting input after the maximum number of turn reached	("Move", 2)	"Maximum number of turns reached. Next turn: Josh"

Test cases for Player.java class

```
public Player(ArrayList<Strings>[0..3] itemArrayList, String playerName, Integer
playerPosition) {
  //...
}
```

Testing construction	Inputs	Expected Value
Player with the valid given name items List and position. Rooms: Kitchen(1), Bathroom(2), Dining Room (3)	(("Gun", "Knife"), "Josh", 1)	"Player: Josh, Room: Kitchen. Items: Gun, Sticker."
Player with the valid given name items List and same position. Rooms: Kitchen(1) ,Bathroom(2), Dining Room (3)	(("RPG", "Stick"), "Tom", 1)	"Player: Tom, Room: Kitchen. Items: RPG, Stick."
Player with the empty items List and position. Rooms: Kitchen(1) ,Bathroom(2), Dining Room (3)	((""), "Tom", 1)	"Player: Tom, Room: Kitchen. Items:"
Player with the same given name items List and position. Rooms: Kitchen(1), Bathroom(2), Dining Room (3)	(("Gun", "Knife"), "Josh", 1)	IllegalArgumentException(" Player already exists")
Player with the given name items List and invalid position. Rooms: Kitchen(1), Bathroom(2), Dining Room (3)	(("Gun", "Knife"), "Josh", -1)	IllegalArgumentException(" Location does not exists.")
Player with the given name items list and valid position, but invalid Name. Rooms: Kitchen(1), Bathroom(2), Dining Room (3)	(("Gun", "Knife"), "!@#$", -1)	IllegalArgumentException(" Player name does not exists.")

Test addPlayer(String nameOfPlayer, Integer roomId)

Testing addPlayer(String playerName, Integer roomId)	Inputs (String nameOfPlayer, Integer roomId)	Expected Value
For a valid player name and valid room number.	("Subhankar", 0)	true
For a valid player name and the same room number.	("Tom", 0)	true

For a valid player name and the different room number.	("Cat", 1)	true
For the same player name and the different room number.	("Cat", 1)	IllegalArgumentException(" Player is already added")
For a valid player name and the different room number.	("Cat", 1)	IllegalArgumentException(" Player is already added")
For an invalid string character	("!@#$", 1)	IllegalArgumentException(" Player name should valid")
For an integer value in the name of the player	(123, 1)	IllegalArgumentException(" Player name should valid")
For an empty value in the name of the player	("", 1)	IllegalArgumentException(" Player name should valid")
For invalid room id but valid name	("Subhankar", -1)	IllegalArgumentException(" Room id should be value")
For room which does not exists but valid string	("Subhankar", 100)	IllegalArgumentException(" Room id should be value")

Test addComputerPlayer()

Testing addComputerPlayer()	Input	Expected Value
Invokes the function	void	No.of player = 1(increased by 1)

Test movePlayer(Integer roomId)

Testing movePlayer(Integer roomId)	Input (Integer roomId)	Expected Value
Move to valid space id	(2)	true
Move to valid space id	(3)	true
Move from last space id to first space id. If the player is 20.	(1)	true
Move to invalid space id	(-100)	IllegalArgumentException("

		Space id should be valid")
Move to space id which is out of bound	(100)	IllegalArgumentException(" Space id should be valid")
Move to space that is not the neighbour. If the player is in room 1.	(20)	IllegalArgumentException(" Cannot move to non neighbours")
Move to space id in which the player is already in	(2)	IllegalArgumentException(" Player is already in the space")

Test pickItem()

Testing pickItem(String itemName)	Input (String itemName)	Expected Value
Pick an item in the current space	("Gun")	true
Pic an item in the room which has multiple rooms	("Broom")	true
Pick an item in the room. Before reaching the maximum limit	("stick")	true
Pick an item in the room after reaching maximum value	("stone")	IllegalArgumentException(" Maximum number of items reached")
Pick an item which is in the other room	("Knife")	IllegalArgumentException(" Item should be in the same player")
Pick an item which is already picked	("Gun")	IllegalArgumentException(" The item is already picked.")
Pick an item which is not available	("Bazzuca")	IllegalArgumentException(" The item is not available".)

Test displayDescription(String playerName)

Testing displayDescription(String	Input (String playerName)	Expected Value

playerName)		
Display for a valid player with 2 items	("Subhankar")	"Location: Kitchen, Items: Knife, Spoon"
Display for a valid player with 0 items	("Subhankar")	"Location: Dining Room, items:"
Display for a valid player with maximum items	("Subhankar")	"Location: Dining Room, Items Knife, spoon, rock, gun"
Display for an invalid player String	("!@#$")	IllegalArgumentException(" Player name should be valid".)
Display for a computer controlled player	("Josh")	"Location: Dining Room, Items Knife, spoon, rock, gun"

Test lookAround("String playerName")

Test LookAround(String playerName)	Input (String playerName)	Expected Value
Look around for a valid player with 1 neighbour.	("Josh")	"Location: Dining Room"
Look around for a valid player with 2 neighbours.	("Tom")	"Location: Dining Room, Kitchen"
Look around for a valid player with 0 neighbours.	("Cat")	""
Look around for an invalid player	("!@#$%")	IllegalArgumentException(" Player name should be valid".)

Test getTurn()

getTurn()	Input	Expected Value
Get turn for 5 players in the list ("Tom", "Josh", "Rick",	getTurn()	"Tom"

"Call")		
Get turn for 4 players in the list ("Josh", "Rick", "Call","Tom")	getTurn()	"Josh"
Get turn for 1 players ("Josh")	getTurn()	"Josh"
Get turn for 0 players	getTurn()	"No players added yet"

Test cases for Target.java class

Constructor of the StevensonReading class:

```
public Target(ArrayList<Integer>[0..3] targetPosition, Integer targetHealth) {
//...
}
```

Testing construction	Inputs	Expected Value
Target with a given location and health	((22, 19, 23, 26), 100)	"Target location(30,20,5,15)"
Target with a given location and health	((16 21 21 28),80)	"Target Location(16,21,21,28)"
Target with a given location and negative health	((16 21 21 28), -10)	IllegalArgumentException
Target with a given negative location and negative health	((-30,20,5,15), 100)	IllegalArgumentException
Target with a given negative location and negative health	((-30,-20,-5,-15), 100)	IllegalArgumentException
Target with a given location and health as zero	((22, 19, 23, 26), 0)	IllegalArgumentException
Target with an Invalid given location and valid health	((22, 19, 23, Integer.MAX), 100)	IllegalArgumentException
Target with given location and invalid health	((22, 19, 23, 26), Integer.MAX)	IllegalArgumentException

Target with given location and given health more than total weapon damage	((22, 19, 23, 26), 1000)	IllegalArgumentException
Target with given location which is out of bound of board and given health more than total weapon damage	((330,30,15,0), 10)	IllegalArgumentException

Testing getCharacterType()	Inputs	Expected Value
For valid position and health		CharacterType.Target

Testing getCharacterHealth()	Inputs	Expected Value
For valid position and health	((330,30,15,0), 10)	10
For valid position and health	((22, 19, 23, 26), 100)	100

Testing getCharacterPosition()	Inputs	Expected Value
For valid position and health	((22, 19, 23, 26), 100)	1
After Moving the character to next highest int	((16,21,21,28), 100)	2

Testing getCharacterPosition()	Inputs	Expected Value
For valid position and health	((22, 19, 23, 26), 100)	1
After Moving the character to next highest int	((16,21,21,28), 100)	2

Testing moveToNexSpace()	Inputs	Expected Value

Move from current room to next subsequent highest int	1	2
Move from current room to next subsequent highest int	50	51
If at the highest space number	100	100

Testing decreaseTargetHealth()	Inputs	Expected Value
Decrease target health from current health 100	10	90
Decrease target health from current health 1000	900	100
Decrease target health from current health 100	100	0
Decrease target health from current health 100	200	0

Test class for Weapons.java class

```
public Weapon(String weaponName, Integer damage Value, List<Integer>[0..3]
ItemRoomLocation) {
//...
}
```

Testing construction	Inputs	Expected Value

Weapon with valid damage value and location	("Knife", 20, (22, 19, 23, 26))	"Knife, 20, 1"
Weapon with valid big damage value and location	("Fork", 100, (22, 19, 23, 26))	"Fork, 100, 1"
Weapon with invalid damage value	("Screw", -100, (22, 19, 23, 26))	IllegalArgumentException
Weapon with invalid room location	("Spoon", 100, (-22, 19, 23, 26))	IllegalArgumentException
Weapon with invalid room location	("Broom", 100, (22, -19, 23, 26))	IllegalArgumentException
Weapon with invalid room location	("Lemon", 100, (22, 19, -23, 26))	IllegalArgumentException
Weapon with invalid room location	("Gun", 100, (22, 19, 23, -26))	IllegalArgumentException
Weapon with invalid room location coordinates	("Axe", 100, (10000, 19, 23, 26))	IllegalArgumentException

Weapon with invalid room location coordinates	("Blade", 100, (10, 1900, 23, 26))	IllegalArgumentException
Weapon with invalid room location coordinates	("Sword", 100, (10, 19, 2300, 26))	IllegalArgumentException
Weapon with invalid room location coordinates	("Machine Gun", 100, (10, 19, 23, 2600))	IllegalArgumentException

Testing getItemType()	Inputs	Expected Value
Weapons with given name and location coordinates	("Fork", 100, (10, 19, 2300, 26))	("Fork", 1)
Weapons with given name and location coordinates	("Knife", 10, (22, 19, 23, 26))	("Fork, 2)

Testing getDamageValue()	Inputs	Expected Value
Damage value when values are given in constructor	("Fork", 100, (10, 19, 2300, 26))	100

Damage value when values are given in constructor	("Knife", 10, (22, 19, 23, 26))	10

Testing getItemRoomLocation()	Inputs	Expected Value
Get room item id for valid item and valid location	("Fork", 100, (10, 19, 2300, 26))	1
Get room item id for valid item and valid location	("Knife", 10, (22, 19, 23, 26))	2

Test Room

```
public Room(Integer roomId, List<Integer> roomPosition ) {
//...
}
```

Testing construction	Inputs	Expected Value
Room with valid with position id and room position	(1, (22, 19, 23, 26))	"Room location(22, 19, 23, 26)"
Room with valid with position id and room position	(2, (24, 20, 24, 30))	"Room Location(24, 20, 24, 30)"
Room with invalid with position id and room position	(3, (-24, 20, 24, 30))	IllegalArgumentException
Room with invalid with position id and room position	(3, (24, -20, 24, 30))	IllegalArgumentException
Room with invalid with position id and room position	(3, (24, 20, -24, 30))	IllegalArgumentException
Room with invalid with position id and room position	(3, (24, 20, 24, -30))	IllegalArgumentException
Room with invalid with position id and room position	(Integer.MAX, (24, 20, 24, 30))	IllegalArgumentException

Testing getSpaceId()	Inputs	Expected Value

Room with valid with position id and room position	(2, (24, 20, 24, 30))	2
Room with valid with position id and room position	(3, (1, 5, 24, 30))	3

Testing getSpaceLocation()	Inputs	Expected Value
Room with valid with position id and room position	(2, (24, 20, 24, 30))	(24, 20, 24, 30)
Room with valid with position id and room position	(3, (1, 5, 24, 30))	(1, 5, 24, 30

Testing getNeighnoringSpanceLocation()	Inputs	Expected Value
Room with valid with position id and room position	(2, (24, 20, 24, 30))	(24, 20, 24, 30) , (1, 5, 24, 30)
Room with valid with position id and room position	(3, (1, 5, 24, 30))	(1, 5, 12, 32), (24, 20, 24, 30)

Testing getNeighnoringSpanceLocation()	Inputs	Expected Value
Room with valid with position id and room position	(2, (24, 20, 24, 30))	(24, 20, 24, 30) , (1, 5, 24, 30)
Room with valid with position id and room position	(3, (1, 5, 24, 30))	(1, 5, 12, 32), (24, 20, 24, 30)

Testing ifAnyNeighbours()	Inputs	Expected Value
Room with valid with position id and room position	(2, (24, 20, 24, 30))	True

Room with valid with position id and room position	(3, (1, 5, 24, 30))	False

Test ArkhamAsylumWorld

```
public ArkhamAsylumWorld( List<Space> spaceList, Target targetCharacter, List<Items>
itemList) {
//...
}
```

Testing construction	Inputs	Expected Value
Room with valid Space location, Target character, and item List	((1, (22, 19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	"DrLuckyWorld((1, (22, 19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22)))))"
Room with valid Space location, Target character, and item List	((2, (22, 19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	"DrLuckyWorld((2, (22, 19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22)))))"
Room with Invalid Space location, Target character, and item List	((-2, (22, 19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException
Room with Invalid Space location, Target character, and item List	((2, (-22, 19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException
Room with Invalid Space location, Target character, and item List	((2, (22, -19, 23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException
Room with Invalid Space location, Target character, and item List	((2, (22, 19, -23, 26),), (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException
Room with Invalid Space location, Target character, and item List	((2, (22, 19, -23, 26)) (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException
Room with Invalid Space location, Target character, and item List	((2, (22, 19, 23, -26)) (2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException
Room with Invalid Space location, Target character, and item List	((2, (22, 19, 23,6)) (-2, (24, 20, 24, 30)), (3, (21, 22, 40, 22))))	IllegalArgumentException

Room with Invalid Space location, Target character, and item List	((2, (22, 19, 23,6)) (2, (24, 20, 24, 30)), (-3, (21, 22, 40, 22))))	IllegalArgumentException