CS5010 Milestone4 Preliminary Design

By Linhao Qian (NUID: 002325915)

UML Diagram

PreGamePanel
- serialVersionUID: long
- model: ReadOnlyWorld
- hintLabel: JLabel
- addComputerPlayerButton: JButton
- addHumanPlayerButton: JButton
- startGameButton: JButton

+ PreGamePanel(ReadOnlyWorld)
+ addActionListener(ActionListener)
+ refresh()

GamePanel
- serialVersionUID: long
- model: ReadOnlyWorld
- hintLabel: JLabel
- operationLabel: JLabel
- resultLabel: JLabel

+ GamePanel(ReadOnlyWorld)
+ setHint()
+ setResult(String)

FrameView
- serialVersionUID: long
- scrollAboutPane: JScrollPane
- scrollMainPane: JScrollPane
- aboutPanel: AboutPanel
- mainPanel: MainPanel

+ FrameView(ReadOnlyWorld)
+ enterGame()
+ getNewFile(): File
+ showGameInterface(World)
+ showError(String)
+ addKeyListener(KeyListener)
+ addActionListener(ActionListener)
+ getPlayerName(): String
+ getSpaceName(): String
+ getItemName(): String
+ getAttemptChoice(): String
+ getIntendedSpace(): String
+ addPlayer()
+ startGame()
+ setResult(String)
+ endGame(String)
+ configureSpaceClickListener(MouseAdapter)
+ configurePlayerClickListener()
+ resetFocus()

AboutPanel
- serialVersionUID: long
- welcomeLabel: JLabel
- creditInfo: JLabel
- confirmButton: JButton

+ AboutPanel()
+ addActionListener(ActionListener)

MockView
- log: StringBuilder

+ MockView(StringBuilder)
+ enterGame()
+ getNewFile(): File
+ showGameInterface(World)
+ showError(String)
+ addKeyListener(KeyListener)
+ addActionListener(ActionListener)
+ getPlayerName(): String
+ getSpaceName(): String
+ getItemName(): String
+ getAttemptChoice(): String
+ getIntendedSpace(): String
+ addPlayer()
+ startGame()
+ setResult(String)
+ endGame(String)
+ configureSpaceClickListener(MouseAdapter)
+ configurePlayerClickListener()
+ resetFocus()

InfoPanel
- serialVersionUID: long
- preGamePanel: PreGamePanel
- gamePanel: GamePanel

+ InfoPanel(ReadOnlyWorld)
+ addActionListener(ActionListener)
+ addPlayer()
+ startGame()
+ setResult(String)

GameActionListener
- gameActions: Map<String, Runnable>

+ setGameActionMap(Map<String, Runnable>)
+ actionPerformed(ActionEvent)

KeyboardListener
- keyTypedMap: Map<Character, Runnable>
- keyPressedMap: Map<Integer, Runnable>
- keyReleasedMap: Map<Integer, Runnable>

+ KeyboardListener()
+ setKeyTypedMap(Map<Character, Runnable>)
+ setKeyPressedMap(Map<Integer, Runnable>)
+ setKeyReleasedMap(Map<Integer, Runnable>)
+ keyTyped(KeyEvent)
+ keyPressed(KeyEvent)
+ keyReleased(KeyEvent)

MainPanel
- serialVersionUID: long
- menuBar: JMenuBar
- gameMenu: JMenu
- newWorldItem: JMenuItem
- currentWorldItem: JMenuItem
- quitItem: JMenuItem
- worldPlaceholder: JLabel
- mapPanel: MapPanel
- infoPanel: InfoPanel

+ MainPanel(ReadOnlyWorld)
+ addActionListener(ActionListener)
+ showGameInterface()
+ addPlayer()
+ startGame()
+ setResult(String)
+ endGame(String)
+ configureSpaceClickListener(MouseAdapter)
+ configurePlayerClickListener()

<<interface>> View
+ enterGame()
+ getNewFile(): File
+ showGameInterface(World)
+ showError(String)
+ addKeyListener(KeyListener)
+ addActionListener(ActionListener)
+ getPlayerName(): String
+ getSpaceName(): String
+ getItemName(): String
+ getAttemptChoice(): String
+ getIntendedSpace(): String
+ addPlayer()
+ startGame()
+ setResult(String)
+ endGame(String)
+ configureSpaceClickListener(MouseAdapter)
+ configurePlayerClickListener()
+ resetFocus()

<<interface>> WorldCommand
+ execute(World, Appendable): String

GameController
- currentFile: File
- model: World
- view: View
- in: Readable
- out: Appendable
- turnLimit: int
- initialCommands: Map<String, Function<Scanner, WorldCommand>>
- commands: Map<String, Function<Scanner, WorldCommand>>
- humanCommands: Map<String, Function<Scanner, WorldCommand>>
- computerCommands: Map<String, Function<World, WorldCommand>>
- keyTypes: Map<Character, Runnable>
- gameActions: Map<String, Runnable>

+ GameController(World, View, File, int)
+ GameController(Readable, Appendable, int)
+ start(World)
+ getKeyTypes: Map<Character, Runnable>
+ getGameActions: Map<String, Runnable>
- configureKeyBoardListener()
- configureActionListener()
- configureSpaceClickListener()
- configurePlayerClickListener()
- selectNewWorld()
- selectCurrentWorld()
- exitProgram()
- addComputerPlayer()
- addHumanPlayer()
- startGame()
- checkAutomaticExecution()
- shouldGameContinue(): boolean
- movePlayer(int, int)
- pickUpItem()
- lookAround()
- makeAnAttempt()
- movePet()

MapPanel
- serialVersionUID: long
- model: ReadOnlyWorld
- mapLabel: JLabel
- layeredPane: JLayeredPane
- characters: Map<String, JLabel>

+ MapPanel(ReadOnlyWorld)
+ configureSpaceClickListener(MouseAdapter)
+ drawCharacter(String, int, int)
- drawTarget()
- drawPlayers()
+ refresh()

AddComputerPlayer
- name: String
- spaceName: String

+ AddComputerPlayer(Scanner, Appendable)
+ execute(World, Appendable): String

AddHumanPlayer
- name: String
- spaceName: String

+ AddHumanPlayer(Scanner, Appendable)
+ execute(World, Appendable): String

DisplayPlayerInformation
- playerName: String

+ DisplayPlayerInformation(Scanner, Appendable)
+ execute(World, Appendable): String

DisplaySpaceInformation
- spaceName: String

+ DisplaySpaceInformation(Scanner, Appendable)
+ execute(World, Appendable): String

GenerateMap
+ GenerateMap()
+ execute(World, Appendable): String

LookAround
+ LookAround()
+ execute(World, Appendable): String

MovePlayer
- spaceName: String

+ MovePlayer(Scanner, Appendable)
+ MovePlayer(World)
+ execute(World, Appendable): String

PickUpItem
- itemName: String

+ PickUpItem(Scanner, Appendable)
+ PickUpItem(World)
+ execute(World, Appendable): String

MakeAnAttempt
- itemName: String

+ MakeAnAttempt(Scanner, Appendable)
+ MakeAnAttempt(World)
+ execute(World, Appendable): String

MockModel
- log: StringBuilder
- name: String
- rows: int
- cols: int
- targetCharacter: TargetCharacter
- pet: Pet
- spaces: List<Space>
- players: List<Player>
- turn: Player
- turnCount: int
- canBeSeenByOthers: boolean

+ MockModel(StringBuilder, String, int, int, TargetCharacter, Pet, List<Space>, List<Player>, Player, boolean)
+ getName: String
+ getSize: int[]
+ getTargetCharacter: TargetCharacter
+ getPet: Pet
+ getSpaces: List<Space>
+ getSpace(String): Space
+ getSpace(int, int): Space
+ getPlayers: List<Player>
+ getPlayer(String): Player
+ displaySpaceInformation(String): String
+ displayPlayerInformation(String): String
+ getTargetCharacterPosition: int
+ getNeighbors(Space): List<Space>
+ addComputerPlayer(String, String)
+ addHumanPlayer(String, String)
+ moveTargetCharacter
+ resetTurn
+ getTurn: Player
+ nextTurn
+ getTurnCount: int
+ movePlayer(String)
+ pickupItem(String)
+ lookAround: String
+ movePet(String)
+ canBeSeenByOthers: boolean
+ makeAnAttempt(String): boolean
+ generateMap: BufferedImage

<<interface>> ReadOnlyWorld
+ getName: String
+ getSize: int[]
+ getTargetCharacter: TargetCharacter
+ getPet: Pet
+ getSpaces: List<Space>
+ getSpace(String): Space
+ getSpace(int, int): Space
+ getPlayers: List<Player>
+ getPlayer(String): Player
+ displaySpaceInformation(String): String
+ displayPlayerInformation(String): String
+ getTargetCharacterPosition: int
+ getNeighbors(Space): List<Space>
+ getTurn: Player
+ getTurnCount: int
+ canBeSeenByOthers: boolean
+ generateMap: BufferedImage

<<interface>> World
+ addComputerPlayer(String, String)
+ addHumanPlayer(String, String)
+ moveTargetCharacter
+ resetTurn
+ nextTurn
+ movePlayer(String)
+ pickupItem(String)
+ lookAround: String
+ movePet(String)
+ makeAnAttempt(String): boolean

MovePet
- spaceName: String

+ MovePet(Scanner, Appendable)
+ MovePet(World)
+ execute(World, Appendable): String

MyWorld
- name: String
- rows: int
- cols: int
- targetCharacter: TargetCharacter
- pet: Pet
- spaces: List<Space>
- players: List<Player>
- targetCharacterPosition: int
- turn: Player
- turnCount: int
- orderedSpaces: List<Space>
- separator: String

+ MyWorld(Readable)
- areSpacesOverlapping(Space, Space): boolean
- getLookAroundNeighborInformation(Space): String
- spacesDepthFirstTraversal(Space, List<Space>)
+ getName: String
+ getSize: int[]
+ getTargetCharacter: TargetCharacter
+ getPet: Pet
+ getSpaces: List<Space>
+ getSpace(String): Space
+ getSpace(int, int): Space
+ getPlayers: List<Player>
+ getPlayer(String): Player
+ displaySpaceInformation(String): String
+ displayPlayerInformation(String): String
+ getTargetCharacterPosition: int
+ getNeighbors(Space): List<Space>
+ addComputerPlayer(String, String)
+ addHumanPlayer(String, String)
+ moveTargetCharacter
+ resetTurn
+ getTurn: Player
+ nextTurn
+ getTurnCount: int
+ movePlayer(String)
+ pickupItem(String)
+ lookAround: String
+ movePet(String)
+ canBeSeenByOthers: boolean
+ makeAnAttempt(String): boolean
+ generateMap: BufferedImage

Driver
+ main(String[])

TargetCharacter
- name: String
- health: int

+ TargetCharacter(String, int)
+ getName: String
+ getHealth: int
+ reduceHealth(int)

<<interface>> Character
+ getName: String

Pet
- name: String
- space: Space

+ Pet(String)
+ getName: String
+ getSpace: Space
+ setSpace(Space)

ComputerControlledPlayer
- random: Random
- operationIndex: int
- neighborIndex: int
- itemIndex: int
- petIndex: int

+ ComputerControlledPlayer(String, Space)
+ ComputerControlledPlayer(String, Space, int, int, int, int)
+ getRandomOperation: String
+ getRandomNeighborName(List<Space>): String
+ getRandomItemName(List<Item>): String
+ getRandomSpaceName(List<Space>): String
+ getMostPowerfulItemName(List<Item>): String

HumanControlledPlayer
+ HumanControlledPlayer(String, Space)

Player *(abstract)*
name: String
space: Space
items: List<Item>
itemLimit: int

Player(String, Space)
+ getName: String
+ getSpace: Space
+ addItem(Item)
+ removeItem(Item)
+ getItems: List<Item>
+ getItem(String): Item
+ isNeighbor(Player): boolean
+ isSameSpace(Player): boolean

<<interface>> Space
+ getPosition: int[]
+ getName: String
+ getItems: List<Item>
+ getNeighbors: List<Space>
+ getItem(String): Item
+ addItem(Item)
+ removeItem(Item)
+ addNeighbor(Space)

<<interface>> Item
+ getName: String
+ getDamage: int

MyItem
- name: String
- damage: int

+ MyItem(String, int)
+ getName: String
+ getDamage: int

MySpace
- row1: int
- col1: int
- row2: int
- col2: int
- name: String
- items: List<Item>
- neighbors: List<Space>

+ MySpace(int, int, int, int, String)
+ getPosition: int[]
+ getName: String
+ getItems: List<Item>
+ getNeighbors: List<Space>
+ getItem(String): Item
+ addItem(Item)
+ removeItem(Item)
+ addNeighbor(Space)

Model Change Explanation

In Milestone 4 Preliminary Design, to make the model more adapted to the MVC design pattern, I primarily made these changes to my model:

(1) Separate a brand new ReadOnlyWorld interface from the original World interface, and make the World interface extends it. This is intended to provide an access to the model for the components of the View part.

(2) Add a new getSpace(int, int) method in the model. The method is for getting a space according to the coordinate of the mouse because the project requires players to move to other spaces using mouse click.

(3) Modify the generateMap() method. The world map in the GUI window is unclear, so the internal logic of the generateMap() needs to be modified.

Given that the other parts can be well isolated from the view and the controller, I didn't change them.

The sketch of the UI

AboutPanel
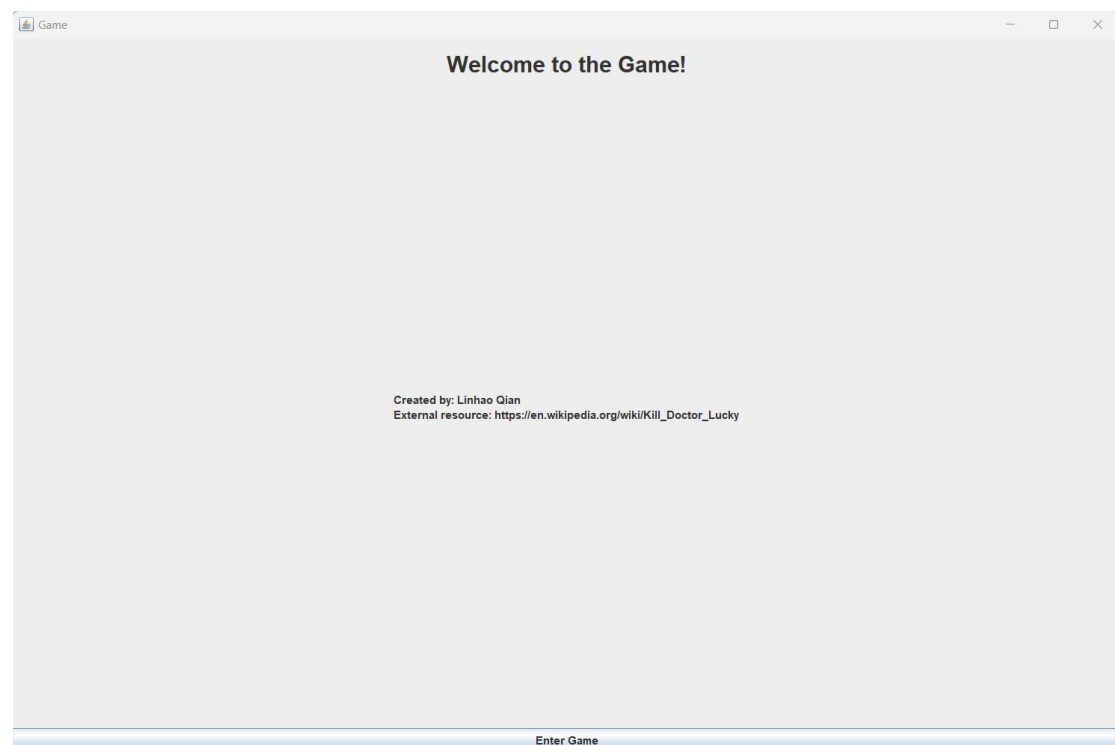
MainPanel (Initial screen)
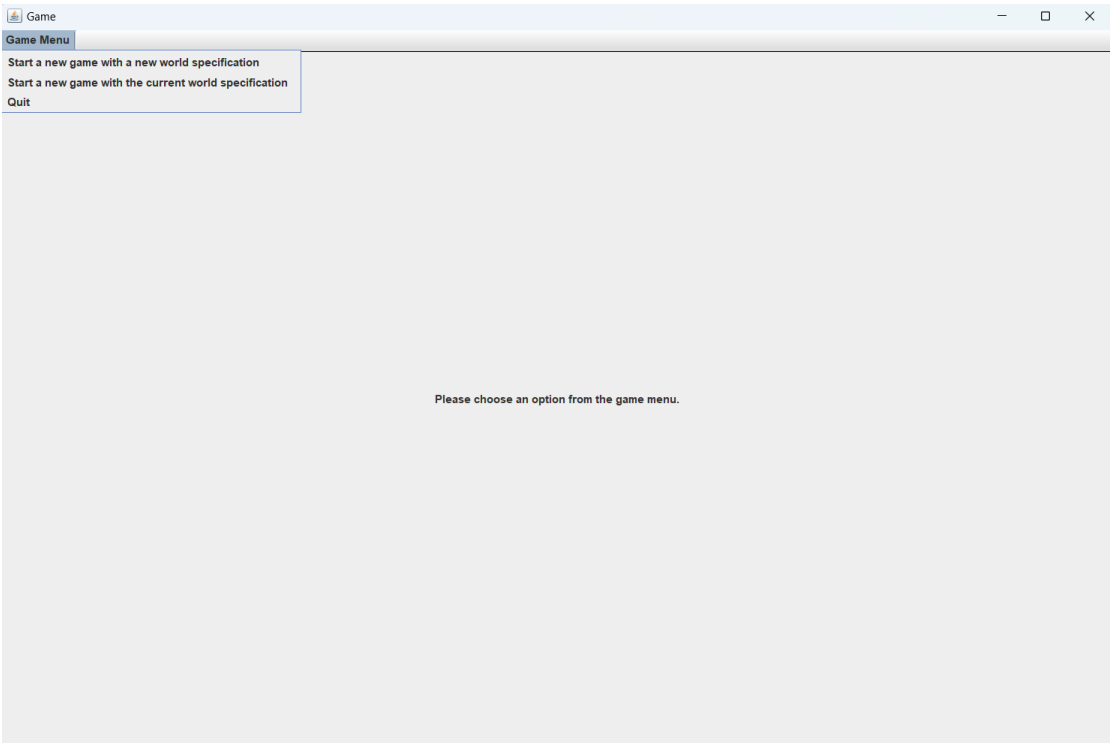
MainPanel (MapPanel & InfoPanel(PreGamePanel))
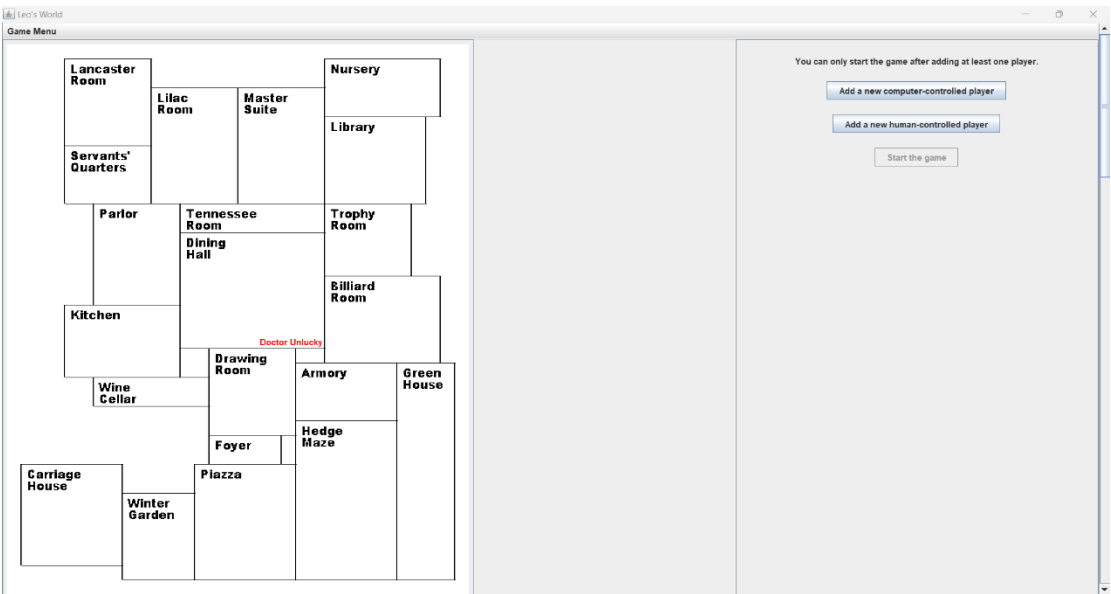
Add player dialog
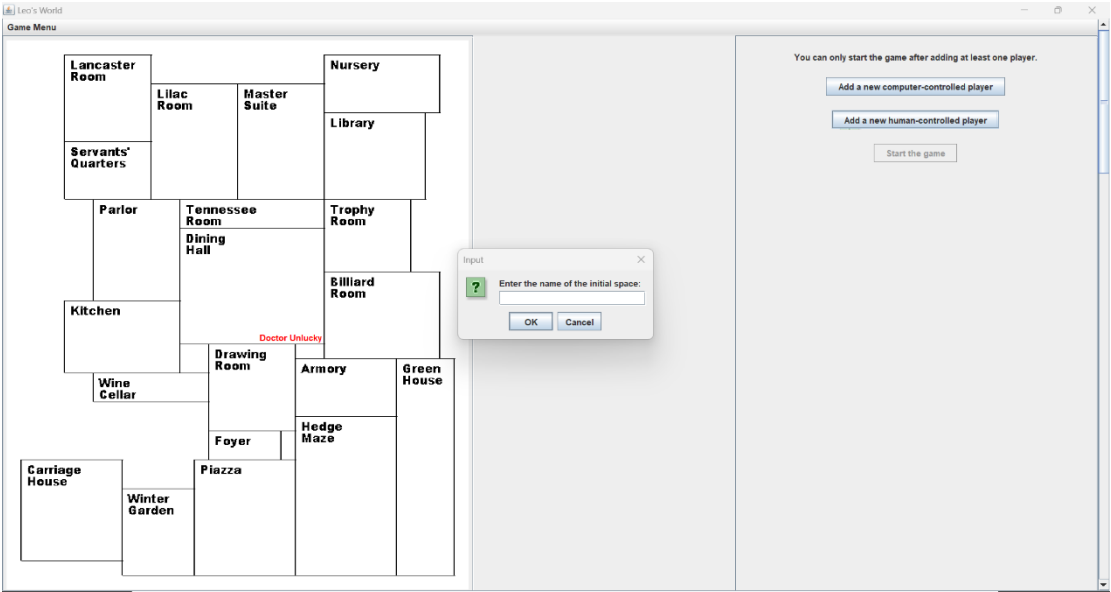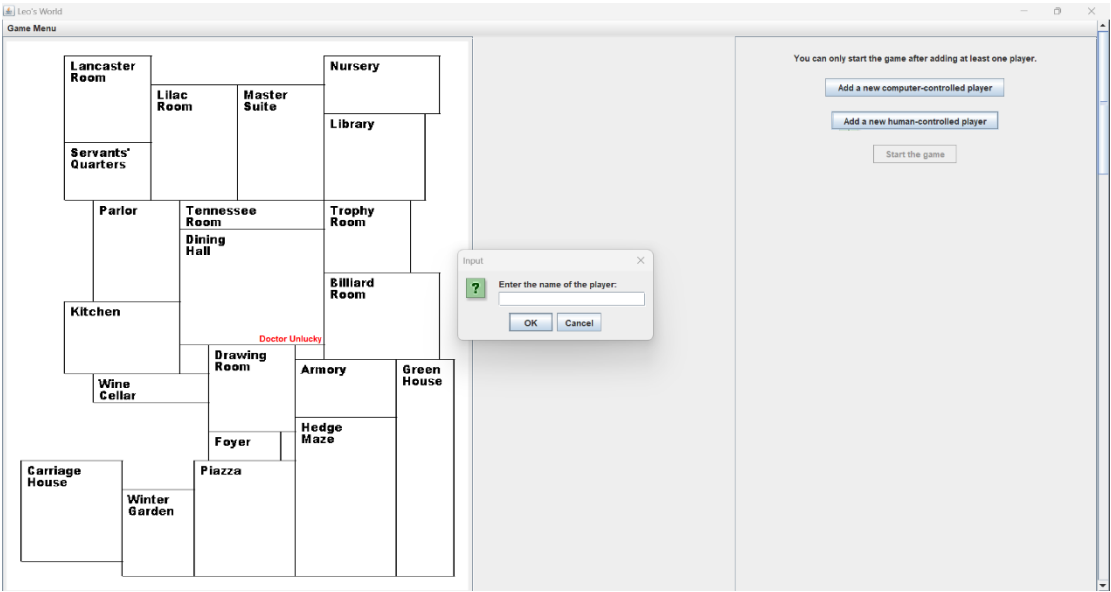
MainPanel (MapPanel & InfoPanel(GamePanel))
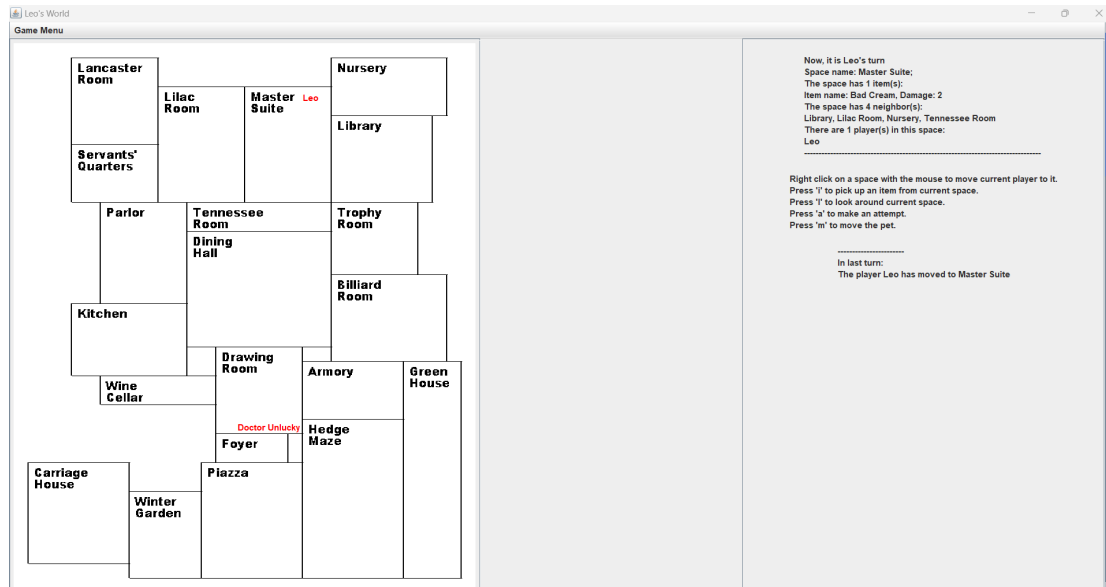
Testing Plan (for Milestone 4)

Tests for the Model (Most of the model logic have been tested in previous Milestone 1-3, so here I only include those newly-added functions' tests)

What is testing	Setup	Expected output
Get a space according to the mouse's coordinate	Initialize a world with several spaces, then use the getSpace(int x, int y) method to get one of the spaces	The corresponding space object is returned.
Get a space according to the mouse's coordinate, but there is no corresponding space where the mouse clicks	Initialize a world with several spaces, then call the getSpace(int x, int y) method with invalid x or y	A null space.
Adding multiple players to the game, with the total number exceeds 10	Initialize a world with several spaces, then add 11 players with addComputerPlayer and	When add the 11th player, an UnsupportedOperationException should be thrown out.

	addHumanPlayer methods	

Mock Model for controller tests

You can find my mock model details in the UML Diagram, which implements the World interface and uses a log field to record the input from the controller. The returned value of each method is a pre-defined value, making sure the controller can be tested isolation from the model

Mock View for controller tests

You can find my mock view details in the UML Diagram, which implements the View interface and uses a log field to record the input from the controller. The returned value of each method is a pre-defined value, making sure the controller can be tested isolation from the view.

Tests for the GameController

What is testing	Setup	Expected output
Tests whether the game correctly triggers the "Enter Game" functionality.	Default GameController setup with a mock model and view. Calls Enter Game action.	viewLog contains enterGame called.
Tests starting a new game with a new world specification.	Calls "Start a new game with a new world specification".	viewLog contains getNewFile, showGameInterface, addActionListener, configureSpaceClickListener, resetFocus.
Tests starting a new game with the current world specification.	Calls "Start a new game with the current world specification".	viewLog contains showGameInterface, addActionListener, configureSpaceClickListener, resetFocus.
Tests adding a new computer-controlled player.	Calls "Add a new computer-controlled player".	viewLog contains getPlayerName, getSpaceName, addPlayer, configurePlayerClickListener.
Tests adding a new	Calls "Add a new	viewLog contains getPlayerName,

human-controlled player.	human-controlled player".	getSpaceName, addPlayer, configurePlayerClickListener.
Tests initializing the game logic and starting a new game.	Calls "Start the game".	modelLog contains "The turn has been reset". viewLog contains startGame, resetFocus.
Tests moving a player to a valid space.	Calls movePlayer(1, 1) with valid coordinates.	modelLog contains "get space according to coordinates, move player to xxx". viewLog contains setResult with movement details, resetFocus.
Tests moving a player to an invalid space.	Calls movePlayer(0, 0) with invalid coordinates.	modelLog contains get space according to coordinates. viewLog contains "Please choose a valid space!".
Tests the player's ability to pick up an item in the current space.	Calls key handler for 'i'.	viewLog contains getItemName, setResult with item details, resetFocus. modelLog contains pick up item, next turn.
Tests the player's ability to look around in the current space.	Calls key handler for 'l'.	viewLog contains setResult with look-around details, resetFocus. modelLog contains next turn.
Tests the player's ability to make a successful attack on the target character.	Calls key handler for 'a'. Mock model is configured for success.	viewLog contains getAttemptChoice, setResult with attack success, resetFocus. modelLog contains make an attempt, next turn.
Tests the player's attempt failing due to being seen by others.	Creates a mock model where attacks are visible. Calls key handler for 'a'.	viewLog contains getAttemptChoice, setResult with failure message, resetFocus. modelLog contains make an attempt, next turn.
Tests moving the pet to a new space.	Calls key handler for 'm'.	viewLog contains getIntendedSpace, setResult with pet movement details, resetFocus. modelLog contains move pet, next turn.
Tests the automatic movement of a computer-controlled player.	Configures a computer-controlled player and calls "Start the game".	modelLog contains move player to, next turn. viewLog contains setResult with movement details, resetFocus.
Tests the automatic pickup of an item by a computer-	Configures a computer-controlled player. Calls "Start the game".	modelLog contains pick up item, next turn. viewLog contains setResult with pickup details,

		resetFocus.
Tests the automatic "look around" action by a computer-controlled player.	Configures a computer-controlled player. Calls "Start the game".	modelLog contains next turn. viewLog contains setResult with look-around details, resetFocus.
Tests the automatic movement of the pet during the game.	Configures a computer-controlled player and calls "Start the game".	modelLog contains move pet, next turn. viewLog contains setResult with pet movement details, resetFocus.
Tests that the computer-controlled player selects the most powerful item for an attack.	Configures a computer-controlled player with multiple items. Calls "Start the game".	modelLog contains make an attempt with (most powerful item), next turn. viewLog contains setResult with attack details, resetFocus.
Tests the winning condition where a human player kills the target character.	Configures a model where the target character is weak. Calls key handler for 'a'.	viewLog contains endGame with victory message. modelLog contains make an attempt, next turn.
Tests the winning condition where a computer-controlled player kills the target character.	Configures a computer-controlled player and a weak target character. Calls "Start the game".	viewLog contains endGame with victory message. modelLog contains make an attempt, next turn.
Tests the game ending after reaching the maximum number of turns.	Configures a model with a 0 turnLimit. Calls "Start the game".	viewLog contains endGame with maximum-turns message.

Testing Plan (for Milestone 1 – 3)

Testing design for TargetCharacter

Testing construction	Input	Expected Value
Constructor disallows null name	TargetCharacter(null, 50)	IllegalArgumentException
Constructor disallows empty name	TargetCharacter("", 50)	IllegalArgumentException

Constructor disallows non-positive health	TargetCharacter("doctor", 0)	IllegalArgumentException

Testing getName()	Input	Expected Value
TargetCharacter with normal name	TargetCharacter("Leo", 50)	"Leo"

Testing getHealth()	Input	Expected Value
TargetCharacter with positive health	TargetCharacter("Leo", 50)	50

Testing reduceHealth(int damage)	Input	Parameter	Actual Testing
Reduce positive health value	TargetCharacter("Leo", 50)	3	assertEquals(getHealth(), 47)
Test invalid damage	TargetCharacter("Leo", 50)	-3	IllegalArgumentException

Testing design for Pet

Testing construction	Input	Expected Value
Constructor disallows null name	Pet(null)	IllegalArgumentException
Constructor disallows empty name	Pet("")	IllegalArgumentException

Testing getName()	Input	Expected Value
Pet with normal name	Pet("cat")	"cat"

Testing setSpace(Space space) and getSpace()	Input	Expected Value
Set a normal space and get it	setSpace(new MySpace(0, 0, 3, 3, "Kitchen")); getSpace();	new MySpace(0, 0, 3, 3, "Kitchen")
Disallow set a null space	setSpace(null);	IllegalArgumentException

Testing design for Player

Testing construction	Input	Expected Value
Test Computer Constructor With Null Name	ComputerControlledPlayer (null, new MySpace(0, 0, 3, 3, "Kitchen"))	IllegalArgumentException
Test Human Constructor With Null Name	HumanControlledPlayer (null, new MySpace(0, 0, 3, 3, "Kitchen"))	IllegalArgumentException
Test Computer Constructor With Empty Name	ComputerControlledPlayer ("", new MySpace(0, 0, 3, 3, "Kitchen"))	IllegalArgumentException
Test Human Constructor With Empty Name	HumanControlledPlayer ("", new MySpace(0, 0, 3, 3, "Kitchen"))	IllegalArgumentException
Test Computer Constructor With Null Space	ComputerControlledPlayer ("Leo", null)	IllegalArgumentException
Test Human Constructor With Null Space	HumanControlledPlayer ("Leo", null)	IllegalArgumentException

Testing getName()	Input	Expected Value
HumanControlledPlayer with normal name	HumanControlledPlayer ("Leo", new MySpace(0, 0, 3, 3, "Kitchen"))	"Leo"
ComputerControlledPlayer with normal name	ComputerControlledPlayer ("Leo", new MySpace(0, 0, 3, 3, "Kitchen"))	"Leo"

Testing addItem(Item item), getItem(String itemName), and getItems()	Input	Expected Value
Add and get an item	addItem(new MyItem("Revolver", 3)); getItems().get(0); getItem("Revolver")	new MyItem("Revolver", 3)

Testing removeItem(Item item)	Input	Expected Value
Remove an item	Item = new MyItem("Revolver", 3) addItem(item); getItems().size(); removeItem(item);	1; 0;

	getItems().size();	

Testing setSpace(Space space) and getSpace()	Operation	Expected Value
Set and get a space	setSpace(new MySpace(0, 0, 3, 3, "Kitchen")); getSpace()	MySpace(0, 0, 3, 3, "Kitchen")

Testing isNeighbor ()	Testing ideas
Determine whether a player is the neighbor of current player	Add a player to the neighbor space of current player, then test if the new added player is current player's neighbor.

Testing isSameSpace ()	Testing ideas
Determine whether a player is in the same space of current player	Add a player to the current space of current player, then test if the new added player is in the same space of current player.

Testing getMostPowerful ItemName ()	Input	Expected Value
Get the most powerful item name	addItem(new MyItem("Revolver", 3)) addItem(new MyItem("Knife", 2))	"Revolver"

Testing design for MyItem

Testing construction	Input	Expected Value
Constructor disallows empty name	MyItem("", 3)	IllegalArgumentException
Constructor disallows null name	MyItem(null, 3)	IllegalArgumentException
Constructor disallows non-positive damage	MyItem("Revolver", 0)	IllegalArgumentException

Testing getName()	Input	Expected Value
Item with normal name	MyItem("Revolver", 0, 3)	"Revolver"

Testing getDamage()	Input	Expected Value
Item with positive damage	MyItem("Revolver", 0, 3)	3

Testing design for MySpace

Testing construction	Input	Expected Value
Constructor disallows negative row1	MySpace(-1, 0, 3, 3, "Kitchen")	IllegalArgumentException
Constructor disallows negative col1	MySpace(0, -1, 3, 3, "Kitchen")	IllegalArgumentException
Constructor disallows the value of row2 to be less than the value of row1	MySpace(0, 0, -3, 3, "Kitchen")	IllegalArgumentException
Constructor disallows the value of col2 to be less than the value of col1	MySpace(0, 0, 3, -3, "Kitchen")	IllegalArgumentException
Constructor disallows empty name	MySpace(0, 0, 3, 3, "")	IllegalArgumentException
Constructor disallows null name	MySpace(0, 0, 3, 3, null)	IllegalArgumentException

Testing getPosition()	Input	Expected Value
Space with correct position	MySpace(0, 0, 3, 3, "Kitchen")	new int[]{0, 0, 3, 3}

Testing getName()	Input	Expected Value
Space with normal name	MySpace(0, 0, 3, 3, "Kitchen")	"Kitchen"

Testing addItem(Item item), removeItem(Item item), getItem(String itemName), and getItems()	Operation	Actual Testing

Test multiple methods	Item item = new Item("Revolver", 0, 3);	addItem(item);
		assertTrue(getItems().get(0).equals(item));
		assertEquals(getItem("Revolver", item)
		removeItem(item);
		assertEquals(getItems().size, 0);

Testing	Operation	Actual Testing

addNeighbor(Space space) and getNeighbors()		
Add a neighbor and get the neighbors list	Space space = new MySpace(0, 0, 3, 3, "Kitchen");	Space newSpace = new MySpace(0, 4, 3, 6, "Parlor"); space.addNeighbor(newSpace); assertEquals(getNeighbors ().get(0), newSpace)

Testing design for MyWorld (i.e., the model testing design)

Testing reading world.txt file	Testing ideas
Read the txt file and generate the world	The MyWorld toString() method can print the world's information, so we can compare the printed information with the expected information after generating the MyWorld object.

Testing invalid world description	Testing ideas
The txt file has an invalid world size	Set the world size in the txt file to 36 × 0, then read the file. It should throw an IllegalArgumentException

Testing invalid space	Testing ideas
The txt file has overlapping sapces	Set two overlapping sapces in the txt file, then read the file. It should throw an IllegalArgumentException

Testing invalid item	Testing ideas
The txt file has an item with an space index out of bounds	Set an item's space index to an non-existent index in the txt file, then read the file. It should throw an IllegalArgumentException

Testing getNeighbors() and displaySpaceInformation() under different neighboring conditions	Testing ideas
Test a space with no neighbors	Make one space non adjacent to other spaces in the txt file, then read the file and get that space. Use assertEquals() to check if its getNeighbors()

	method returns a 0-length list, and check if its displaySpaceInformation() method returns the correct string.
Test a space with one neighbor	Find one space with only one neighbor. Use assertEquals() to check if its getNeighbors() method returns a 1-length list, and check if its displaySpaceInformation() method returns the correct string.
Test a space with multiple neighbors	Find one space with multiple neighbors. Use assertEquals() to check if its getNeighbors() method returns correct space list, and check if its displaySpaceInformation() method returns the correct string.

Testing displaySpaceInformation() under other conditions	Testing ideas
Test a space with no Items	Find one space with no items. Use assertEquals() to check if its displaySpaceInformation() method returns the correct string.
Test a space with one item	Find one space with one item. Use assertEquals() to check if its displaySpaceInformation() method returns the correct string.
Test a space with players	Add a player to a specified space. Use assertEquals() to check if its displaySpaceInformation() method returns the correct string.

Testing the start space of the target character	Testing ideas
The target character should start from space 0.	The target character should start from space 0, so we can just call getTargetCharacterPosition() method at the beginning to check if it returns 0.

Testing moving target character	Testing ideas
The target character can move from space 0 to space 1.	Call moveTargetCharacter() method once. Check if getTargetCharacterPosition() returns 1.
The target character can move multiple times.	Call moveTargetCharacter() method 10 times. Check if getTargetCharacterPosition() returns 11.
The target character can move from the last room in the index	Set x equals to the length of the space list. Call moveTargetCharacter() method x times. Check

list to room 0.	if getTargetCharacterPosition() returns 0.

Testing getting players	Testing ideas
Get the wrold's player list	The world should have an empty player list at the beginning, so getPlayers() returns an empty list.

Testing adding players	Testing ideas
Add human players	Add 2 human players, then use assertEquals() and assertSame() to test if they are the correct players.
Add computer players	Add 2 computer players, then use assertEquals() and assertSame() to test if they are the correct players.

Testing displayPlayerInformation()	Testing ideas
Display human player information	Add 1 human player carrying 1 item and another human player carrying 0 item, then use assertEquals() to check if the displayPlayerInformation() method returns the correct strings for both of them.
Display computer player information	Add 1 computer player carrying 1 item and another computer player carrying 0 item, then use assertEquals() to check if the displayPlayerInformation() method returns the correct strings for both of them.

Testing taking turns	Testing ideas
The players in the player list should take turns to play the game	Add 2 players, and reset the turn. Use assertSame() to check if the getTurn() method returns the first player. Then call nextTurn(), check if getTurn() method returns the second player.

Testing player moving around	Testing ideas
The players can move to neighboring spaces	Add 1 human player and 1 computer player, then move them to one of their neighboring space, respectively. Use assertSame() to check if they are in the expected spaces.
The players cannot move to non-neighboring spaces	Move a player to a non-neighboring space. It should throw an IllegalArgumentException.

Testing player picking up item	Testing ideas
The players can pick up item in current space when their item number doesn't reach the limit.	Add a player to a space with 2 items, then make the player pick up the 2 items. Use assertSame() to check if the player carry the expected items.
The players cannot pick up items that doesn't exist in current space	Make a player pick up a non-exist item. It should throw an IllegalArgumentException.
The players cannot pick up items when they have already carried enough items	Make a player pick up an item after their item number reached the limit. It should throw an UnsupportedOperationException.

Testing player looking around	Testing ideas
The player is in the initial space with the target character and the pet.	Add a player to the initial space. Use assertEquals() to check if the lookAround method returns the expected information (including target character and pet information).
The player is in a space which has a neighbor where the pet stays.	Add a player to a space which has a neighbor where the pet stays. Use assertEquals() to check if the lookAround method returns the expected information (the pet-occupied space should not reveal its information).
The player is in a space which doesn't have any items.	Add a player to a space which doesn't have any items. Use assertEquals() to check if the lookAround method returns the expected information (the item information should be correct).

Testing the start space of the pet	Testing ideas
The pet should start from space 0.	The pet should start from space 0, so we can just call assertSame() method to test if it stays in the correct space.

Testing moving pet	Testing ideas
The pet can be moved to a specified space	Move the pet to a specified space. Use assertSame() to check if it is in the expected space.

Testing pet wandering	Testing ideas
The pet should move with every turn following a depth-first traversal of the spaces in the	Let the game go through 4-5 turns. After each turn, check the position of the pet to confirm if it has entered each space according to the depth

world.	first rule.

Testing if the current player can be seen by others	Testing ideas
A player cannot be seen by its neighbor if the pet is in the same space.	Add a player A to the space where the pet stays, then add another player B to its neighbor. Player B shouldn't be able to see player A.
A player can be seen by other players in the same space, even if the pet is in the same space.	Add two players to the space where the pet stays. The two players should be able to see each other.
A player cannot be seen by others if there is no other player staying in the same space or neighbor spaces.	Add a player to a space which doesn't have any other player as well as neighbor players. The player shouldn't be seen by any others.

Testing computer player automatic make an attempt	Testing ideas
When a computer player stays with the target character in the same space, the player should automatically make an attempt with the most powerful item if no one can see he/she.	Add a computer player to the space where the target character stays. Make sure the player has some items and cannot be seen by others. The player should then automatically make an attempt with the most powerful item.

Testing making an attempt	Testing ideas
A computer player makes an attempt with an item successfully.	Add a computer player to the space where the target character stays. Make sure the player has some items and cannot be seen by others. The player should then make an attempt successfully.
A human player makes an attempt with an item successfully.	Add a human player to the space where the target character stays. Make sure the player has some items and cannot be seen by others. Make the player attack the target character. The player should then make an attempt successfully.
A computer player fails to make an attempt because there are other players in the same space.	Add a computer player to the space where the target character stays. Then add another player to the same space. Make the player attack the target character. The player should then fail to make an attempt.
A human player fails to make an attempt because there are other players in the same space.	Add a human player to the space where the target character stays. Then add another player to the same space. Make the player attack the target

	character. The player should then fail to make an attempt.
A computer player makes an attempt successfully with pet's help.	Add a computer player to the space where the target character stays. Make sure the player has a neighbor player and the pet is in that neighbor space. Make the player attack the target character. The player should then make an attempt successfully.
A human player makes an attempt successfully with pet's help.	Add a human player to the space where the target character stays. Make sure the player has a neighbor player and the pet is in that neighbor space. Make the player attack the target character. The player should then make an attempt successfully.
A computer player fails to make an attempt because there are other players in the neighbor space without pet.	Add a computer player to the space where the target character stays. Then add another player to a neighbor space without pet. Make the player attack the target character. The player should then fail to make an attempt.
A human player fails to make an attempt because there are other players in the neighbor space without pet.	Add a human player to the space where the target character stays. Then add another player to a neighbor space without pet. Make the player attack the target character. The player should then fail to make an attempt.
A computer player makes an attempt without any item successfully.	Add a computer player to the space where the target character stays. Make sure the player has no item and cannot be seen by others. The player should then make an attempt successfully.
A human player makes an attempt without any item successfully.	Add a human player to the space where the target character stays. Make sure the player has no item and cannot be seen by others. Make the player attack the target character. The player should then make an attempt successfully.
A player cannot use pokeEyes if the player has at least 1 item.	Add a player to the space where the target character stays. Make sure the player has at least 1 item and cannot be seen by others. Make the player attack the target character by poking him in the eye. This should throw an IllegalArgumentException.

Testing design for GameController

Testing start()	Testing ideas

| Start the game | Use a mock model to test whether the game can successfully complete a run |

Testing AddComputerPlayer	**Testing ideas**
Add a computer-controlled player	Use a mock model to test whether a computer controlled player is successfully added to the game after execute()

Testing AddHumanPlayer	**Testing ideas**
Add a humancontrolled player	Use a mock model to test whether a human controlled player is successfully added to the game after execute()

Testing DisplayPlayerInformation	**Testing ideas**
Display a player's information	Use a mock model to test whether a player's information is correctly displayed after execute()

Testing DisplaySpaceInformation	**Testing ideas**
Display a space's information	Use a mock model to test whether a space's information is correctly displayed after execute()

Testing GenerateMap	**Testing ideas**
Generate the world map	Use a mock model to test whether the map is successfully generated after execute()

Testing LookAround	**Testing ideas**
Display the neighboring information of the space a computer player is currently occupying	Use a mock model to test whether the current computer player's neighboring information is correctly displayed after execute()
Display the neighboring information of the space a human player is currently occupying	Use a mock model to test whether the current human player's neighboring information is correctly displayed after execute()

Testing MovePlayer	**Testing ideas**
Move a computer player from	Use a mock model to test whether the computer

current space to a neighbor space	player moves to expected space after execute()
Move a human player from current space to a neighbor space	Use a mock model to test whether the human player moves to expected space after execute()

Testing PickUpItem	Testing ideas
A computer player pick up an item from current space	Use a mock model to test whether the chosen item is successfully picked up by the computer player after execute()
A human player pick up an item from current space	Use a mock model to test whether the chosen item is successfully picked up by the human player after execute()

Testing movePet	Testing ideas
A computer player move the pet to a specified space	Use a mock model to test if the pet is in the specified space after execute()
A human player move the pet to a specified space	Use a mock model to test if the pet is in the specified space after execute()

Testing makeAnAttempt	Testing ideas
Make an attempt on the target character's life successfully	Use a mock model to test if the target character's life is reduced after execute()
Fail to make an attempt on the target character's life	Use a mock model to test if the attack fails after execute()
A computer player automatically makes an attempt with the most powerful item	Use a mock model to test if the computer player automatically makes an attempt with the most powerful item after execute()

Testing Human player win	Testing ideas
A human player kills the target character and win the game	Use a mock model to let a human player kills the target character, and test whether the printed information shows that the human player win

Testing Computer player win	Testing ideas
A computer player kills the target character and win the game	Use a mock model to let a computer player kills the target character, and test whether the printed information shows that the computer player win

Testing target character escapes	Testing ideas
The maximum number of turns is reached in which case the target character escapes and runs away to live another day and nobody wins	Use a mock model to play the game, and make sure the target character is not killed until the game ends. Test whether the printed information shows that it's a tie game