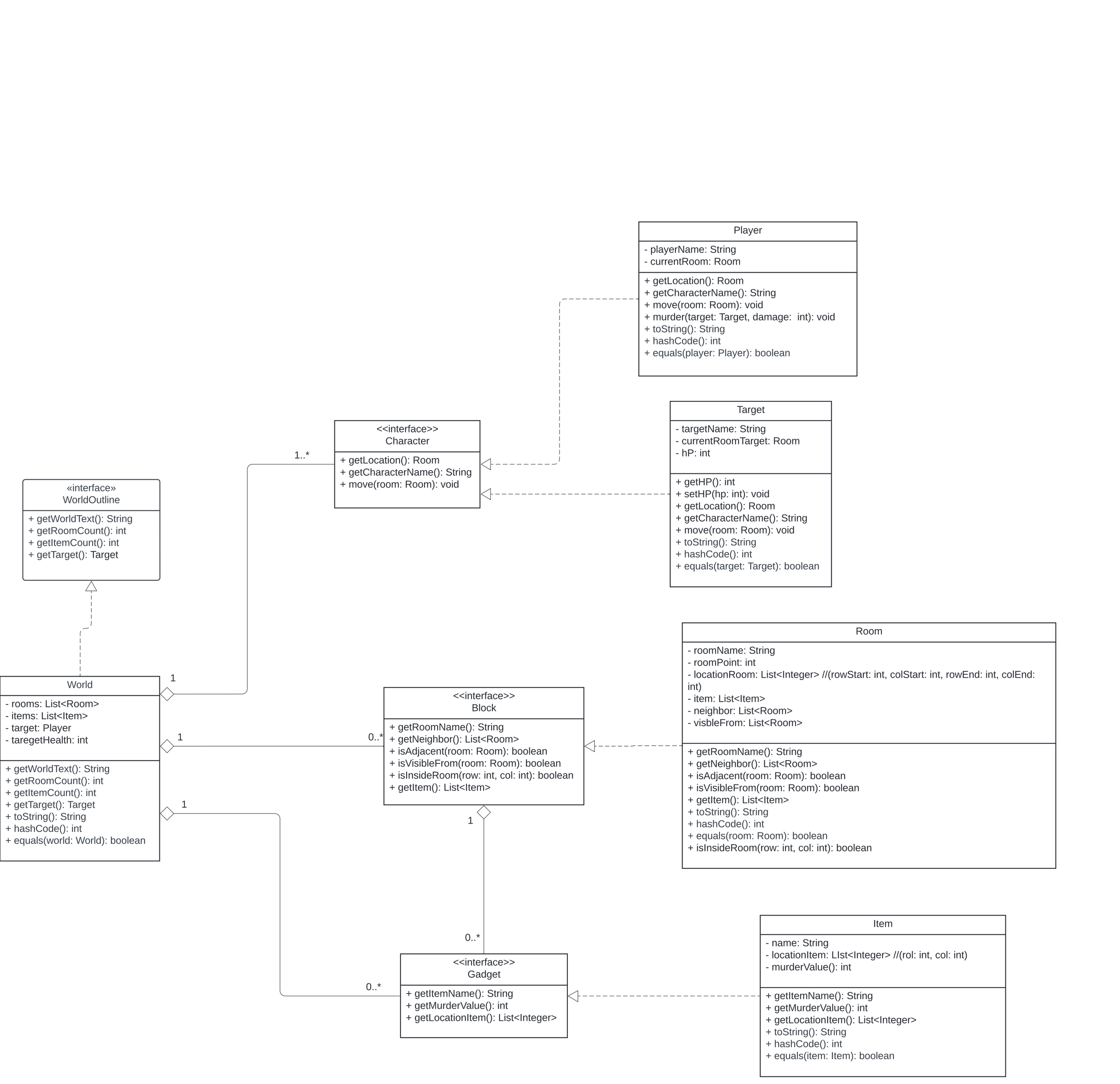
Player
- playerName: String
- currentRoom: Room
+ getLocation(): Room
+ getCharacterName(): String
+ move(room: Room): void
+ murder(target: Target, damage: int): void
+ toString(): String
+ hashCode(): int
+ equals(player: Player): boolean
Target
- targetName: String
- currentRoomTarget: Room
- hP: int
+ getHP(): int
+ setHP(hp: int): void
+ getLocation(): Room
+ getCharacterName(): String
+ move(room: Room): void
+ toString(): String
+ hashCode(): int
+ equals(target: Target): boolean
<<interface>>
Character
+ getLocation(): Room
+ getCharacterName(): String
+ move(room: Room): void
1..*
«interface»
WorldOutline
+ getWorldText(): String
+ getRoomCount(): int
+ getItemCount(): int
+ getTarget(): Target
World
- rooms: List<Room>
- items: List<Item>
- target: Player
- taregetHealth: int
+ getWorldText(): String
+ getRoomCount(): int
+ getItemCount(): int
+ getTarget(): Target
+ toString(): String
+ hashCode(): int
+ equals(world: World): boolean
1
1
1
0..*
<<interface>>
Block
+ getRoomName(): String
+ getNeighbor(): List<Room>
+ isAdjacent(room: Room): boolean
+ isVisibleFrom(room: Room): boolean
+ isInsideRoom(row: int, col: int): boolean
+ getItem(): List<Item>
Room
- roomName: String
- roomPoint: int
- locationRoom: List<Integer> //(rowStart: int, colStart: int, rowEnd: int, colEnd: int)
- item: List<Item>
- neighbor: List<Room>
- visbleFrom: List<Room>
+ getRoomName(): String
+ getNeighbor(): List<Room>
+ isAdjacent(room: Room): boolean
+ isVisibleFrom(room: Room): boolean
+ getItem(): List<Item>
+ toString(): String
+ hashCode(): int
+ equals(room: Room): boolean
+ isInsideRoom(row: int, col: int): boolean
1
0..*
<<interface>>
Gadget
+ getItemName(): String
+ getMurderValue(): int
+ getLocationItem(): List<Integer>
0..*
Item
- name: String
- locationItem: LIst<Integer> //(rol: int, col: int)
- murderValue(): int
+ getItemName(): String
+ getMurderValue(): int
+ getLocationItem(): List<Integer>
+ toString(): String
+ hashCode(): int
+ equals(item: Item): boolean

Test Plan	Test Input	Expect Output
Class Item		
Test Item Constructor	*coordList is the List<Integer> only contain 2 values	
create an Item object with correct parameters	Item("Crepe Pan", coordList, 3)	Player object created
create an Item object with incorrect parameters (wrong types of parameter or wrong numbers of parameters)	Item("Crepe Pan", 8, 3, 3) or Item("Crepe Pan", "8", 3)	throws IllegalArgumentException
create an Item object with coordList size not equal to 2	Item("Crepe Pan", coordListSizeOne, 3)	throws IllegalArgumentException
Test getItemName(): String		
It should return the name of the item as String	nothing, as long as our object is created properly (Item("Crepe Pan", coordList, 3))	Crepe Pan
Test getMurderValue(): int		
It should return the value of the item as an interger number	nothing, as long as our object is created properly (Item("Crepe Pan", coordList, 3))	3
Test getLocation(): List<int>		
It should return the xy coordinate as an int array with only 2 elements	nothing, as long as our object is created properly (Item("Crepe Pan", coordList, 3))	coordList, the first element is 8 and second element is 3
Class Room		
Test Room Constructor	*locationList is List<Integer> with 4 elements, ItemList is List<Item>, neighborList and visibleList are List<Room>	
create a room object with correct parameters	room("Armory", 15, locationList, ItemList, neighborList, visibleList)	Room object created
create a room object with incorrect parameters (wrong types of parameter or wrong numbers of parameters)	room("Armory", 15, 1,2,3,4, ItemList, neighborList, visibleList) or room("Armory", 15, locationList, ItemList, "Piazza", visibleList)	throws IllegalArgumentException
create a room object with incorrect size of locationList (the list size is not 4)	room("Armory", 15, locationListSizeFive, ItemList, neighborList, visibleList)	throws IllegalArgumentException
Test getRoomName(): String		
It should return the name of the room as String	nothing, as long as our object is created properly room("Armory", 15, locationList, ItemList, neighborList, visibleList)	Armory
Test getNeighbor(): List<Room>		
It should return the List of the Room objects	nothing, as long as our object is created properly room("Armory", 15, locationList, ItemList, neighborList, visibleList)	List<Room>
Test getItem(): List<Item>		
It should return the List of the Item objects	nothing, as long as our object is created properly room("Armory", 15, locationList, ItemList, neighborList, visibleList)	List<Item>
Test isAdjacent(room: Room): boolean		
it should check is the parameter an element of the field neighbor, if it is, return True.	Piazza	False
Pass incorrect type of parameter is not allowed or not only one input parameter	"Piazza" or (empty)	throws IllegalArgumentException
Test isVisibleFrom(room: Room): boolean		
it should check is the parameter an element of the field visbleFrom, if it is, return True.	Piazza	True
Pass incorrect type of parameter is not allowed or not only one input parameter	"Piazza" or (empty)	throws IllegalArgumentException
Test isInsideRoom(row: int, col: int): boolean		
It should check is the coordinate(2 integers) inside this room or not	0,0	False

Pass incorrect numbers of parameter(not 2) or incorrect types of parameter(not int) is not allowed	0,0,0 or "0","0"	throws IllegalArgumentException
Class Player		
Test Player Constructor		
create a Player object with correct parameters	Player("Eric", Piazza)	Player object created
create a Player object with incorrect parameters (wrong types of parameter or wrong numbers of parameters)	Player("Eric", "Piazza")	throws IllegalArgumentException
Test getCharacterName(): String		
It should return the name of the Player as String	nothing, as long as our object is created properly (Player("Eric", Piazza))	Eric
Test getLocation(): Room		
It should return the Room object the Player object locate at	nothing, as long as our object is created properly (Player("Eric", Piazza))	Room object Piazza
Test move(room Room):void		
It should move the player to the destinated room	Foyer	Nothing should return, but can check with the getLocation to show Room object Foyer
Pass incorrect numbers of parameter(not 1) or incorrect types of parameter(not Room) is not allowed	"Foyer" or (empty)	throws IllegalArgumentException
Pass the room is not adjacent to the room the player locates at is not allowed	Library	throws IllegalArgumentException
Test murder(target Target):void		
It should try to kill the target by reducing target's health point by certain amount	Doctor Lucky, 5	Nothing should return, but it should reduce target's hp by the item's murderValue by using setTargetHP(), can check with getTargetHP()
Pass incorrect numbers of parameter(not 2) or incorrect types of parameter(not Target) is not allowed	Doctor Lucky, 5, 5 or Doctor Lucky, "5"	throws IllegalArgumentException
Pass the damage number below 1 is not allowed	Doctor Luckly, 0	throws IllegalArgumentException
Class Target		
Test Target Constructor		
create a Target object with correct parameters	Target("Doctor Luckly", Piazza, 10)	Target object created
create a Target object with incorrect parameters (wrong types of parameter or wrong numbers of parameters)	Target("Doctor Luckly", Piazza)	throws IllegalArgumentException
create a Target object with non-positive hp to start with (hP < 1)	Target("Doctor Luckly", Piazza, 0)	throws IllegalArgumentException
Test getCharacterName(): String		
It should return the name of the Target as String	nothing, as long as our object is created properly (Target("Doctor Luckly", Piazza, 10))	Doctor Luckly
Test getLocation(): Room		
It should return the Room object the Target object locate at	nothing, as long as our object is created properly (Target("Doctor Luckly", Piazza, 10))	Piazza
Test move(room Room):void		
It should move the target to the destinated room	Foyer	Nothing should return, but can check with the getLocation to show Room object Foyer
Pass incorrect numbers of parameter(not 1) or incorrect types of parameter(not Room) is not allowed	"Foyer" or (empty)	throws IllegalArgumentException
Pass the room is not adjacent to the room the player locates at is not allowed	Library	throws IllegalArgumentException

Test getHP(): int		
It should return the number as an integer	nothing, as long as our object is created properly (Target("Doctor Luckly", Piazza, 10))	10
Test setHP(hp: int): void		
It should set the target hp to desired amount	20	nothing will return, but the hp should set to 20, we can check it with getHP
Pass incorrect numbers of parameter(not 1) or incorrect types of parameter(not int) is not allowed	15, 20 or "20"	throws IllegalArgumentException
Pass the hp is below 1 is not allowed	0	throws IllegalArgumentException
Class World		
Test World Constructor		
correct input file	the valid txt file (mansion.txt)	contructor parses
invalid input file	some other type file, or incorrect txt file (mansion.jpg)	throws IllegalArgumentException
invalid input path	wrong file path	throws FileNotFoundException
Test getWorldText()		
return our world structure in the txt file as String	none, as long the txt file is valid for our constructor	return everything inside our txt file as string
Test getRoomCount()		
numbers of room in valid txt file is the same as the the number after the target in txt file	none, as long the txt file is valid for our constructor	return the correctly total number of room in the structure (21)
numbers of room in valid txt file is the different as the number after the target in txt file	none, as long the txt file is valid with unmatchable information for our constructor (we have 21 lines of rooms but the number after target is 20)	throws IllegalArgumentException
Test getItemCount()		
numbers of room in valid txt file is the same as the the number after the last room in txt file	none, as long the txt file is valid for our constructor	return the correctly total number of items in the structure (20)
numbers of room in valid txt file is the different as the the number after the last room in txt file	none, as long the txt file is valid for our constructor with unmatchable information (we have 20 lines of items but the number after target is 21)	throws IllegalArgumentException
numbers of room should be greater than 1	none, as long the txt file is valid for our constructor with incorrect information (we only have 1 or less room, and the room count is 1 or less)	throws IllegalArgumentException
Test getTarget()		
should return the correct target name and the target should be Doctor Lucky	none, as long the txt file is valid for our constructor	Doctor Lucky
anything for the target except String Doctor Lucky is not allowed	none, as long the txt file is valid for our constructor with incorrect information (something like 50 Lucky Doctor)	throws IllegalArgumentException