MovePlayer
- playerName: String
- targetSpace: int
PickUpItem
- playerName: String
- itemName: String
CreatePlayer
- name: String
- currentSpaceIndex: int
- maxItems: int
- rg: RandomGenerator
MaxTurn
- maxTurn: int
Parse
- fileReader: FileReader
1
1
1
1
1
1
RandomGenerator
- random: Random
- mockedValues: int[]
- currentIndex: int
+ nextInt(bound: int): int
<<KillDoctorLuckyController>>
+ playGame(m: KillDoctorLucky)
KillDoctorLuckyConsoleController
- out: Appendable
- scan: Scanner
- rg: RandomGenerator
- killDoctorLuckyCommand: KillDoctorLuckyCommand
- filePath: String
- appendCommand(str: String)
- checkQuit(input: String): Boolean
- checkInt(input: String): Boolean
<<KillDoctorLuckyCommand>>
+ execute(m: KillDoctorLucky)
1
1
<<Mansion>>
+ getSpacesNum(): int
+ setSpacesNum(spacesNum: int)
+ getItemsNum(): int
+ setItemsNum(ItemsNum: int)
+ addSpace(index: int, name: String, points: String[])
+ getSpaces(): List<Space>
+ getHeight(): int
+ getwidth(): int
+ getSpaceByIndex(index: int): Space
<<KillDoctorLucky>>
+ setMansion(mansionName: String, mansionHeight: int, mansionWidth: int)f
+ setDoctorLucky(doctorName: String, doctorHealth: int)
+ setMaxTurn(maxTurn: int)
+ setPlayer(playerName: String, spaceIndex: int, maxItem: int)
+ getPlayersInfo(): String
+ getPlayerInfoByName(name: String): String
+ getMaxTurn: int
+ getPlayerNameByTurn(): String
+ initiateGame(readable: Readable)
+ outputMap(): String
+ movePlayer(playerName: String, targetSpace: int)
+ pickUpItem(playerName: String, itemName: String)
+ getAroundInfo(playerName: String)
+ getTurns(): int
+ getMansionInfo(): String
+ getDoctorLuckyInfo(): String
+ getCurrentSpaceIndexByPlayerName(playerName: String): String
+ getNeighborsBySpaceIndex(index: int): List<Space>
+ getItemsBySpaceIndex(index: int): List<Item>
+ gePlayerTypeByName(playerName: String): PlayerType
+ getDoctorLuckyName(): String
+ getDoctorLuckyCurrentSpaceIndex(): int
+ getDoctorLuckyHealth(): int
+ getSpaceNameByIndex(index: int): String
+ getSpaceInfo(index: int): String
+ setPet(petName: String)
+ getPetCurrentSpaceIndex(): int
+ getSpaceNumFromMansion(): int
+ movePet(index: int)
+ getItemsByPlayerName(name: Sting): List<Item>
+ getMostDamageItemNameByPlayerName(playerName: String): String
+ makeAttempt(playerName: String. itemName: String): Boolean
+ getPetInfo(): String
+ getNeighborsInfoBySpaceIndex(spaceIndex: int): String
+ getItemsInfoBySpaceIndex(spaceIndex: int): String
KillDoctorLuckyModel
- doctorLucky: DoctorLucky
- mansion : Mansion
- players : List<Player>
- maxTurn: int
- turn: int
- pet: PetModel
- splitLine(line: String): String[]
- newTurn(): void
- getPlayerByName(name: String): Player
1
1
MansionModel
- name: String
- height : int
- width : int
- spacesNum: int
- itemsNum: int
- spaces: List<Space>
1
1
0..*
0..*
ItemModel
- name: String
- position : int
- damage : int
<<Item>>
+ getPosition(): int
+ getDamage(): int
0..*
1
21..*
SpaceModel
- index: int
- name : String
- items : List<Item>
- points: int[]
- neighbors: List<Space>
- players: List<Player>
- isNeighbor(pointsIn: int[]): boolean
1
<<Space>>
+ getItems(): List<Item>
+ getPoints(): int[]
+ getNeighbors(): List<Space>
+ addItem(name: String, position: int, damage: int)
+ addNeighbor(newSpace: Space)
+ addPlayer(player: Player)
+ getIndex(): int
+ getItemByname(name: String): Item
+ removeItem(item: Item)
<<Nameable>>
+ getName(): String
1
0..*
<<Player>>
+ getItems(): List<Item>
+ pickUpItem(item: Item)
+ getPlayerType(): PlayerType
+ getMaxItems(): int
+ removeItemByName(itemName: String)
+ getItemNameWithMaxDamage(): String
<<Enumeration>>
PlayerType
HUMAN
ROBOT
1
1
1..*
PlayerModel
- items: List<Item>
- maxItems : int
- PlayerType: playerType
CharacterModel
# name: String
# currentSpaceIndex : int
<<Character>>
+ getCurrentSpaceIndex(): int
+ move(spaceIndex: int): int
DoctorLuckyModel
- health: int
1
<<DoctorLucky>>
+ getHealth(): int
+ deductHealth(damage: int)
PetModel
- route: List<Integer>
- visited: Set<Integer>
- dfs(index: int. mansion: Mansion)
1
<<Pet>>
+ createRoute(index: int, mansion: Mansion)
+ nextSpaceIndexInRoute: (currentSpaceIndex: int)

KillDoctorLuckyConsoleController		
Test Case	Input	Expected Outcome
Constructor - Valid Input	KillDoctorLuckyConsoleController controller = new KillDoctorLuckyConsoleController(new StringReader("input"), new StringBuilder(), new RandomGeneratorImpl());	No Exception is thrown, and the controller object is created.
Constructor - Null Readable or Appendable	KillDoctorLuckyConsoleController controller = new KillDoctorLuckyConsoleController(null, new StringBuilder(), new RandomGeneratorImpl());	IllegalArgumentException is thrown.
playGame - Valid Game Flow	Create a test environment with necessary input.	Executes the game flow without errors, displaying prompts and game actions.
playGame - Invalid Specification File	Create a test environment with a non-existent specification file.	Controller displays an error message and prompts for a valid file.
playGame - Player Move Choice	Create a test environment and simulate player choice.	Controller correctly processes the player's move choice.
playGame - Player Item Pickup Choice	Create a test environment and simulate player choice.	Controller correctly processes the player's item pickup choice.
playGame - Player Look Around Choice	Create a test environment and simulate player choice.	Controller correctly processes the player's "look around" choice.
playGame - Player Pet Move Choice	Create a test environment and simulate player choice.	Controller correctly processes the player's move pet choice.
playGame - Player Make an Attempt Choice	Create a test environment and simulate player choice.	Controller correctly processes the player's "make an attempt" choice.
joinNames - Valid List of Objects	A list of objects with names.	Returns a string containing the joined names separated by commas.
joinNames - Empty List of Objects	An empty list of objects.	Returns an empty string.
appendCommand - Valid Input	controller.appendCommand("Test Message");	The message is appended to the Appendable without errors.
appendCommand - IOException	controller.appendCommand("Test Message"); with Appendable that throws an IOException.	IllegalStateException is thrown with a wrapped IOException.
CreatePlayer		
Test Case	Input	Expected Outcome
Constructor - Valid Input	CreatePlayer createPlayer = new CreatePlayer(new RandomGeneratorImpl());	No Exception is thrown, and the object is created.
Constructor - Name and Space Index	CreatePlayer createPlayer = new CreatePlayer("Player1", 1, 5);	No Exception is thrown, and the object is created with the specified parameters.
execute - Auto-Player Creation	Create a KillDoctorLucky model, execute with name = null.	Auto-Player is created with random values for name, currentSpaceIndex, and maxItems.
execute - Player Creation	Create a KillDoctorLucky model, execute with valid name and parameters.	Player is created with the specified name, currentSpaceIndex, and maxItems.
MaxTurn		

Test Case	Input	Expected Outcome
Constructor - Valid Input	MaxTurn maxTurnCommand = new MaxTurn(10);	No Exception is thrown, and the object is created.
execute - Set Max Turn	Create a KillDoctorLucky model, execute with maxTurn = 10.	The maximum turn is set to 10 in the model.
execute - Null Model	Execute the command with a null model.	IllegalArgumentException is thrown.

Parse

Test Case	Input	Expected Outcome
Constructor - Valid Input	Parse parseCommand = new Parse("specification.txt");	No Exception is thrown, and the object is created.
Constructor - File Not Found	Parse parseCommand = new Parse("nonexistent.txt");	FileNotFoundException is thrown.
execute - Parse Specification File	Create a KillDoctorLucky model and execute the command with a valid specification file.	The model is populated with mansion details, target character, spaces, and items.
execute - Null Model	Execute the command with a null model.	IllegalArgumentException is thrown.
execute - Malformed File	Create a KillDoctorLucky model and execute the command with a specification file that has incorrect formatting.	IllegalStateException is thrown.
Valid Specification File	Valid file with correct formatting	Model populated with mansion details, target character, spaces, and items.
Invalid Specification File - Missing Mansion Information	File missing mansion information	IllegalStateException is thrown due to missing mansion information.
Invalid Specification File - Incorrect Format	File with incorrect formatting	IllegalStateException is thrown due to incorrect format.
Null Model	Execute command with a null model	IllegalArgumentException is thrown since the model cannot be null.
Maximum Items Section Missing	File missing the section for items	Model populated with mansion details, target character, and spaces, but no items added.
Spaces Without Neighbors	File with spaces but without neighbors	Spaces created, but no neighbors defined.
Target Character Name Only	File with target character name only	Target character created with the name, index set to 0 by default.
Target Character Index Only	File with target character index only	Target character created with the index, name set to null by default.
Empty Specification File	An empty specification file	Model remains unchanged as there is no data in the file

MovePlayer

Test Case	Input	Expected Outcome
Constructor Test 1	MovePlayer("Player1", 4)	Create a MovePlayer instance with playerName set to "Player1" and targetSpace set to 4.
Constructor Test 2	MovePlayer("Player2", 7)	Create a MovePlayer instance with playerName set to "Player2" and targetSpace set to 7.
Execute Test 1	Create a KillDoctorLucky instance, create a MovePlayer instance with playerName set to "Player1" and targetSpace set to 4, then execute the MovePlayer command.	The player with the name "Player1" is moved to space 4 in the game.

Execute Test 2	Create a KillDoctorLucky instance, create a MovePlayer instance with playerName set to "Player2" and targetSpace set to 7, then execute the MovePlayer command.	The player with the name "Player2" is moved to space 7 in the game.
PickUpItem		
Test Case	Input	Expected Outcome
Constructor Test 1	PickUpItem("Player1", "Item1")	Create a PickUpItem instance with playerName set to "Player1" and itemName set to "Item1".
Constructor Test 2	PickUpItem("Player2", "Item2")	Create a PickUpItem instance with playerName set to "Player2" and itemName set to "Item2".
Execute Test 1	Create a KillDoctorLucky instance, create a PickUpItem instance with playerName set to "Player1" and itemName set to "Item1", then execute the PickUpItem command.	Player "Player1" picks up item "Item1" in the game.
Execute Test 2	Create a KillDoctorLucky instance, create a PickUpItem instance with playerName set to "Player2" and itemName set to "Item2", then execute the PickUpItem command.	Player "Player2" picks up item "Item2" in the game.
RandomGenerator		
Test Case	Input	Expected Outcome
Constructor - Real Random Generator	RandomGenerator randomGen = new RandomGenerator();	No Exception is thrown, and the object is created with a real random generator.
Constructor - Mocked Random Generator	RandomGenerator randomGen = new RandomGenerator(1, 2, 3);	No Exception is thrown, and the object is created with mocked values.
nextInt - Real Random Generator	Generate random integers using a real random generator.	Random integers within the specified bound are generated.
nextInt - Mocked Random Generator	Generate integers using a mocked generator.	Integers from the mocked values are generated in the same order.
nextInt - No More Mocked Values	Attempt to generate values when there are no more mocked values.	IllegalStateException is thrown.
PetModel		
Test Case	Input	Expected Outcome
Constructor - Valid Input	PetModel petModel = new PetModel("Cat");	No Exception is thrown, and the object is created with the specified name.
toString - Valid name	Create a PetModel with a valid name value and call toString.	Returns a string representation of the PetModel object with the name.
hashCode - Same Name	Create two PetModel objects with the same name and compare their hash codes.	Hash codes of both objects should be equal.
equals - Equal Objects	Create two PetModel objects with the same name and compare them using the equals method.	The equals method should return true as the objects have the same name.

equals - Not Equal Objects	Create two PetModel objects with different names and compare them using the equals method.	The equals method should return false as the objects have different names.
equals - Comparison with Different Class	Compare a PetModel object with an object of a different class using the equals method.	The equals method should return false as the classes are different.
DoctorLuckyModel		
Test Case	Input	Expected Outcome
Constructor - Valid Input	DoctorLuckyModel doctorLucky = new DoctorLuckyModel("Lucky", 100);	No Exception is thrown, and the object is created with the specified name and health.
Constructor - Invalid Health	DoctorLuckyModel doctorLucky = new DoctorLuckyModel("Lucky", -10);	IllegalArgumentException is thrown due to an invalid health value.
getHealth - Valid Health	Create a DoctorLuckyModel with a valid health value and call getHealth.	Returns the valid health value.
toString - Valid Health	Create a DoctorLuckyModel with a valid health value and call toString.	Returns a string representation of the DoctorLucky object with the name and health.
hashCode - Same Name	Create two DoctorLuckyModel objects with the same name and compare their hash codes.	Hash codes of both objects should be equal.
equals - Equal Objects	Create two DoctorLuckyModel objects with the same name and compare them using the equals method.	The equals method should return true as the objects have the same name.
equals - Not Equal Objects	Create two DoctorLuckyModel objects with different names and compare them using the equals method.	The equals method should return false as the objects have different names.
equals - Comparison with Different Class	Compare a DoctorLuckyModel object with an object of a different class using the equals method.	The equals method should return false as the classes are different.
PlayerModel		
Test Case	Input	Expected Outcome
Constructor - Valid Input	PlayerModel player = new PlayerModel("Alice", 1, 5);	No Exception is thrown, and the object is created with the specified name, currentSpaceIndex, and maxItems.
setItem - Add Item Below Max Limit	Create a PlayerModel with maxItems = 5, add 3 items using setItem.	Items are added successfully, and the items list contains 3 items.
setItem - Add Item at Max Limit	Create a PlayerModel with maxItems = 5, add 5 items using setItem.	The 5th item addition should throw an IllegalStateException since the maxItems limit is reached.
getItems - Get Items	Create a PlayerModel with items added, call getItems.	Returns the list of items added to the player.
toString - Valid Data	Create a PlayerModel with valid data and call toString.	Returns a string representation of the Player object with the name, maxItems, and currentSpaceIndex.
hashCode - Same Name	Create two PlayerModel objects with the same name and compare their hash codes.	Hash codes of both objects should be equal.

equals - Equal Objects	Create two PlayerModel objects with the same name and compare them using the equals method.	The equals method should return true as the objects have the same name.
equals - Not Equal Objects	Create two PlayerModel objects with different names and compare them using the equals method.	The equals method should return false as the objects have different names.
equals - Comparison with Different Class	Compare a PlayerModel object with an object of a different class using the equals method.	The equals method should return false as the classes are different.
getItemNameWithMaxDamage	Create a PlayerModel instance and pick up two Item instances using pickUpItem with different damages, then call getItemNameWithMaxDamage.	The name of the item with the maximum damage should be returned.
removeItemByName Test 1	Create a PlayerModel instance and call removeItemByName with the name of an item that does not exist in the player's items.	No items should be removed, and no exceptions should be thrown.
removeItemByName Test 2	Create a PlayerModel instance, pick up two Item instances using pickUpItem, and call removeItemByName with the name of one of the picked up items.	The specified item should be removed from the player's items.
removeItemByName Test 3	Create a PlayerModel instance, pick up two Item instances using pickUpItem, and call removeItemByName with the name of another item that does not exist in the player's items.	No items should be removed, and no exceptions should be thrown.
	ItemModel	
Test Case	Input	Expected Outcome
Constructor - Valid Input	ItemModel item = new ItemModel("Sword", 1, 10);	No Exception is thrown, and the object is created with the specified name, position, and damage.
getName - Get Name	Create an ItemModel and call getName.	Returns the name of the item.
getPosition - Get Position	Create an ItemModel and call getPosition.	Returns the position of the item.
getDamage - Get Damage	Create an ItemModel and call getDamage.	Returns the damage value of the item.
toString - Valid Data	Create an ItemModel with valid data and call toString.	Returns a string representation of the Item object with the name, position, and damage.
hashCode - Same Name	Create two ItemModel objects with the same name and compare their hash codes.	Hash codes of both objects should be equal.
equals - Equal Objects	Create two ItemModel objects with the same name and compare them using the equals method.	The equals method should return true as the objects have the same name.
equals - Not Equal Objects	Create two ItemModel objects with different names and compare them using the equals method.	The equals method should return false as the objects have different names.

equals - Comparison with Different Class	Compare an ItemModel object with an object of a different class using the equals method.	The equals method should return false as the classes are different.
KillDoctorLuckyModel		
Test Case	Input	Expected Outcome
Constructor - Valid Mansion Specification	Create a KillDoctorLuckyModel object with valid mansion specification.	The mansion and doctorLucky are correctly initialized, and no exceptions are thrown.
getMansion - Get Mansion	Create a KillDoctorLuckyModel object and call getMansion.	Returns the mansion object that was set using the constructor.
setDoctorLucky - Set DoctorLucky	Create a KillDoctorLuckyModel object and call setDoctorLucky to set the doctor's name and health.	The doctorLucky object is created with the specified name and health.
setPlayer - Set Player	Create a KillDoctorLuckyModel object and call setPlayer to add a player.	The player object is added to the players list with the specified name, space index, and max items.
getMansionInfo - Get Mansion Info	Create a KillDoctorLuckyModel object with a predefined mansion and call getMansionInfo.	Returns a string containing information about the mansion.
getPlayersInfo - Get Players Info	Create a KillDoctorLuckyModel object with predefined players and call getPlayersInfo.	Returns a string containing information about the players.
getMaxTurn - Get Max Turn	Create a KillDoctorLuckyModel object and call getMaxTurn.	Returns the maximum turn value that was set.
getPlayerByTurn - Get Player By Turn	Create a KillDoctorLuckyModel object with predefined players and call getPlayerByTurn with various turn numbers.	Returns the player corresponding to the given turn number.
getDoctorLucky - Get DoctorLucky	Create a KillDoctorLuckyModel object with a predefined doctorLucky and call getDoctorLucky.	Returns the doctorLucky object that was set.
setMansion - Set Mansion	Create a KillDoctorLuckyModel object and call setMansion to set the mansion name, height, and width.	The mansion object is created with the specified name, height, and width.
getMansionInfo Test	Create a KillDoctorLuckyModel instance with a pre-defined mansion and call getMansionInfo.	A string representation of the mansion should be returned.
getDoctorLuckyInfo Test	Create a KillDoctorLuckyModel instance with a pre-defined doctorLucky and call getDoctorLuckyInfo.	A string representation of the doctorLucky should be returned.
getPlayerByName Test 1	Create a KillDoctorLuckyModel instance with pre-defined players and call getPlayerByName with an existing player name.	The player with the specified name should be returned.
getPlayerByName Test 2	Create a KillDoctorLuckyModel instance with pre-defined players and call getPlayerByName with a non-existing player name.	null should be returned as there is no player with the specified name.

getCurrentSpaceIndexByPlayerName Test	Create a KillDoctorLuckyModel instance with pre-defined players and call getCurrentSpaceIndexByPlayerName with a player name.	The current space index of the specified player should be returned.
getNeighborsBySpaceIndex Test	Create a KillDoctorLuckyModel instance with a pre-defined mansion and call getNeighborsBySpaceIndex with a space index.	A list of neighboring spaces for the specified space index should be returned.
getItemsBySpaceIndex Test	Create a KillDoctorLuckyModel instance with a pre-defined mansion and call getItemsBySpaceIndex with a space index.	A list of items in the specified space should be returned.
gePlayerTypeByName Test	Create a KillDoctorLuckyModel instance with pre-defined players and call gePlayerTypeByName with a player name.	The player type of the specified player should be returned.
getDoctorLuckyName Test	Create a KillDoctorLuckyModel instance with a pre-defined doctorLucky and call getDoctorLuckyName.	The name of the doctorLucky should be returned.
getDoctorLuckyCurrentSpaceIndex Test	Create a KillDoctorLuckyModel instance with a pre-defined doctorLucky and call getDoctorLuckyCurrentSpaceIndex.	The current space index of the doctorLucky should be returned.
getDoctorLuckyHealth Test	Create a KillDoctorLuckyModel instance with a pre-defined doctorLucky and call getDoctorLuckyHealth.	The health of the doctorLucky should be returned.
getSpaceNameByIndex Test	Create a KillDoctorLuckyModel instance with a pre-defined mansion and call getSpaceNameByIndex with a space index.	The name of the space with the specified index should be returned.
getSpaceInfo Test	Create a KillDoctorLuckyModel instance with a pre-defined mansion and call getSpaceInfo with a space index.	A string representation of the space with the specified index should be returned.
setPet Test	Create a KillDoctorLuckyModel instance and call setPet with a pet name.	A new pet should be created with the specified name.
getPetName Test	Create a KillDoctorLuckyModel instance with a pre-defined pet and call getPetName.	The name of the pet should be returned.
getPetCurrentSpaceIndex Test	Create a KillDoctorLuckyModel instance with a pre-defined pet and call getPetCurrentSpaceIndex.	The current space index of the pet should be returned.
getSpaceNumFromMansion Test	Create a KillDoctorLuckyModel instance with a pre-defined mansion and call getSpaceNumFromMansion.	The number of spaces in the mansion should be returned.
movePet Test	Create a KillDoctorLuckyModel instance with a pre-defined pet and call movePet with an index.	The pet should be moved to the specified index.
	MansionModel	
Test Case	Input	Expected Outcome

Constructor - Valid Mansion	Create a MansionModel object with valid mansion specifications (name, height, and width).	The mansion object is correctly initialized with the specified name, height, and width. No exceptions are thrown.
getSpacesNum - Get Spaces Number	Create a MansionModel object and call getSpacesNum.	Returns the number of spaces that was set using the setSpacesNum method.
getItemsNum - Get Items Number	Create a MansionModel object and call getItemsNum.	Returns the number of items that was set using the setItemsNum method.
setSpacesNum - Set Spaces Number	Create a MansionModel object and call setSpacesNum to set the number of spaces.	The spacesNum is correctly set to the specified value.
setItemsNum - Set Items Number	Create a MansionModel object and call setItemsNum to set the number of items.	The itemsNum is correctly set to the specified value.
addSpace - Add Space to Mansion	Create a MansionModel object and call addSpace to add a space to the mansion with valid index, name, and points.	A space is added to the list of spaces with the specified index, name, and points. No exceptions are thrown.
getSpaces - Get Spaces List	Create a MansionModel object with predefined spaces and call getSpaces.	Returns a list of spaces that were added using the addSpace method.
toString - Convert to String	Create a MansionModel object with predefined name, height, width, spacesNum, and itemsNum and call toString.	Returns a string representation of the mansion including its name, height, width, spacesNum, and itemsNum.
getSpaceByIndex Test 1	Create a MansionModel instance, add spaces using addSpace, and call getSpaceByIndex with a valid index.	The space at the specified index should be returned.
getSpaceByIndex Test 2	Create a MansionModel instance, add spaces using addSpace, and call getSpaceByIndex with an index that is out of bounds.	null should be returned as there is no space at the specified index.
	SpaceModel	
Test Case	Input	Expected Outcome
Constructor Test	SpaceModel(index, name, [], points, [], [])	Space is created with provided parameters.
build Test	Set properties using builder and build	SpaceModel object created with builder properties.
isNeighbor Test	space.isNeighbor(validNeighborPoints)	true if points represent a neighbor, false otherwise.
getName Test	space.getName()	Returns the name of the space.
getItems Test	space.getItems()	Returns an unmodifiable list of items.
getPoints Test	space.getPoints()	Returns a clone of the points array.
getNeighbors Test	space.getNeighbors()	Returns an unmodifiable list of neighbors.
addItem Test	space.addItem(validItemName, validPosition, validDamage)	Item is added to the list of items.
addNeighbor Test	space.addNeighbor(newSpace)	New space added to list of neighbors if valid.
addPlayer Test	space.addPlayer(validPlayer)	Player is added to the list of players.
toString Test	space.toString()	Returns a string representation of the space.
hashCode Test	space.hashCode()	Hash code based on name and points.
equals Test	space.equals(sameNameSamePointsSpac e)	true if name and points match, false otherwise.

getIndex Test	space.getIndex()	Returns the index of the space.