MurderTargetCommand

- World world
- Scanner scanner
- int playerId

+ excute(Appendable output): void

MovePetCommand

- World world
- Scanner scanner
- int playerId

+ excute(Appendable output): void

LookAroundCommand

- World world
- Scanner scanner

+ excute(Appendable output): void

CreatePlayerCommand

- World world
- Scanner scanner
- List<Integer> playerIds
- Map<Integer, Boolean> isComputer
- Map<Integer, String> playerNames

+ excute(Appendable output): void

TargetInfoCommand

- World world

+ excute(Appendable output): void

SaveWorldMapCommand

- World world

+ excute(Appendable output): void

PlayerInfoCommand

- World world
- Scanner scanner
- int playerId

+ excute(Appendable output): void

DisplayRoomInfoCommand

- World world
- Scanner scanner

+ excute(Appendable output): void

MovePlayerCommand

- World world
- Scanner scanner
- int playerId

+ excute(Appendable output): void

PickUpItemCommand

- World world
- Scanner scanner
- int playerId

+ excute(Appendable output): void

CreateComputerPlayerCommand

- World world
- Scanner scanner
- List<Integer> playerIds
- Map<Integer, Boolean> isComputer
- Map<Integer, String> playerNames

+ excute(Appendable output): void

<<interface>>
Command

+ excute(Appendable output): void

GameController

- scanner: Scanner
- output: Appendable
- maxTurns: int
- currentTurn: int
- gameStarted: boolean
- playerIds: List<Integer>
- currentPlayerIndex: int
- isComputer: Map<Integer, Boolean>
- rng: RandomUmberGenerator
- playerNames: Map<Integer, String>
- isRunning: boolean

+ playGame(WorldOutline world): void
+ getIsRunning(): boolean
- setRunning(boolean isRunningInput): void
- setupGame(WorldOutline world): void
- runGame(WorldOutline world): void
- processPlayerInput(String input, WorldOutline world, nt playerId): void
- computerPlayerActions(WorldOutline world, int playerId): void
- pickUpItem(WorldOutline world, int playerId): void
- advanceTurn(WorldOutline world): void
- saveWorldMap(WorldOutline world): void
- print(String message): void
- displayMenu(): void
- getTargetInfo(): void
- murderAttempt(): void
- movePet(): void
- lookAround(): void

MockWorldOutline

- roomCount = 3
- mockId = 0
- mockTargetHealthPoint = 0
- mockMurderAttempt = false

+ Mock Class with the same thing in World Class

FakeRandomNumberGenerator

- nextIntResult: int

+ setNextIntResult(int result): void
+ nextInt(int bound): int

ControllerDriver

+ main(String[] args): void

RandomNumberGenerator

- random: Random
- fixedOutputs: Queue<Integer>

+ nextInt(int bound): int

<<interface>>
Controller

+ playGame(WorldOutline world): void
+ getIsRunning(): boolean

Controller

Model

«interface»
WorldOutline

+ getWorldText(): String
+ getRoomCount(): int
+ getItemCount(): int
+ drawWorld(): void
+ setWorldText(): void
+ getRoomData(): List<String[]>
+ getItemData(): List<String[]>
+ moveTargetToRoom(String roomName): String
+ moveTargetToNextRoom(): String
+ getRoomOccupants(Block room): String
+ setItemLimit(int newItemLimit): void
+ displayRoomInfo(String roomName): String
+ callCreateRoom(String roomName, int roomId, int[] coordinates, List<String[]> allRoomData): String
+ callCreateTarget(String name, Block room, int health): String
+ callCreateItem(String name, int location, int murderValue): String
+ callCreatePlayer(String playerName, int startRoomIndex): int
+ getTargetInfo(): String
+ getPlayerInfo(int playerId): String
+ setMaxTurn(int maxTurn): void
+ getMaxTurn(): int
+ movePlayer(int playerId, int roomId): String
+ playerPickUpItem(int playerId, String itemName): String
+ playerLookAround(int playerId): String
+ getRoomItems(int roomId): List<String>
+ getPlayerItems(int playerId): List<String>
+ getPlayerRoomId(int playerId): int
+ getNeighborRooms(int roomId): List<Integer>
+ getPetInfo(): String
+ playerMovePet(int playerId, int roomId): String
+ murderAttempt(int playerId): String
+ canPlayerBeSeenByAny(int playerId): boolean
+ initializePetDFS(): void
+ movePetToNextRoom(): String
+ movePet(int playerId, int targetRoomId): String
+ canInteractWithPet(int playerId): boolean
+ getTargetHealthPoint(): int
+ getPlayerItems(int playerId): List<String>
+ usePlayerItem(int playerId, String itemName): void
+ removePet(): void
+ getPlayerLocation(int playerId): String
+ getPlayerItemsInfo(int playerId): String

<<interface>>
CharacterTarget

+ getLocation(): Block
+ getCharacterName(): String
+ move(Block room): void
+ getCharacterInfo(): String
+ gethealthPoint(): int
+ sethealthPoint(hp: int): void

Target

- healthPoint: int
- currentRoom: Block
- name: String

+ gethealthPoint(): int
+ sethealthPoint(hp: int): void
+ getCharacterInfo(): String
+ getLocation(): Block
+ getCharacterName(): String
+ move(Block room): void
+ getCharacterInfo(): String

<<interface>>
CharacterPlayer

+ getLocation(): Block
+ getCharacterName(): String
+ move(Block room): void
+ getCharacterInfo(): String
+ murder(CharacterTarget target, int damage): void
+ getItem(): List<Gadget>
+ pickItem(Gadget item): void
+ lookAround(): String
+ getPlayerId(): int
+ move(Block room): void
+ setItemLimit(int newItemLimit): void
+ useItem(Gadget item):void
+ useHighestItem(): void
+ canSeeEachOther(CharacterPlayer player): boolean

Player

- itemLimit: int
- item: List<Gadget>
- playerId: int
- currentRoom: Block
- name: String
- murderPoint: int

+ getLocation(): Block
+ getCharacterName(): String
+ murder(CharacterTarget target, int damage): void
+ toString(): String
+ hashCode(): int
+ equals(Player player): boolean
+ getCharacterInfo(): String
+ getItem(): List<Gadget>
+ pickItem(Gadget): void
+ lookAround(): String
+ getPlayerId(): int
+ move(Block room): void
+ setItemLimit(int newItemLimit): void
+ useItem(Gadget item):void
+ useHighestItem(): void
+ canSee(CharacterPlayer player): boolean

<<interface>>
CharacterPet

+ getLocation(): Block
+ getCharacterName(): String
+ move(Block room): void
+ getCharacterInfo(): String

Pet

- currentRoom: Block
- name: String

+ getLocation(): Block
+ getCharacterName(): String
+ move(Block room): void
+ getCharacterInfo(): String

World

- rooms: List<Block>
- items: List<Gadget>
- target: CharacterTarget
- taregetHealth: int
- worldText: String
- rows: int
- cols: int
- roomData: List<String[]>
- itemData: List<String[]>
- targetName: String
- worldName: String
- pet: CharacterPet

+ getWorldText(): String
+ getRoomCount(): int
+ getItemCount(): int
+ getTarget(): CharacterTarget
+ toString(): String
+ hashCode(): int
+ equals(world: World): boolean
+ createItem(String name, int location, int murderValue): Gadget
+ createTarget(String name, Block room, int health): CharacterTarget
+ createRoom(String roomName, int roomId, int[] coordinates, List<String[]> allRoomData): Block
+ drawWorld(): void
+ setWorldText(): void
+ getRooms(): List<Block>
+ getItems(): List<Gadget>
+ getRoomData(): List<String[]>
+ getItemData(): List<String[]>
+ moveTargetToRoom(String roomName): String
+ moveTargetToNextRoom(): String
- establishRoomNeighbors(): void
- establishRoomVisble(): void
+ createPlayer(String playerName, int startRoomIndex): CharacterPlayer
+ getRoomOccupants(Block room): String
+ setItemLimit(int newItemLimit): void
+ displayRoomInfo(String roomName): String
+ callCreateRoom(String roomName, int roomId, int[] coordinates, List<String[]> allRoomData): String
+ callCreateTarget(String name, Block room, int health): String
+ callCreateItem(String name, int location, int murderValue): String
+ callCreatePlayer(String playerName, int startRoomIndex): int
- getPlayerById(int playerId): CharacterPlayer
- getRoomById(int playerId): Block
+ getTargetInfo(): String
+ getPlayerInfo(int playerId): String
+ setMaxTurn(int maxTurn): void
+ getMaxTurn(): int
+ movePlayer(int playerId, int roomId): String
+ playerPickUpItem(int playerId, String itemName): String
+ playerLookAround(int playerId): String
+ getRoomItems(int roomId): List<String>
+ getPlayerItems(int playerId): List<String>
+ getPlayerRoomId(int playerId): int
+ getNeighborRooms(int roomId): List<Integer>
+ getPetInfo(): String
+ movePet(int roomId): String
+ murderAttempt(int playerId): String
+ canPlayerBeSeenByAny(int playerId): boolean
+ initializePetDFS(): void
+ movePetToNextRoom(): String
- dfsVisit(Block room, Set<Integer> visited, Stack<Block> path): void
+ movePet(int playerId, int targetRoomId): String
+ canInteractWithPet(int playerId): boolean
+ getTargetHealthPoint(): int
- getItemByName(String itemName): Gadget
+ getPlayerItems(int playerId): List<String>
+ usePlayerItem(int playerId, String itemName): void
+ removePet(): void
+ getPlayerLocation(int playerId): String
+ getPlayerItemsInfo(int playerId): String

<<interface>>
Block

+ getRoomName(): String
+ getNeighbor(): List<Block>
+ isAdjacent(Block room): boolean
+ isVisibleFrom(Block room): boolean
+ getItem(): List<Gadget>
+ getVisibleFrom(): List<Block>
+ getRoomId(): int
+ getNeighborNames(): String
+ getInfo(): String
+ getCoordinates(): int[]
+ addItem(Gadget item): void
+ addNeighbor(Block room): void
+ addVisibleFromRoom(Block room): void

Room

- roomName: String
- roomId: int
- locationRoom: List<Integer> //(rowStart: int, colStart: int, rowEnd: int, colEnd: int)
- item: List<Gadget>
- neighbor: List<Block>
- visbleFrom: List<Block>
- allRoomData: List<String[]>

+ getRoomName(): String
+ getNeighbor(): List<Block>
+ isAdjacent(Block room): boolean
+ isVisibleFrom(Block room): boolean
+ getItem(): List<Gadget>
+ getVisibleFrom(): List<Block>
+ getRoomID(): int
+ getNeighborNames(): String
- areNeighbors (Room room): boolean
- canSeeFrom(Room room): boolean
+ getInfo(): String
+ getCoordinates(): int[]
+ addItem(Gadget item): void
+ addNeighbor(Block room): void
+ addVisibleFromRoom(Block room): void

<<interface>>
Gadget

+ getItemName(): String
+ getMurderValue(): int
+ getLocationItem(): List<Integer>

Item

- name: String
- locationItem: Block
- murderValue(): int

+ getItemName(): String
+ getMurderValue(): int
+ getLocationItem(): List<Integer>
+ toString(): String
+ hashCode(): int
+ equals(Item item): boolean

1..*
0..*
1
*

c					
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 getNeighborNames(): String					
it should return all neighbors of this Room object	(For Armory)Nothing	Drawing Room, Liliard Room, Dining Hall			
Test getRoomID(): int					
it should return the roomID of this room, which coreesponds to the sequence in which the room appear in the input file	(For Armory)Nothing	0			
Test getInfo(): String					

It should return all detail of selected room. Including Room ID, Room Name, Neigbors of this room, Visible from this room, Items in this room	(For Armory)Nothing	Room Name: Armory Room ID: 0 Coordinates: [22, 19, 23, 26] Neighbors: Billiard Room, Dining Hall, Drawing Room Visible From: Billiard Room, Dining Hall, Drawing Room, Library, Master Suite, Nursery, Tennessee Room, Trophy Room, Wine Cellar Items: Revolver
Test getCoordinates(): int[]		
It should return the four number when we contruct room to represent the location of selected room	(For Armory)Nothing	[22, 19, 23, 26]
Test addItem(Item item): void		
It should add the item into the selected room	(Item) Revolver	Nothing will return, but revolver will be added into the room's item list
Wrong type object(not item)	String Revolver	throws IllegalArgumentException
Test addNeighbor(Room room): void		
Add the room into the neigbor list of selected Room	(Room) Dining Hall	Nothing will return, but Dining Hall will be added in to neighbor list of Armory
Wrong type of parameter, not room Object	(Item) Revolver	throws IllegalArgumentException
Test addVisibleFromRoom(Room room): void		
Add the room into the visble list of selected Room	(Room) Dining Hall	Nothing will return, but Dining Hall will be added in to visble list of Armory
Wrong type of parameter, not room Object	(Item) Dining Hall	throws IllegalArgumentException
Test canSeeFrom(Room otherRoom): boolean		
Use coordinates and world infomation to check is two room are neighbor or not. This will be checked during initialization of world.	(Room) Dining Hall	TRUE
Wrong type of parameter, not room Object	(Item) Dining Hall	throws IllegalArgumentException
Test areNeighbors(Room room): boolean		
Use coordinates and world infomation to check is two room are visible or not. This will be checked during initialization of world.	(Room) Dining Hall	TRUE
Wrong type of parameter, not room Object	(Item) Dining Hall	throws IllegalArgumentException
Test getVisibleFrom():List<Room>		
Return the list of room from the selected room	(For Armory)Nothing	List<Room>
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, damage: int): boolean		
It should try to kill the target by reducing target's health point by certain amount	Doctor Lucky, 5	True if muder success. 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

Test getCharacterInfo(): String					
It should return the character info of the player.	Nothing	Player Name: Luke Skywalker Current Location: Master Suite Item: Light Saber			
Test getItem(): List<Item>					
It should return the item this player has in a list	Nothing	[light saber, light gun]			
Test pickItem(item: Item): void					
It should allow player to pick one of items in the room the player is currently in	Item Object	Nothing, but this item should be added into the player's item list, and remove from item list of the room			
Wrong type of parameter cannot be passed	Room object	throws IllegalArgumentException			
The item is not in the itemlist of the room that the player is currently in cannot be picked	Item Nuclear Weapon	throws IllegalArgumentException			
Test lookAround():String					
It should return all information of all the room that are neighbors of the room where the player stays at	Nothing	It should include room name, statues of player or target who are in that room, the items in that room of all neigbors of the rooms where the player stay at			
Test getPlayerId():int					
It should return the id of the selected player	Nothing	997			
Test getLocation(): Room					
It should return the Room object the character object locate at	Nothing	Room Object			
Test getCharacterName(): String					
It should return the name of the character as String	Nothing	Doctor Lucky			
Test move(room: Room): void					
It should move the character 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			
Test setItemLimit(int newItemLimit): void					
It should change the item limit of a player to certain value	3	The itemLimit of a player should be set to 3			
The input should only be int	"a"	throws IllegalArgumentException			
The input should not be negative	-1	throws IllegalArgumentException			
Test useItem(Gadget item):void					
It should allow a player to use the item they have to update the attack damage when call murder. The used item should be removed from player's item list.	Gadget object (Revolver)	The murder damage should become 3, and the revolver should be removed from player list.			
Use an item not in player's itemlist is not prohibited.	Gadget object (Light Saber)	throws IllegalArgumentException			
Wrong type input is prohibited.	"Revolver"	throws IllegalArgumentException			
Test useHighestItem(): void					
It allows a player(mostly computer player) to use the highest murder value item in it's list. The used item should be remove from player's list. If the player has no item, it will only attack with bare hand.	None	The murder damage should be the same as the highest murder value in item list , and the item with highest murder value should be removed from player list.			
Test canSee(CharacterPlayer player): boolean					
This method to check can two players can see each other or not when they are in the same room or rooms are neighbor	CharacterPlayer object (Player2)	TRUE			
If they are not in the same room or rooms are neighbor	CharacterPlayer object (Player2)	FALSE			
If the pet is in the same room as the CharacterPlayer being checked	CharacterPlayer object (Player2)	FALSE			
Wrong type input is prohibited.	"Player2"	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			
Test getsetHealthPoint(): 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 setHealthPoint(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			
Test getCharacterInfo(): String					
It should return the character info of the target.	Nothing	Target Name: Doctor Lucky Health Points: 50 Current Location: Master Suite			
Test getLocation(): Room					
It should return the Room object the character object locate at	Nothing	Room Object			
Test getCharacterName(): String					
It should return the name of the character as String	Nothing	Doctor Lucky			
Test move(room: Room): void					
It should move the character 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			
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			
Test getWorldText():String					
It should return the current status of the board.	Nothing	World Name: Doctor Lucky's Mansion World Dimensions: 30x36 Number of Rooms: 21 Number of Items: 20			
Test toString(): String					
It should return a short description of the world	Nothing	The board is mansion!			
Test createItem(String name, int location, int murderValue): Item					
It should create an item into the room with the location roomId	"light saber", 0, 100	Iterm object			
Passing wrong type of parameter should throw an exception	"light saber", Room armory, 100	throws IllegalArgumentException			
Test createTarget(String name, Room room, int health): Target					
It should create an target into the room we passed	"Dr Lucky", Room armory, 50	Target Object			
Passing wrong type of parameter should throw an exception	"Dr Lucky", 3, 50	throws IllegalArgumentException			
Test createRoom(String roomName, int roomId, int[] coordinates, List<String[]> allRoomData): Room					
It should create an room object in the location we designed into the world	armory, 0, [0,1,0,1], List<string[]>	Room Object			
Passing wrong type of parameter should throw an exception	armory, 0, 0,1,0,1, List<string[]>	throws IllegalArgumentException			
Test setWorldText(): void					
It should update the world text since the world is changing	Nothing	Nothing, but the world text should be updated			
Test getRooms(): List<Room>					
It should return the List of Room object	Nothing	List of room object			
Test getItems(): List<Item>					
It should return the List of Item object	Nothing	List of Item object			
Test getRoomData(): List<String[]>					
It should return the all room data we write when we read the set up file	Nothing	List<String[]>			
Test getRoomData(): List<String[]>					
It should return the all item data we write when we read the set up file	Nothing	List<String[]>			
Test moveTargetToRoom(String roomName): void					
It should move the target object into the room we selected	"Armory"	Nothing, but target object should be putted into room armory			
Passing wrong type of parameter should throw an exception	Room armory	throws IllegalArgumentException			
The room name is not in the room list of the world is created	"Mars"	throws IllegalArgumentException			
Test moveTargetToNextRoom(): void					
it should move the target to the room with next roomId	Nothing	Nothing, but target should move to the next room order			
Test getRoomOccupants(Block room): String					
It should return all occupants character in that room	Armory	Target: Doctor Lucky; Player: player A			
Wrong type of parameter cannot be passed	"Armory"	throws IllegalArgumentException			
Not existed Block object cannot be passed	Block (Mars)	throws IllegalArgumentException			

Test setItemLimit(int newItemLimit): void					
It should update the itemLimit for all player	3	Nothing, but item limit for each player should be updated to 3			
Wrong type of parameter cannot be passed	"A"	throws IllegalArgumentException			
Test displayRoomInfo(String roomName): String					
It should display room detail	"Armory"	Room Name: Armory Room ID: 1 Coordinates: [22, 19, 23, 26] Neighbors: Billiard Room, Dining Hall, Drawing Room Visible From: Billiard Room, Dining Hall, Drawing Room, Library, Master Suite, Nursery, Tennessee Room, Trophy Room, Wine Cellar Items: Revolver Occupants: Target: Doctor Lucky , Player: Human, Player: Computer			
Wrong type of parameter cannot be passed	Block (Armory)	throws IllegalArgumentException			
The string is not room name cannot be passed	"Bus"	throws IllegalArgumentException			
Test callCreateRoom(String roomName, int roomId, int[] coordinates, List<String[]> allRoomData): String					
The method should call createRoom method in model.	"Armory", 0, [0,0,1,1], List<String[]> allRoomData	Armory created			
Wrong type of parameter cannot be passed	Armory, 0, [0,0,1,1], List<String[]> allRoomData	throws IllegalArgumentException			
Test callCreateTarget(String name, Block room, int health): String					
The method should call createTarget method in model.	"Dr. Lucky", Block(Armory), 50	Target created			
Wrong type of parameter cannot be passed	"Dr. Lucky", "Armory", 50	throws IllegalArgumentException			
Test callCreateItem(String name, int location, int murderValue): String					
The method should call createItem method in model.	"Knife", 0, 3	Item created			
Wrong type of parameter cannot be passed	"Knife", "Armory", 3	throws IllegalArgumentException			
Test callCreatePlayer(String playerName, int startRoomIndex): int					
The method should call createPlayer method in model, and return playerId	"PlayerA", 0	0			
Wrong type of parameter cannot be passed	"PlayerA", Armory	throws IllegalArgumentException			
Test getTargetInfo(): String					
It should return the target infomation	None	Target is in Armory			
Test getPlayerInfo(int playerId): String					
It should return Player info with cerrtain playerId	0	Player A, location: Armory, Item: None			
Wrong type of parameter cannot be passed	a	throws IllegalArgumentException			
None exist playerId cannot be passed	999	throws IllegalArgumentException			
Test setMaxTurn(int maxTurn): void					
It will set the MaxTurn in the world	50	Nothing, but maxTurn will become 50			
Wrong type of parameter cannot be passed	"50"	throws IllegalArgumentException			
Test getMaxTurn(): int					
It will get the MaxTurn in the world	Nothing	500			
Test movePlayer(int playerId, int roomId): String					
It will call move within player class, and make player move	0, 0	Player with id 0 will try to move to Armory with room id 0			
Wrong type of parameter cannot be passed	"Peter", 0	throws IllegalArgumentException			
Non existing room id and player id should not be allowed	999,-1	throws IllegalArgumentException			
Test playerPickUpItem(int playerId, String itemName): String					
It will make player try to pick up certain item in room	0, "Revolver"	Player A picks up Revolver			

Wrong type of parameter cannot be passed	"Eric", "Revolver"	throws IllegalArgumentException			
Non existing player id should not be allowed	-1, "Revolver"	throws IllegalArgumentException			
Item not in the room is not allowed	-1, "Light Saber"	throws IllegalArgumentException			
Test playerLookAround(int playerId): String					
It call LookAround method in player class	0	Start Looking Around Computer player: You are in Room Armory. Current Room ID: 1 Current Room Name: Armory Current Room Items: None Neighboring and Visible Rooms: Room ID: 2, Room Name: Billiard Room, Items: Billiard Cue Room ID: 4, Room Name: Dining Hall, Items: None Room ID: 5, Room Name: Drawing Room, Items: None Other Visible Rooms: Room ID: 11, Room Name: Library, Items: None Room ID: 13, Room Name: Master Suite, Items: Shoe Horn Room ID: 14, Room Name: Nursery, Items: Bad Cream Room ID: 18, Room Name: Tennessee Room, Items: None Room ID: 19, Room Name: Trophy Room, Items: Duck Decoy, Monkey Hand Room ID: 20, Room Name: Wine Cellar, Items: Rat Poison, Piece of Rope			
Wrong type of parameter cannot be passed	"Eric"	throws IllegalArgumentException			
Non existing player id should not be allowed	999	throws IllegalArgumentException			
Test getRoomItems(int roomId): List<String>					
It return all items with list of string in the certain room with id	0	["Revolver"]			
Wrong type of parameter cannot be passed	"Armory"	throws IllegalArgumentException			
Non existing room id should not be allowed	999	throws IllegalArgumentException			
Test getPlayerRoomId(int playerId): int		0			
It will return the room id of player's current location	0	throws IllegalArgumentException			
Wrong type of parameter cannot be passed	"Eric"	throws IllegalArgumentException			
Non existing player id should not be allowed	999				
Test getNeighborRooms(int roomId): List<Integer>					
It will return list of int with neighbors room id of selected room	0	[2,4,5]			
Wrong type of parameter cannot be passed	"Armory"	throws IllegalArgumentException			
Non existing room id should not be allowed	999	throws IllegalArgumentException			
Test getPetInfo(): String					
It should return a string about pet location	None	Pet: Cat, Location: Armory			
Test movePet(int playerId, int roomId): boolean					
The method should move the pet within the same room of player into the other room. If it is success, return True.	0,0	TRUE			
Wrong type input is prohibited	0, "Armory"	throws IllegalArgumentException			
Non existing player id or room id should not be passed	999, 999	throws IllegalArgumentException			
Test murderAttempt(int playerId): String					
This method should call murder method in player class to murder target. It murder success, return success. Also the game should end when the target health drop equal or less than 0	0	Murder Successfuly			
When the murder can be seen by other player, it should fail	0	Murder Failed			
When the pet is in the same room as the target, it should fail	0	Murder Failed			
Wrong type input is prohibited	0	throws IllegalArgumentException			
Non existing player id should not be passed	999	throws IllegalArgumentException			
target					
Class Pet					
Test Pet Constructor					
create a Pet object with correct parameters	Pet("Cat", Piazza)	Pet object created			
create a Pet object with incorrect parameters (wrong types of parameter or wrong numbers of parameters)	Pet("Cat", Piazza)	throws IllegalArgumentException			
Test getCharacterInfo(): String					

It should return the character info of the pet.	Nothing	Pet Name: Cat Current Location: Master Suite			
Test getLocation(): Room					
It should return the Room object the character object locate at	Nothing	Room Object			
Test getCharacterName(): String					
It should return the name of the character as String	Nothing	Cat			
Test move(room: Room): void					
It should move the character 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			
Controller Part					
All Player Actions are method in playGame(WorldOutline mode):void					
PlayerInfo Method					
Test PlayerInfo(int playerId)					
The command should call displayPlayerInfo method in the model end.	PlayerInfo(playerId = 1)	The info of player 1 should be displayed after excute run() method			
Having a non exist player id in the field should not allowed	PlayerInfo(playerId = 1)	throws IllegalArgumentException			
Having a wrong type parameter should not allowed	PlayerInfo(playerId = "1")	throws IllegalArgumentException			
RoomInfo Method					
Test RoomInfo(int playerId)					
The command should call displayRoomInfo method in the model end.	RoomInfo(playerId = 1)	The info of room where the player 1 stays at should be displayed after excute run() method			
Having a non exist player id in the field should not allowed	RoomInfo(playerId = 999)	throws IllegalArgumentException			
Having a wrong type parameter should not allowed	RoomInfo(playerId = "1")	throws IllegalArgumentException			
Move Method					
Test setupGame(WorldOutline world): void runGame(WorldOutline world): void playGame(WorldOutline world): void					
Those method should make the controller end running the game	world model	Should return a string about game running status			
Passing incorrect model should not allowed	null	throws IllegalArgumentException			
Test Move(int playerId, String itemName)					
The command should call move method in the model end.	Move(playerId = 1, roomName = "Dining Hall")	The player 1 should move to Dining Hall after excute run() method			
Having non exist player id or non exist room name in the field should not allowed	Move(playerId = 999, roomName = "Earth")	throws IllegalArgumentException			
Having a wrong type parameter should not allowed	Move(playerId = "1", roomName = "Dining Hall")	throws IllegalArgumentException			
Pick Method					
Test Pick(int playerId, String itemName)					
The command should call pick method in the model end.	Pick(playerId = 1, itemName = "Revolver")	The item resolver should be picked up for player 1 after excute run() method			
Having non exist player id or non exist item name in the field should not allowed	Pick(playerId = 999, itemName = "Saber")	throws IllegalArgumentException			
Having a wrong type parameter should not allowed	Pick(playerId = "1", roomName = "Dining Hall")	throws IllegalArgumentException			
Test createGraph (World): String					
It should create the graph of current world and save it as png	world model	Should return a string says "Current graph is created".			
Passing incorrect model should not allowed	null	throws IllegalArgumentException			
CreatePlayer Method					
Test CreatePlayer(int playerId, String roomName, String: playerName)					
The command should call createPlayer method in the model end.	CreatePlayer(1, "Dining Hall", "Luke")	The player name Liuke should be created in Dining Hall after excute run() method			
Having non exist player id or non exist room name in the field should not allowed	CreatePlayer(999, "Dining Hall", "Luke")	throws IllegalArgumentException			

Passing incorrect model should not allowed	CreatePlayer(1, Room diningHall, "Luke")	throws IllegalArgumentException			
LookAround Method					
Test LookAround (int playerId)					
The command should call lookAround method in the model end.	LookAround(1)	The player 1 should get nearby info after excute run() method			
Having non exist player id in the field should not allowed	LookAround(999)	throws IllegalArgumentException			
Passing incorrect model should not allowed	LookAround("1")	throws IllegalArgumentException			
murderAttempt Method					
Test murder (int playerId, String itemName)					
The command should call muderAttempt method in the model end.	murderAttempt(1, "Revolver")	The player 1 should attack target with weapon revolver after excute run() method			
Having non exist player id or non exist room name in the field should not allowed	murderAttempt(-1, "Revolver")	throws IllegalArgumentException			
Passing incorrect model should not allowed	murderAttempt(1, Item revolver)	throws IllegalArgumentException			
Test getTargetInfo()					
It should call getTargetInfo in model end, and the same string should be printed out in command	None	None on controller end, but the string from model end should be returned and printed.			
Test movePet(): void					
It should call movePet in model end, and the same string should be printed out in command	None	None on controller end, but the string from model end should be returned and printed.			