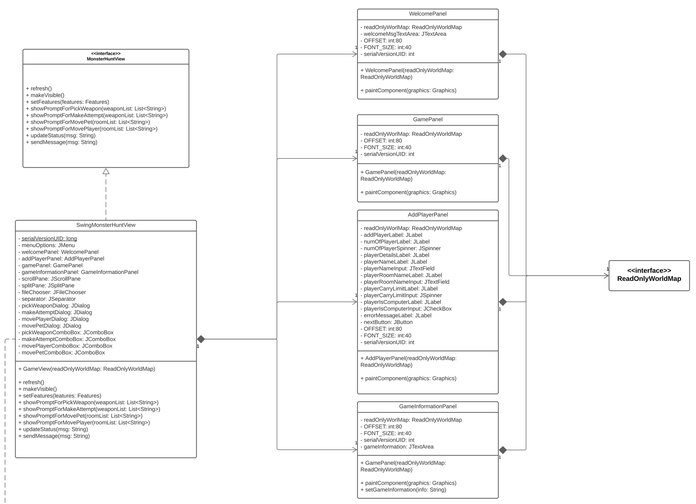
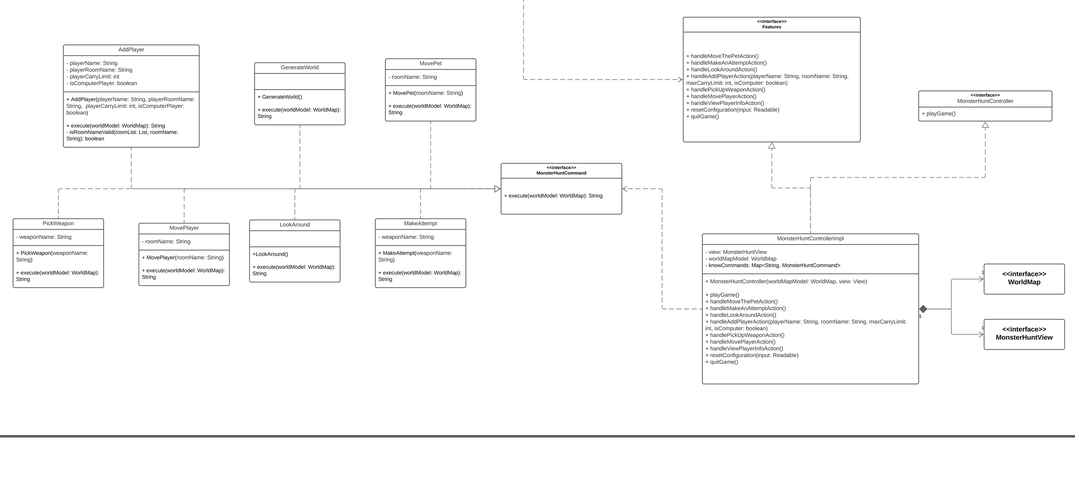
Milestone 4
Prilliminary
Design

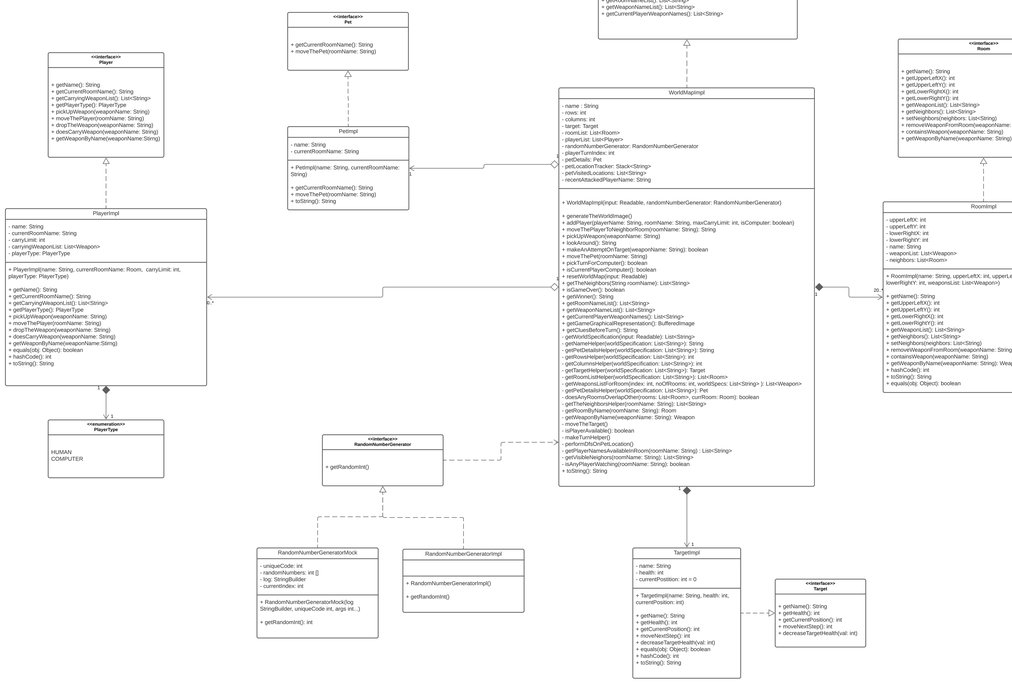
View



Controller

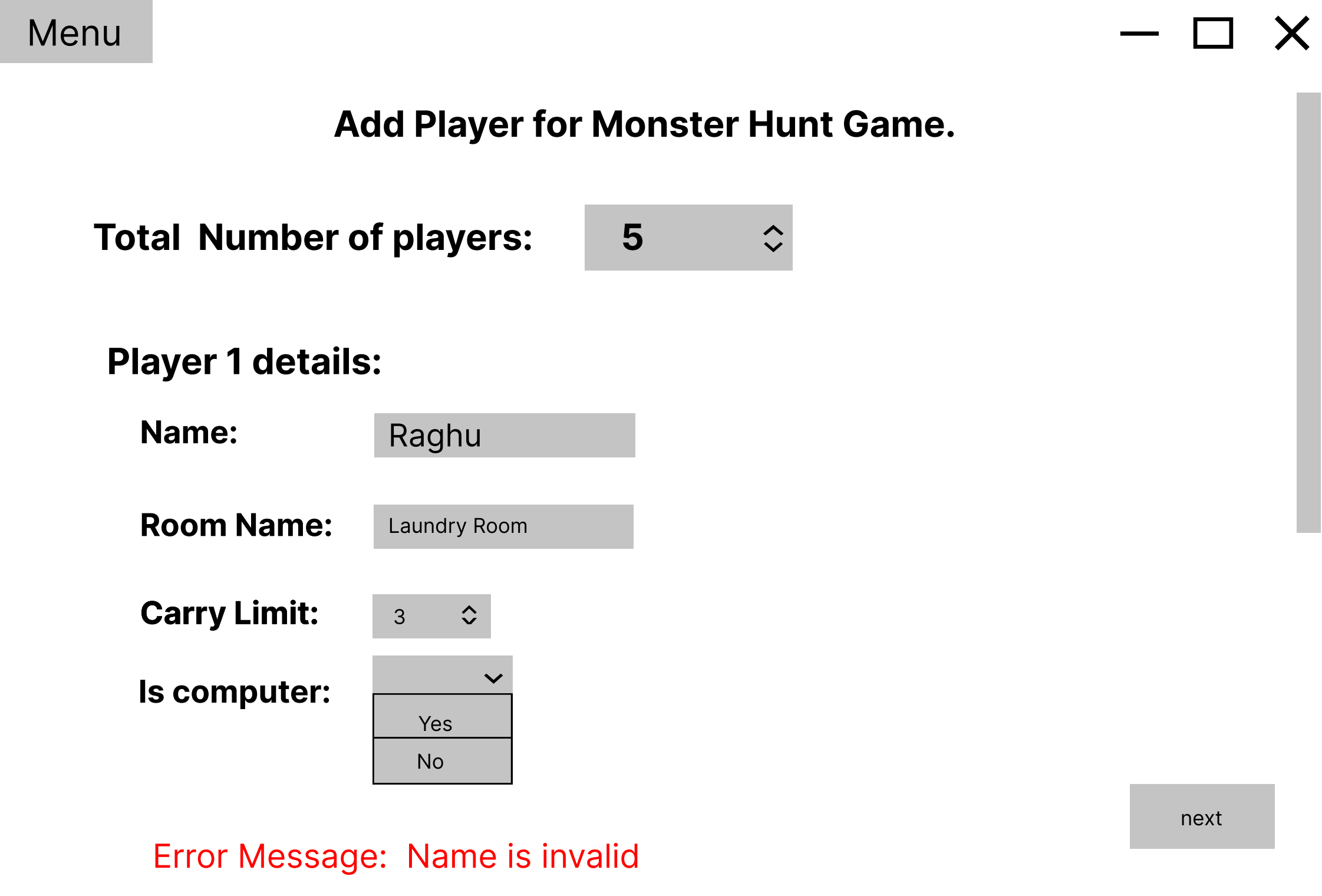


Model

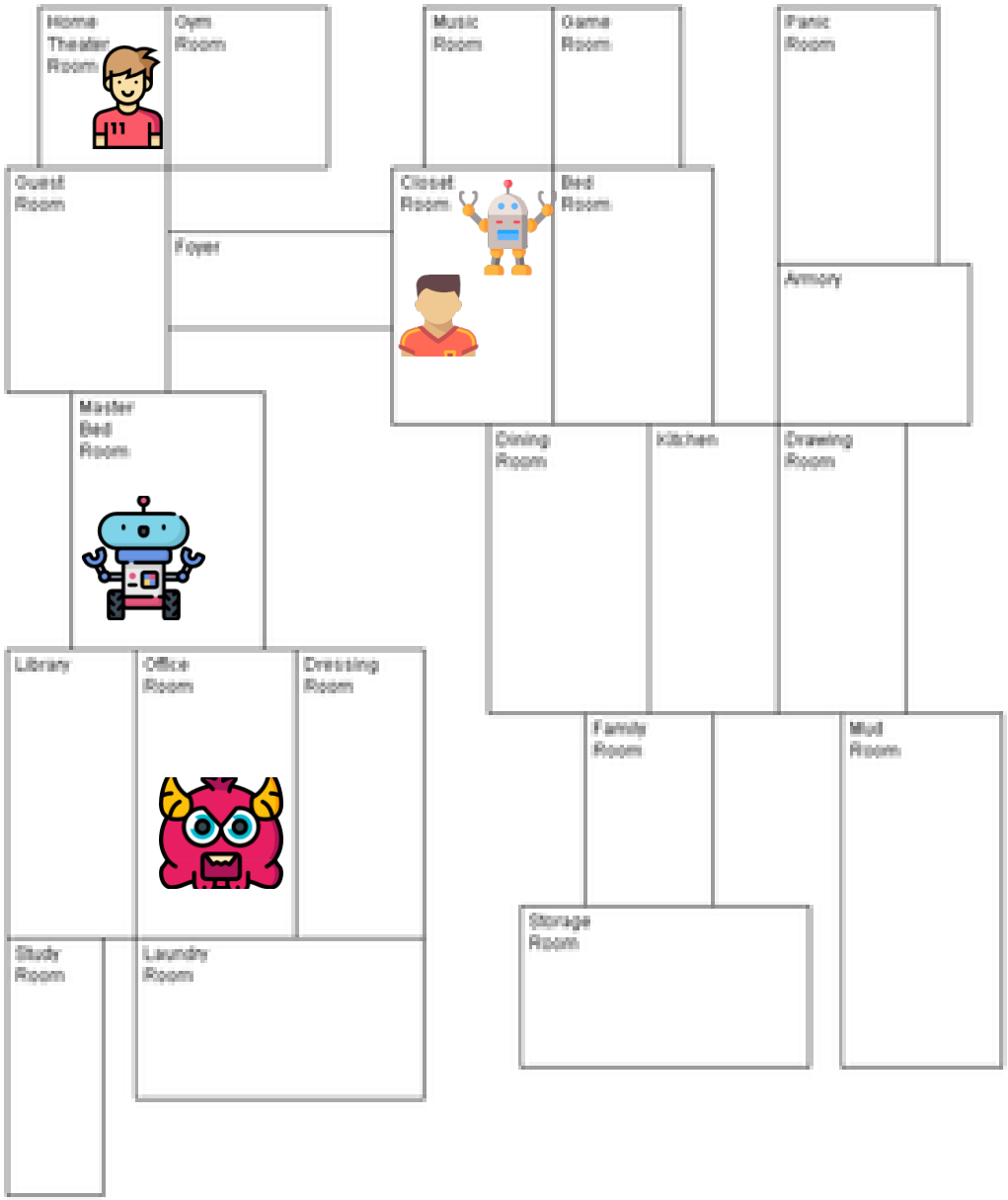


Welcome to Monster Hunt Game!

Credits :
Nandhini Ilangilli.
Raghunath Reddy Arava.


Menu
Add Player for Monster Hunt Game.
Total Number of players:
5
Player 1 details:
Name:
Raghu
Room Name:
Laundry Room
Carry Limit:
3
Is computer:
Yes
No
next
Error Message: Name is invalid

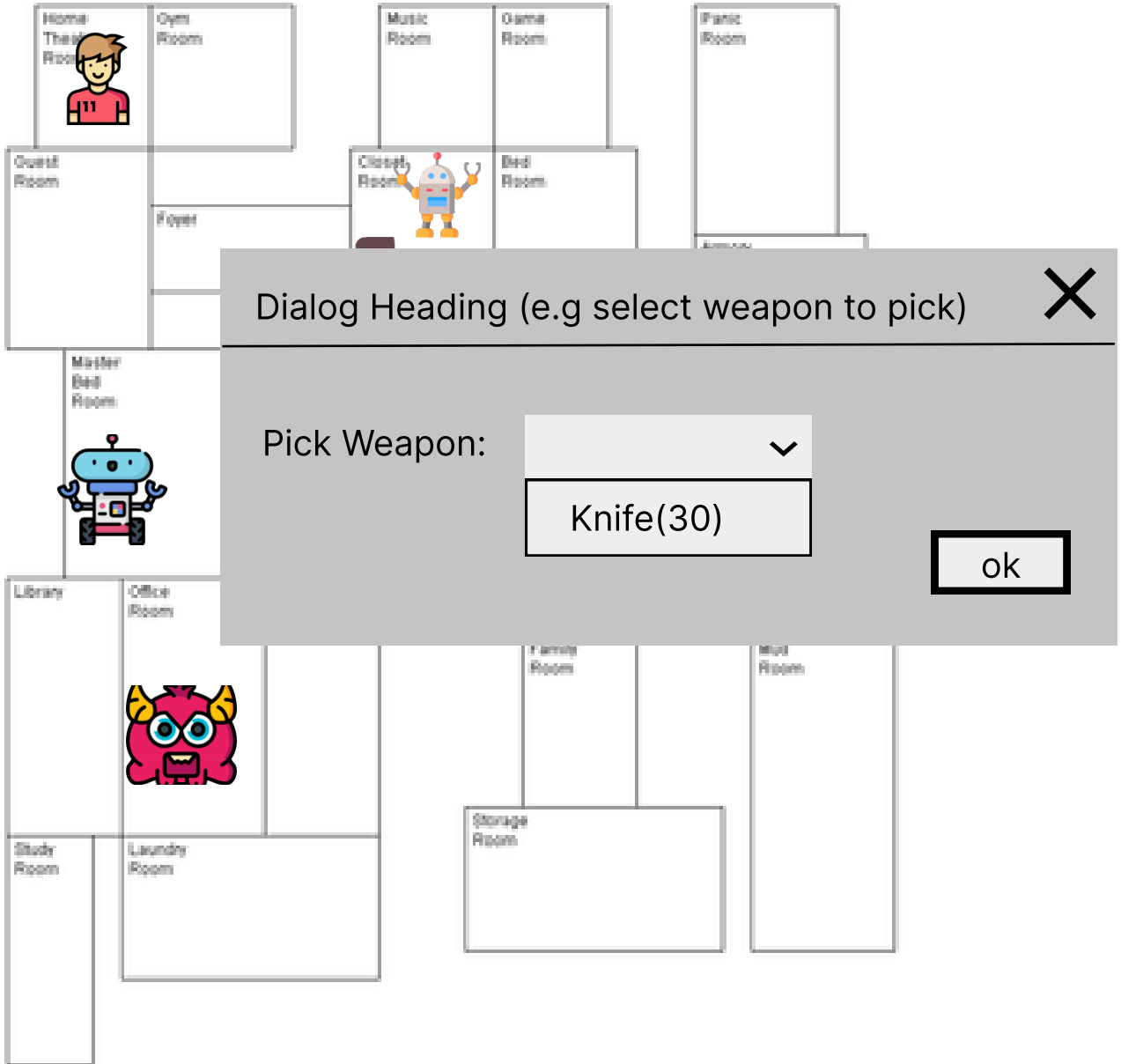
Menu



Current turn Details:

Player Name: Jhon
Room Information:
Name: Closet Room
Items:
[knife(20)]
Players:
[Jhon, computer1]
Is Pet available: false

Menu



Current turn Details:

Player Name: Jhon
Room Information:
Name: Closet Room
Items:
[knife(20)]
Players:
[Jhon, computer1]
Is Pet available: false

Milestone 4 Test Cases:

WorldMap model test cases:

Testing for WorldMapImpl class:

Write a one-paragraph explanation in your design document that summarizes any of the changes that you needed to make the model to separate it from the text-based controller.

The major changes for the model in order to support the view based game is that added a new read only interface called ("**ReadOnlyWorldMap**") which has methods to get the graphical representation of the current world and also one more method called 'getCluesBeforeTurn()' which will provide information regarding the current player name and the room information of current player location. There is one more method added to the WorldMap interface which is used to reset the world configuration by taking the input as Readable.

Note:

Here considering a helper method named `worldMap()` which will give an expected WorldMap instance.

Here `worldSpec` parameter is string builder which has information related to the world.

Testing WorldMapImpl class	Input	Expected Value
Get clues before each turn. (human player) ` getCluesBeforeTurn()`	worldMap(worldSpec) getCluesBeforeTurn()	Current Turn Details: Player Name: Raghu Player Location: Kitchen Room Details: Name: Kitchen Items: [Knife, Fork] Neighbors: [Drawing Room] Players: [Raghu, Computer 1] Is pet available: False
Get clues before each turn. (pet available, human player) ` getCluesBeforeTurn()`	worldMap(worldSpec) getCluesBeforeTurn()	Current Turn Details: Player Name: Raghu Player Location: Foyer Room Details: Name: Foyer Items: [] Neighbors: [Drawing Room] Players: [Raghu] Is pet available: True
Get clues before each turn. (pet available, computer player) ` getCluesBeforeTurn()`	worldMap(worldSpec) getCluesBeforeTurn()	Current Turn Details: Player Name: Comp 1 Player Location: Kitchen Room Details: Name: Kitchen Items: [Knife, Fork] Neighbors: [Drawing Room] Players: [Raghu, Comp 1] Is pet available: True

Testing WorldMapImpl class	Input	Expected Value
Get clues before each turn. (pet not available, computer player) ` getCluesBeforeTurn()`	worldMap(worldSpec) getCluesBeforeTurn()	Current Turn Details: Player Name: Comp 2 Player Location: Foyer Room Details: Name: Foyer Items: [] Neighbors: [] Players: [Comp 2] Is pet available: False
Get game graphical representation (Buffered Image). `getGameGraphicalRepresentation()`	worldMap(worldSpec) getGameGraphicalRepresentation()	The buffered image should be returned with all the world graphical representation.
Get game graphical representation (Buffered Image, error while generating image). `getGameGraphicalRepresentation()`	worldMap(worldSpec) getGameGraphicalRepresentation()	IllegalStateException

Testing WorldMapImpl class	Input	Expected Value
Reset the world specification. (Input is invalid). `resetWorldMap(input)`	worldMap(worldSpec) resetWorldMap(null) input - Readable	IllegalArgumentException
Reset the world specification. (Input is valid but details are incorrect). `resetWorldMap(input)`	worldMap(worldSpec) resetWorldMap(input) input - Readable	IllegalArgumentException
Reset the world specification. (Input is valid). `resetWorldMap(input)`	worldMap(worldSpec) resetWorldMap(input) input - Readable	Should reset the model according to the new world specifications.

Testing WorldMapImpl class	Input	Expected Value
Is game over or not. (Game over, target health is zero). `isGameOver()`	worldMap(worldSpec) isGameOver()	True
Is game over or not. (Game not over, target health is greater than zero). `isGameOver()`	worldMap(worldSpec) isGameOver()	False
Get winner of the game (Game is over, human player wins) `getWinner()`	worldMap(worldSpec) getWinner()	“Raghu”

Testing WorldMapImpl class	Input	Expected Value
Get winner of the game (Game is not over) `getWinner()`	worldMap(worldSpec) getWinner()	“”
Get winner of the game (Game is over, computer player wins) `getWinner()`	worldMap(worldSpec) getWinner()	“Computer 1”

Controller Test (mocking):

-- Create a mock for Model valid -> WorldMapMockValid.java
-- Create a mock for Model in valid -> WorldMapMockInValid.java
-- Create a mock for View valid -> SwingMonsterHuntViewMockValid.java
-- Create a mock for View in valid -> SwingMonsterHuntViewMockInvalid.java

Testing MonsterHuntController class	Input	Expected Value
Testing valid handle add player -Human player	controller.handleAddPlayerAction(“Harsha”, “Gym Room”, 3, false)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt add player is called with playerName: Harsha, roomName: Gym Room, maxCarryLimit: null, isComputer: false) and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing valid handle add player -Computer player	controller.handleAddPlayerAction(“Computer1”, “Gym Room”, 3, true)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt add player is called with playerName: Computer1, roomName: Gym Room, maxCarryLimit: null, isComputer: true) and the unique code is 12345 The method refresh is called with the unique code value as 12345"
Testing invalid handle add player- empty player name	controller.handleAddPlayerAction(“”, “Gym Room”, null, true)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 Invalid player details”

Testing MonsterHuntController class	Input	Expected Value
Testing invalid handle add player- null player name	controller.handleAddPlayerAction(null, “Gym Room”, null, true)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 Invalid player details”
Testing invalid handle add player- invalid room name	controller.handleAddPlayerAction(“Raghu”, “Temple”, null, false)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 Invalid player details”
Testing invalid handle add player- empty room name	controller.handleAddPlayerAction(“Nandhini”, “”, null, false)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 Invalid player details”
Testing invalid handle add player- null room name	controller.handleAddPlayerAction(“Nandhini”, null, null, false)	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 Invalid player details”

Testing MonsterHuntController class	Input	Expected Value
Play the game `playGame()`	controller.playGame()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345
Testing valid input handle move player (human player)	controller.playGame() controller.handleMovePlayerAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move player is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing Valid handle move player (computer player)	controller.playGame() controller.handleMovePlayerAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move player is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 The method refresh is called with the unique code value as 12345"
Testing invalid handle move player but not a neighbor to the player's current room	controller.playGame() controller.handleMovePlayerAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move player is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 Invalid move player action"

Testing MonsterHuntController class	Input	Expected Value
Testing invalid handle move player - the room name is not available in the world.	controller.playGame() controller.handleMovePlayerAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move player is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 Invalid move player action”
Testing invalid handle move player - invalid room name parameters	controller.playGame() controller.handleMovePlayerAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move player is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 Invalid move player action”

Testing MonsterHuntController class	Input	Expected Value
Testing valid input handle move pet (human player)	controller.playGame() controller. handleMovePetAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move pet is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing Valid handle move pet (computer player)	controller.playGame() controller. handleMovePetAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move pet is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing valid handle move pet but not a neighbor to the player's current room	controller.playGame() controller. handleMovePetAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move pet is called and the unique code is 12345 The method Monster hunt show prompt for move player is called with roomList: space1, space2 and the unique code is 12345 The method refresh is called with the unique code value as 12345"
Testing invalid handle move pet - the room name is not available in the world.	controller.playGame() controller. handleMovePetAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move pet is called and the unique code is 12345 The method Monster hunt show prompt for move pet is called and the unique code is 12345 Invalid move pet action"

Testing MonsterHuntController class	Input	Expected Value
Testing invalid handle move pet - invalid room name parameters	controller.playGame() controller. handleMovePetAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt move pet is called and the unique code is 12345 The method Monster hunt show prompt for move pet is called and the unique code is 12345 Invalid move pet action”
Testing valid handle pick weapon	controller.playGame() controller. handlePickWeaponAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt pick weapon is called, and the unique code is 12345 The method Monster hunt show prompt for pick weapon is called with weapon list: weapon1, weapon2 and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing invalid handle pick weapon - invalid weapon name	controller.playGame() controller. handlePickWeaponAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt pick weapon is called, and the unique code is 12345 The method Monster hunt show prompt for pick weapon is called with weapon list: weapon1, weapon2 and the unique code is 12345 Invalid pick weapon action”
Testing invalid handle pick weapon - empty weapon name	controller.playGame() controller. handlePickWeaponAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt pick weapon is called, and the unique code is 12345 The method Monster hunt show prompt for pick weapon is called with weapon list: weapon1, weapon2 and the unique code is 12345 Invalid pick weapon action”

Testing MonsterHuntController class	Input	Expected Value
Testing invalid handle pick weapon - null weapon name	controller.playGame() controller. handlePickWeaponAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt pick weapon is called, and the unique code is 12345 The method Monster hunt show prompt for pick weapon is called with weapon list: weapon1, weapon2 and the unique code is 12345 Invalid pick weapon action”
Testing valid handle look around- no pet in neighboring spaces	controller.playGame() controller.handleLookAround Action()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt look around is called and the unique code is 12345 The method Monster hunt show prompt for look around is called and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing valid handle look around- with pet in neighboring spaces	controller.playGame() controller.handleLookAroundAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt look around is called and the unique code is 12345 The method Monster hunt show prompt for look around is called and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
Testing valid handle make an attempt	controller.playGame() controller.handleMakeAttemptAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list: weapon1, weapon2 is called and the unique code is 12345 The method refresh is called with the unique code value as 12345"
Testing invalid handle make an attempt - invalid weapon name (human player)	controller.playGame() controller.handleMakeAttemptAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list:

Testing MonsterHuntController class	Input	Expected Value
		weapon1, weapon2 is called and the unique code is 12345 Invalid make attempt on target”
Testing invalid handle make an attempt - empty weapon name(human player)	controller.playGame() controller. handleMakeAttemptAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list: weapon1, weapon2 is called and the unique code is 12345 Invalid make attempt on target”

Testing MonsterHuntController class	Input	Expected Value
Testing invalid handle make an attempt - null weapon name(human player)	controller.playGame() controller. handleMakeAttemptAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list: weapon1, weapon2 is called and the unique code is 12345 Invalid make attempt on target”
Testing invalid handle make an attempt - invalid weapon name(computer player)	controller.playGame() controller. handleMakeAttemptAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list: weapon1, weapon2 is called and the unique code is 12345

Testing MonsterHuntController class	Input	Expected Value
		The method refresh is called with the unique code value as 12345" Invalid make attempt on target”
Testing invalid handle make an attempt - empty weapon name(computer player)	controller.playGame() controller. handleMakeAttemptAction()	"The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list: weapon1, weapon2 is called and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
		Invalid make attempt on target”
Testing invalid handle make an attempt - null weapon name(computer player)	controller.playGame() controller.handleMakeAttemptAction()	""The method setFeatures is called with the unique code value as 12345 The method makeVisible is called with the unique code value as 12345 The method Monster hunt make attempt on target is called and the unique code is 12345 The method Monster hunt show prompt for make an attempt with weapon list: weapon1, weapon2 is called and the unique code is 12345 The method refresh is called with the unique code value as 12345"

Testing MonsterHuntController class	Input	Expected Value
		Invalid make attempt on target”

Milestone 3 Testing plan:

Model testing plan:

Testing for PetImpl class:

Testing PetImpl class	Input	Expected Value
Get current room name of the pet.	new PetImpl(“Bella the cat”, “Kitchen”)	“Kitchen”
Constructor disallows invalid pet name.	new PetImpl(null, “Kitchen”)	IllegalArgumentException
Constructor disallows invalid room name of the pet.	new PetImpl(“Bella the cat”, null)	IllegalArgumentException
Equals same object	new PetImpl(“Bella the cat”, “Kitchen”)	True
Equals different objects with same values	new PetImpl(“Bella the cat”, “Kitchen”)	True
Equals different values	new PetImpl(“Bella the cat”, “Kitchen”) new PetImpl(“Luna the cat”, “Kitchen”)	False
Testing toString method	new PetImpl(“Bella the cat”, “Kitchen”)	Pet details: Name: Bella the cat, Current Room: Kitchen.
move the pet (invalid). ` moveThePet(roomName)`	new PetImpl(“Bella the cat”, “Kitchen”) moveThePet(null)	IllegalArgumentException

Testing PetImpl class	Input	Expected Value
move the pet (room name is not available in the world). ` moveThePet(roomName)`	new PetImpl(“Bella the cat”, “Kitchen”) moveThePet(“Balcony”)	IllegalArgumentException
move the pet. (valid) ` moveThePet(roomName)`	new PetImpl(“Bella the cat”, “Kitchen”) moveThePet(“Gym Room”)	Should move the pet to ‘Gym Room’ room. Pet details: Name: Bella the cat, Current Room: Gym Room.

Testing for WorldMapImpl class:

Note:

Here considering a helper method named `worldMap()` which will give an expected WorldMap instance.

Here `worldSpec` parameter is string builder which has information related to the world.

Testing WorldMapImpl class	Input	Expected Value
Get room details (pet details included). `getRoomDetails(roomName)`	worldMap(worldSpec) getRoomDetails(“Library”)	Room Details: Room name: “Library”, Weapons available: [“Knife”] Neighbors: [Kitchen], Available players: [computer1, raghu], Pet Name: “Bella the cat”.
Get room details (pet is not in that room). `getRoomDetails(roomName)`	worldMap(worldSpec) getRoomDetails(“Foyer”)	Room Details: Room name: “Foyer”, Weapons available: [“Steel Rod”] Neighbors: [Game Room], Available players: [john, harsha].
Get room details (pet is available in the neighbor room i.e. Bed Room). `getRoomDetails(roomName)`	worldMap(worldSpec) getRoomDetails(“Foyer”)	(Beed Room should not be there in the neighbor rooms) Room Details: Room name: “Foyer”, Weapons available: [“Steel Rod”] Neighbors: [Game Room], Available players: [john, harsha].
Get room details. (Invalid room name). `getRoomDetails(roomName)`	worldMap(worldSpec) getRoomDetails(null)	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Get room details. (Room name is not available in the world). `getRoomDetails(roomName)`	worldMap(worldSpec) getRoomDetails(“Balcony”)	IllegalArgumentException
Get current turn details (human player). `getCurrentTurnDetails()`	worldMap(worldSpec) getCurrentPlayerDetails()	Turn details (current turn): Name: raghu, Room: Kitchen, Target position: Foyer.
Get current turn details (computer player). `getCurrentTurnDetails()`	worldMap(worldSpec) getCurrentPlayerDetails()	Turn details (current turn): Name: computer 1, Room: Bed Room, Target position: Foyer.

Testing WorldMapImpl class	Input	Expected Value
Look around of current room. (Neighbors has no pet so all are displayed). `lookAround()`	worldMap(worldSpec) lookAround()	Look Around Details: Player's room: Kitchen Other players in room: ["computer 1", "alex"] Items in the room: ["Knife(9)"] Neighbor information: Neighbor 1: Name: Library, Items: ["Lamp(4)", "Fork(2)"] Players: ["computer 2"] Neighbor 2: Name: Bed Room, Items: ["steel rod(2)"] Players: ["john"]

Testing WorldMapImpl class	Input	Expected Value
Look around of current room. (Neighbor Bed Room has pet) `lookAround()`	worldMap(worldSpec) lookAround()	Look Around Details: Player's room: Kitchen Other players in room: ["computer 1", "alex"] Items in the room: ["Knife(9)"] Neighbor information: Neighbor 1: Name: Library, Items: ["Lamp(4)", "Fork(2)"] Players: ["computer 2"].
Can see the player. (Player is in current room) `canSeeThePlayer(playerName)`	worldMap(worldSpec) canSeeThePlayer("raghu")	True
Can see the player. (Player is in the neighbor room) ` canSeeThePlayer(playerName)`	worldMap(worldSpec) canSeeThePlayer("computer 1")	True
Can see the player. (Player is not in neighbor room) ` canSeeThePlayer(playerName)`	worldMap(worldSpec) canSeeThePlayer("jhon")	False

Testing WorldMapImpl class	Input	Expected Value
Can see the player. (Player name is invalid) ` canSeeThePlayer(playerName)`	worldMap(worldSpec) canSeeThePlayer(null)	False
Can see the player. (Player is at neighbor room where pet is available). ` canSeeThePlayer(playerName)`	worldMap(worldSpec) canSeeThePlayer(“Alex”)	False
Move the pet to a room. (Room name is invalid) ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(null)	IllegalArgumentException
Move the pet to a room. (Room name is not available in the world) ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Balcony”)	IllegalArgumentException
Move the pet to a room. (Room name is neighbor to current room). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Bed Room”)	Should move the pet to “Bed Room”.
Move the pet to a room. (Room name is not neighbor to current room). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Library”)	Should move the pet to “Library”.
Move the pet to a room. (To a room where there are at least one player). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Foyer”)	Should move the pet to “Foyer”.

Testing WorldMapImpl class	Input	Expected Value
Move the pet to a room. (To a room where there are no player). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Laundry Room”)	Should move the pet to “Laundry Room”.
Move the pet to a room. (To a room where there are no items). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Foyer”)	Should move the pet to “Foyer”.
Move the pet to a room. (To a room where there are at least 1 items). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Laundry Room”)	Should move the pet to “Laundry Room”.
Move the pet to a room. (To a room where target is available). ` moveThePet(roomName)`	worldMap(worldSpec) moveThePet(“Game Room”)	Should move the pet to “Game Room”.
Make an attempt on the target. (Attempt successful) ` makeAnAttempt(weaponName)`	worldMap(worldSpec) makeAnAttempt(“Knife”)	True
Make an attempt on the target. (Attempt unsuccessful, players are available in the neighbor room) ` makeAnAttempt(weaponName)`	worldMap(worldSpec) makeAnAttempt(“Knife”)	False

Testing WorldMapImpl class	Input	Expected Value
Make an attempt on the target. (Attempt unsuccessful, players are available in the neighbor room, which is invisible) ` makeAnAttempt(weaponName)`	worldMap(worldSpec) makeAnAttempt(“Knife”)	False
Make an attempt on the target. (weapon name is invalid) ` makeAnAttempt(weaponName)`	worldMap(worldSpec) makeAnAttempt(null)	IllegalArgumentException
Make an attempt on the target. (weapon name is invalid). ` makeAnAttempt(weaponName)`	worldMap(worldSpec) makeAnAttempt(“”)	IllegalArgumentException
Make an attempt on the target. (The player doesn’t have the weapon). ` makeAnAttempt(weaponName)`	worldMap(worldSpec) makeAnAttempt(“Steel Rod”)	IllegalArgumentException
Is game over (Target health became zero) ` isGameOver()`	worldMap(worldSpec) isGameOver()	True
Is game over (Target health became zero after player raghu made an attempt) ` isGameOver()`	worldMap(worldSpec) isGameOver()	True

Testing WorldMapImpl class	Input	Expected Value
Is game over (Target is still healthy). ` isGameOver()`	worldMap(worldSpec) isGameOver()	False
Get the winner. (Winner is not yet declared). ` getWinner()`	worldMap(worldSpec) getWinner()	“”
Get the winner. (Winner is finalized, human player). ` getWinner()`	worldMap(worldSpec) getWinner()	“raghu”
Get the winner. (Winner is finalized, computer player). ` getWinner()`	worldMap(worldSpec) getWinner()	“computer 1”

Test cases for MovePet class:

Testing MovePet class	Input	Expected Value
Valid input parameters	MovePet(“Library”)	"Input: Room Name = Library "
Valid input parameters (another player)	MovePet(“Laundry Room”)	"Input: Room Name = Laundry Room "

Testing MovePet class	Input	Expected Value
Correct input parameters but not a neighbor to the player's current room	MovePet("Foyer")	"Input: Room Name = Foyer"
Correct input parameters but the room name is not available in the world.	MovePet("Balcony")	"Input: Room Name = Balcony"
Correct input parameters but invalid room name parameters	MovePet("")	"Input: Room Name = "
Constructor disallows invalid 'in' (input) field.	new MovePet(null, appendable)	IllegalArgumentException
Constructor disallows invalid 'out' (output) field.	new MovePet(readable, null)	IllegalArgumentException

Test cases for MakeAttempt class:

Testing MakeAttempt class	Input	Expected Value
Valid input parameters	MakeAttempt("Knife")	"Input: Weapon Name = Knife"
Valid input parameters (another player)	MakeAttempt("Chain Saw")	"Input: Weapon Name = Chain Saw"
Correct input parameters but the player doesn't have that weapon.	MakeAttempt("Scissors")	"Input: Weapon Name = Scissors"

Testing MakeAttempt class	Input	Expected Value
Correct input parameters but the weapon name is not available in the world.	MakeAttempt(“Machete”)	"Input: Weapon Name = Machete"
Correct input parameters but invalid room name parameters	MakeAttempt(“”)	"Input: Weapon Name = "
Constructor disallows invalid ‘in’ (input) field.	new MovePet(null, appendable)	IllegalArgumentException
Constructor disallows invalid ‘out’ (output) field.	new MovePet(readable, null)	IllegalArgumentException

Mock model test for WorldMapImpl class (WorldMapImplMock):

Note:

Here considering a helper method named ` worldMapImpMock ()` which will give an expected WorldMap instance.

Here `worldSpec` parameter is string builder which has information related to the world.

Testing WorldMapImplMock class	Input	Expected Value
Get room details mock 'getRoomDetails(roomName)'	worldMapImpMock(worldSpec) getRoomDetails("Kitchen")	"Input: Room Name = Kitchen\n" "The getRoomDetails() method called, unique code: 5453453"
Get room details mock (room name is empty). 'getRoomDetails(roomName)'	worldMapImpMock(worldSpec) getRoomDetails("")	"Input: Room Name =\n" "The getRoomDetails() method called, unique code: 5453453"
Get room details mock (room has pet available). 'getRoomDetails(roomName)'	worldMapImpMock(worldSpec) getRoomDetails("Bed Room")	"Input: Room Name = Bed Room\n" "The getRoomDetails() method called, unique code: 5453453"
Get current turn details mock (room has pet available, current player is human player). 'getCurrentTurnDetails()'	worldMapImpMock(worldSpec) getCurrentTurnDetails()	"The getCurrentTurnDetails() method called, unique code: 5453453"

Testing WorldMapImplMock class	Input	Expected Value
Get current turn details mock (room has pet available, current player is computer player). 'getCurrentTurnDetails()'	worldMapImpMock(worldSpec) getCurrentTurnDetails()	"The getCurrentTurnDetails() method called, unique code: 5453453"
Look around mock. 'lookAround()'	worldMapImpMock(worldSpec) lookAround()	"The lookAround() method called, unique code: 5453453"
Look around mock (if the neighbor room has pet). 'lookAround()'	worldMapImpMock(worldSpec) lookAround()	"The lookAround() method called, unique code: 5453453"
Determine if one player can see another mock. (Invalid player name) 'canSeeThePlayer(playerName)'	worldMapImpMock(worldSpec) canSeeThePlayer("raghsds")	"Input: Player name = raghsds." "The canSeeThePlayer() method called, unique code: 5453453"

Testing WorldMapImplMock class	Input	Expected Value
Determine if one player can see another mock. (Valid player name) 'canSeeThePlayer(playerName)'	worldMapImpMock(worldSpec) canSeeThePlayer("raghu")	"Input: Player name = raghu." "The canSeeThePlayer() method called, unique code: 5453453"
Determine if one player can see another mock. (Computer player) 'canSeeThePlayer(playerName)'	worldMapImpMock(worldSpec) canSeeThePlayer("computer1")	"Input: Player name = computer1." "The canSeeThePlayer() method called, unique code: 5453453"
Move the pet mock. 'moveThePet(roomName)'	worldMapImpMock(worldSpec) moveThePet("Kitchen")	"Input: Room name = Kitchen." "The moveThePet() method called, unique code: 5453453"
Move the pet mock. (Empty room name) 'moveThePet(roomName)'	worldMapImpMock(worldSpec) moveThePet("")	"Input: Room name =." "The moveThePet() method called, unique code: 5453453"

Testing WorldMapImplMock class	Input	Expected Value
Move the pet mock. (Room is not neighbor) 'moveThePet(roomName)'	worldMapImpMock(worldSpec) moveThePet("Foyer")	"Input: Room name =." "The moveThePet() method called, unique code: 5453453"
Make an attempt mock 'makeAnAttempt(weaponName)'	worldMapImpMock(worldSpec) makeAnAttempt("Knife")	"Input: Weapon name = Knife." "The makeAnAttempt() method called, unique code: 5453453"
Make an attempt mock (empty weapon name). 'makeAnAttempt(weaponName)'	worldMapImpMock(worldSpec) makeAnAttempt("")	"Input: Weapon name =." "The makeAnAttempt() method called, unique code: 5453453"
Make an attempt mock (invalid weapon name). 'makeAnAttempt(weaponName)'	worldMapImpMock(worldSpec) makeAnAttempt("ssdfs")	"Input: Weapon name = ssdfs." "The makeAnAttempt() method called, unique code: 5453453"

Testing WorldMapImplMock class	Input	Expected Value
Is game over mock. (Game is not yet over) 'isGameOver()'	worldMapImpMock(worldSpec) isGameOver()	"The isGameOver() method called, unique code: 5453453"
Is game over mock. (Game is over) 'isGameOver()'	worldMapImpMock(worldSpec) isGameOver()	"The isGameOver() method called, unique code: 5453453"
Get the winner of the game (human player is the winner). 'getWinner()'	worldMapImpMock(worldSpec) getWinner()	"The getWinner() method called, unique code: 5453453"
Get the winner of the game (computer player is the winner). 'getWinner()'	worldMapImpMock(worldSpec) getWinner()	"The getWinner() method called, unique code: 5453453"

Testing WorldMapImplMock class	Input	Expected Value
Get the winner of the game (game is not yet completed). 'getWinner()'	worldMapImpMock(worldSpec) getWinner()	"The getWinner() method called, unique code: 5453453"

Milestone 2 Testing plan:

Test cases for Controller (MonsterHuntController):

Mock model for Commands:

All the mock models have a log which will log all the input parameters for the commands.

StringBuilder log (which logs the input).

In – Readable

out - Appendable

Test cases for MonsterHuntControllerImpl:

Here in(Readable) and out(Appendable) comes from driver class.

Testing MonsterHuntControllerImpl class	Input	Expected Value
Valid parameter to the constructor.	MonsterHuntControllerImpl(in, out)	Will instantiate the controller class.
Constructor disallows invalid `in` parameter.	MonsterHuntControllerImpl(in, out) - In (null)	IllegalArgumentException
Constructor disallows invalid `out` parameter.	MonsterHuntControllerImpl(in, out) - out (null)	IllegalArgumentException
Invalid model to `playGame(null)` method.	MonsterHuntControllerImpl(in, out)	IllegalArgumentException
Invalid number of turns parameter. `playGame(model, -10)`	MonsterHuntControllerImpl(in, out)	IllegalArgumentException

Test cases for AddPlayer class:

Testing AddPlayer class	Input	Expected Value
Valid input parameters (Player 1)	AddPlayer(“Harsha”, “Kitchen”, 3, HUMAN)	"Input: name=Harsha\n room = Kitchen\n Carry Limit = 3\n Player Type = HUMAN\n"
Valid input parameters (Player 2)	AddPlayer(“Rahul”, “Gym Room”, 2, HUMAN)	"Input: name= Rahul\n room = Gym Room\n Carry Limit = 2\n Player Type = HUMAN\n"
Passing Correct input parameters but invalid Carry limit parameters	AddPlayer(“Rahul”, “Gym Room”, -2, HUMAN)	"Input: name= Rahul\n room = Gym Room\n Carry Limit = -2\n Player Type = HUMAN\n"
Passing Correct input parameters but invalid player name parameters	AddPlayer(“”, “Gym Room”, 2, HUMAN)	"Input: name= \n room = Gym Room\n Carry Limit = 2\n Player Type = HUMAN\n"
Passing Correct input parameters but invalid room name parameters	AddPlayer(“Rahul”, “”, 2, HUMAN)	"Input: name= Rahul \n room = Gym Room\n Carry Limit = 2\n Player Type = HUMAN\n"
Passing Correct input parameters but invalid player type parameters	AddPlayer(“Rahul”, “Gym Room”, 2, null)	"Input: name= Rahul \n room = Gym Room\n Carry Limit = 2\n Player Type = \n"

Test cases for MovePlayer class:

Testing MovePlayer class	Input	Expected Value
Valid input parameters	MovePlayer(“Kitchen”)	"Input: Room Name = Kitchen"
Valid input parameters (another player)	MovePlayer(“Gym Room”)	"Input: Room Name = Gym Room"
Correct input parameters but not a neighbor to the player’s current room	MovePlayer(“Library”)	"Input: Room Name = Library"
Correct input parameters but the room name is not available in the world.	MovePlayer(“Balcony”)	"Input: Room Name = Balcony"
Correct input parameters but invalid room name parameters	MovePlayer(“”)	"Input: Room Name = "

Test cases for PickWeapon class:

Testing PickWeapon class	Input	Expected Value
Valid input parameters	PickWeapon(“Knife”)	"Input: Weapon Name = Knife"
Valid input parameters (another player)	PickWeapon(“Chain Saw”)	"Input: Weapon Name = Chain Saw"
Correct input parameters but the weapon is not available in the player’s current room	PickWeapon(“Scissors”)	"Input: Weapon Name = Scissors"
Correct input parameters but the weapon name is not available in the world.	PickWeapon(“Machete”)	"Input: Weapon Name = Machete"
Correct input parameters but invalid room name parameters	PickWeapon(“”)	"Input: Weapon Name = "

Test cases for DisplayPlayerInfo class:

Testing DisplayplayerInfo class	Input	Expected Value
Valid input parameters	DisplayPlayerInfo(“Harsha”)	"Input: Player Name = Harsha"

Testing DisplayplayerInfo class	Input	Expected Value
Valid input parameters (another player)	DisplayPlayerInfo(“Rahul”)	"Input: Player Name = Rahul"
Correct input parameters but the player’s name is not available.	DisplayPlayerInfo(“XYZ”)	"Input: Player Name = XYZ"
Correct input parameters but invalid player name parameters	DisplayPlayerInfo(“”)	"Input: Player Name ="

Test cases for DisplayRoomInfo class:

Testing DisplayRoomInfo class	Input	Expected Value
Valid input parameters	DisplayRoomInfo(“Kitchen”)	"Input: Room name = Kitchen"
Valid input parameters (another player)	DisplayRoomInfo(“Gym Room”)	"Input: Room name = Gym Room"
Correct input parameters but the Room name is not available in the world.	DisplayRoomInfo(“Balcony”)	"Input: Room name = Balcony"
Correct input parameters but invalid room name parameters	DisplayRoomInfo(“”)	"Input: Room name ="

Test case for controller (output):

Note:

Here for all the test cases, there are few fields:

m – worldMap instance.

input – StringReader (to store the input for model)

gameLog – Appendable (to store the output of model)

controller – MonsterHuntController.

Test case for DisplayRoomInfo (output):

Testing DisplayRoomInfo class	Input	Expected Value
Valid input parameters	DisplayRoomInfo(“Kitchen”) controller.playGame(m)	“Room Details: Name: Kitchen, Weapons available: [“knife”], Neighbors: [“Dining Room”], Players Available: [“Harsha”] ”

Testing DisplayRoomInfo class	Input	Expected Value
Testing getResponse() method	DisplayRoomInfo(“Kitchen”) controller.playGame(m)	“Room Details: Name: Kitchen, Weapons available: [“knife”], Neighbors: [“Dining Room”], Players Available: [“Harsha”] ”
Invalid input parameter	DisplayRoomInfo(“”) controller.playGame(m)	Unsuccessful display room.
Correct input parameters but the Room name is not available in the world.	DisplayRoomInfo(“Balcony”) controller.playGame(m)	Unsuccessful display room.

Test case for GenerateWorld (output):

Testing GenerateWorld class	Input	Expected Value
Successful image creation.	GenerateWorld() controller.playGame(m)	Successfully created the image at output.png

Testing GenerateWorld class	Input	Expected Value
Unsuccessful image creation.	GenerateWorld() controller.playGame(m)	There was some problem with creating image.
Testing `getResponse()` method.	GenerateWorld() controller.playGame(m)	Successfully created the image at output.png

Test case for AddPlayer (output):

Testing AddPlayer class	Input	Expected Value
Successful adding player	AddPlayer(“Harsha”, “Gym Room”, 3, HUMAN) controller.playGame(m)	Successfully added the player to the world.
Unsuccessful adding player.	AddPlayer(“”, “Gym Room”, 3, HUMAN) controller.playGame(m)	Adding player was unsuccessful.

Testing AddPlayer class	Input	Expected Value
Testing `getResponse()` method..	AddPlayer(“Harsha”, “Gym Room”, 3, HUMAN) controller.playGame(m)	Successfully added the player to the world.

Test case for MovePlayer (output):

Testing MovePlayer class	Input	Expected Value
Successful moving player	MovePlayer(“Kitchen”) controller.playGame(m)	Successfully moved the player to Kitchen.
Unsuccessful moving player.	MovePlayer(“”) controller.playGame(m)	Moving player was unsuccessful.
Testing `getResponse()` method..	MovePlayer(“Kitchen”) controller.playGame(m)	Successfully moved the player to Kitchen.

Test case for PickWeapon (output):

Testing PickWeapon class	Input	Expected Value
Successful weapon pickup	PickWeapon(“Knife”) controller.playGame(m)	Successfully picked up the weapon Knife.
Unsuccessful weapon pickup	PickWeapon(“”) controller.playGame(m)	Picking weapon was unsuccessful.
Testing `getResponse()` method..	PickWeapon(“Knife”) controller.playGame(m)	Successfully picked up the weapon Knife.

Test case for LookAround (output):

Testing LookAround class	Input	Expected Value
Successful look around	LookAround() controller.playGame(m)	"Look Around Details: Current room Name: Kitchen, Neighbors: [Dining Room]"
Unsuccessful looking around	LookAround() controller.playGame(m)	Unsuccessful looking around.

Test case for DisplayPlayerInfo (output):

Testing DisplayPlayerInfo class	Input	Expected Value
Successful Player information	DisplayPlayerInfo("Harsha") controller.playGame(m)	"Player Details: Name: Harsha, Room Name: Kitchen, Carrying Weapons: [Knife] "
Unsuccessful Player information	DisplayPlayerInfo("") controller.playGame(m)	Unsuccessful for showing player information.

Testing DisplayPlayerInfo class	Input	Expected Value
Unsuccessful Player information	DisplayPlayerInfo(“Harsha”) controller.playGame(m)	“Player Details: Name: Harsha, Room Name: Kitchen, Carrying Weapons: [Knife] ”

Test Cases for Models:

Testing for PlayerImpl Class:

Note:

Here considering a helper method named `player()` which will give an expected Player instance.

‘weaponList ’is list of weapons for the player.

Testing PlayerImpl class	Input	Expected Value
Get name of the player	player(“Harsha”, “Kitchen”, 3, weaponList, HUMAN)	"Harsha"
Get current room name where player is available.	player(“Harsha”, “Kitchen”, 3, weaponList, HUMAN)	“Kitchen”

Testing PlayerImpl class	Input	Expected Value
Get the maximum limit of weapons that a player can carry.	player(“Harsha”, “Kitchen”, 3, weaponList, HUMAN)	3
Get carrying list of weapons by the player	player(“Harsha”, “Kitchen”, 3, weaponList, HUMAN)	[“Knife”, “Chain Saw”]
Get type of the player	player(“Harsha”, “Kitchen”, 3, weaponList, HUMAN)	HUMAN
Constructor disallows invalid player name	player(“”, “Kitchen”, 3, weaponList, HUMAN)	IllegalArgumentException
Constructor disallows invalid player name	player(null, “Kitchen”, 3, weaponList, HUMAN)	IllegalArgumentException
Constructor disallows invalid room name	player(“Harsha”, “”, 3, weaponList, HUMAN)	IllegalArgumentException
Constructor disallows invalid room name	player(“Harsha”, null, 3, weaponList, HUMAN)	IllegalArgumentException
Constructor disallows invalid room name (where room is not available in the world)	player(“Harsha”, “balcony”, 3, weaponList, HUMAN)	IllegalArgumentException
Constructor disallows invalid carrying limit	player(“Harsha”, “balcony”, -3, weaponList, HUMAN)	IllegalArgumentException

Testing PlayerImpl class	Input	Expected Value
Constructor disallows invalid weapon list.	player(“Harsha”, “balcony”, -3, null, HUMAN)	IllegalArgumentException
Constructor disallows invalid player type.	player(“Harsha”, “balcony”, -3, weaponList, null)	IllegalArgumentException
Pick up certain weapon for a player `pickUpWeapon(“Crepe Pan”)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Weapon Name (Crepe Pan)	Should add ‘Crepe pan’ to the weaponList for the player.
Pick up certain weapon for a player (if limit is exceeded) `pickUpWeapon(“Scissors”)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Weapon Name (Scissors)	IllegalStateException
Pick up certain weapon for a player (if weapon name is invalid) `pickUpWeapon(“”)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Weapon Name ()	IllegalArgumentException
Pick up certain weapon for a player (if weapon name is invalid) `pickUpWeapon(null)`	player(“Harsha”, “”, 3, weaponList, HUMAN)	IllegalArgumentException

Testing PlayerImpl class	Input	Expected Value
Pick up certain weapon for a player (if weapon is not available in the current room) `pickUpWeapon(“Stick”)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Weapon Name (Stick)	IllegalArgumentException
Move current player to one of the neighbor rooms. `moveThePlayer(“Gym Room”)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Room Name (Gym room)	Should change the player `currentRoomName` field to new room.
Move current player to one of the neighbor rooms (Invalid room name). `moveThePlayer(“”)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Room Name (“”)	IllegalArgumentException
Move current player to one of the neighbor rooms (Invalid room name). `moveThePlayer(null)`	player(“Harsha”, “”, 3, weaponList, HUMAN) - Room Name (null)	IllegalArgumentException

Testing PlayerImpl class	Input	Expected Value
Move current player to one of the neighbor rooms (If room is not a neighbor). `moveThePlayer("Armory")`	player("Harsha", "", 3, weaponList, HUMAN) - Room Name (Armory)	IllegalArgumentException
Equals same object	player("Harsha", "", 3, weaponList, HUMAN)	True
Equals different objects with same values	player("Harsha", "", 3, weaponList, HUMAN)	True
Equals different values	player("Harsha", "", 3, weaponList, HUMAN) player("computer", "", 2, weaponList, COMPUTER)	False
Testing toString method	player("Harsha", "", 3, weaponList, HUMAN)	Player Details: Name: "Harsha", Carry Limit: 3, Carrying weapons:["Knife", "Chain Saw"], type: Human.

Testing for WorldMapImpl Class:

Note:

Here considering a helper method named `worldMap()` which will give an expected WorldMap instance.

Here `worldSpec` parameter is string builder which has information late d to the world.

Testing WorldMapImpl class	Input	Expected Value
Get name	worldMap(worldSpec)	"Monster Mansion"
Get number of rows	worldMap(worldSpec)	30
Get number of columns	worldMap(worldSpec)	35
Get the target character	worldMap(worldSpec)	“Monster”
Constructor disallows invalid world details	worldMap(worldSpec) - empty world name. - negative number of rows. - negative number of columns. - Zero columns or rows	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Constructor allows valid target details	worldMap(worldSpec) - Monster (target name) - 70 (health)	Target name = “Monster” Health = 70
Constructor disallows invalid target details.	worldMap(worldSpec) - empty target name. - negative/zero health.	IllegalArgumentException
Constructor allows valid number of rooms	worldMap(worldSpec) 22	22
Constructor disallows invalid number of rooms	worldMap(worldSpec) -10	IllegalArgumentException
Constructor allows valid room details.	worldMap(worldSpec) RoomImpl (“Game Room”, 1, 18, 5, 21)	Room name = ”Game Room” Coordinates = 1, 18, 5, 21
Constructor disallows two rooms overlap	worldMap(worldSpec) RoomImpl (“Game Room”, 1, 18, 5, 21) RoomImpl (“Study Room”, 1, 19, 4, 20)	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Constructor disallows Invalid room details.	worldMap(worldSpec) RoomImpl (null, 1, 18, 5, 21)	IllegalArgumentException
Constructor disallows Invalid room details.	worldMap(worldSpec) RoomImpl (“Game Room”, -1, 18, 5, 21)	IllegalArgumentException
Constructor disallows Invalid room details.	worldMap(worldSpec) RoomImpl ((“Game Room”, 1, -18, 5, 21)	IllegalArgumentException
Constructor disallows Invalid room details.	worldMap(worldSpec) RoomImpl ((“Game Room”, 1, 18, -5, 21)	IllegalArgumentException
Constructor disallows Invalid room details.	worldMap(worldSpec) RoomImpl ((“Game Room”, 1, 18, 5, -21)	IllegalArgumentException
Constructor allows valid weapon details.	worldMap(worldSpec) WeaponImpl(“Chain Saw”, 4, 2)	Weapon name = “Chain Saw”, Damage Value = 4, Room Index = 2

Testing WorldMapImpl class	Input	Expected Value
Constructor disallows invalid weapon details.	worldMap(worldSpec) - empty weapon name. - negative/zero room index. - Negative/ zero damage value.	IllegalArgumentException
toString method test	worldMap(worldSpec)	“Room Details: (Name: Kitchen, Upper left row: 16, Upper left column: 3, Lower right row: 21, Lower right column: 10)”
Get the neighbors of “Kitchen” (Multiple neighbors available) `getTheNeighbors(“Armory”)`	worldMap(worldSpec)	[“Panic Room”, “Drawing Rom”]
Get the neighbors of “Kitchen” (single neighbors available) `getTheNeighbors(“Storage Room”)`	worldMap(worldSpec)	[“Family Room”]
Get the neighbors of “Mud Room” (no neighbors available) `getTheNeighbors(“Mud Room”)`	worldMap(worldSpec)	[“”]
Get the neighbors of room (invalid room name). `getTheNeighbors(null)`	worldMap(worldSpec)	IllegalArgumentException
Get the neighbors of a room (room is not available in the world) `getTheNeighbors(“cellar”)`	worldMap(worldSpec)	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Get the information about a room (invalid room name). ` getRoomDetails (null)`	worldMap(worldSpec)	IllegalArgumentException
Get the information about a room (room name not available in the world). ` getRoomDetails (“cellar”)`	worldMap(worldSpec) - Room name (Cellar)	IllegalArgumentException
Get the information about room `getRoomDetails(“Storage Room”)`	worldMap(worldSpec) - Room name (Storage Room)	Room Details: Room name: Storage Room, Weapons in room: [“Chain saw”], Neighbors: [“Family Room”], Players: [“Harsha”, “Rahul”]
Move the target character (Single time) `moveTheTarget()`	worldMap(worldSpec)	Current position of the character should increase by 1 (if current position is 2 then it will increase to 3)
Move the target character (Multiple times) `moveTheTarget()` `moveTheTarget()`	worldMap(worldSpec)	Current position of the character should increase by 2
Move the target character (Max number of rooms reached) `moveTheTarget()`	worldMap(worldSpec)	Current position of the character should be zero.

Testing WorldMapImpl class	Input	Expected Value
Generate a graphical image for the World. (Successful image generation) `generateTheWorldImage()`	worldMap(worldSpec)	The world image should be created with file name 'output.png'
Generate a graphical image for the World. (Un successful image generation) `generateTheWorldImage()`	worldMap(worldSpec)	IllegalStateException
Adding the player to the game. `addPlayer("harsha", "Kitchen", 3, HUMAN)`	worldMap(worldSpec) - Player Name (Harsha) - Chosen room (Kitchen) - Item carry limit (3) - Player type (HUMAN)	Player needs to be added to the game with following details: Player Name = "Harsha" Chosen room = "Kitchen" Item carry limit = 3 Player type = HUMAN
Adding the player to the game. (Invalid name) `addPlayer(null, "Kitchen", 3, HUMAN)`	worldMap(worldSpec) - Player Name (null) - Chosen room (Kitchen) - Item carry limit (3) - Player type (HUMAN)	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Adding the player to the game. (Invalid room name) `addPlayer(“harsha”, “”, 3, HUMAN)`	worldMap(worldSpec) - Player Name (Harsha) - Chosen room () - Item carry limit (3) - Player type (HUMAN)	IllegalArgumentException
Adding the player to the game. (Invalid items carry limit) `addPlayer(“harsha”, “Kitchen”, -3, HUMAN)`	worldMap(worldSpec) - Player Name (Harsha) - Chosen room (Kitchen) - Item carry limit (-3) - Player type (HUMAN)	IllegalArgumentException
Adding the player to the game. (Invalid player type) `addPlayer(“harsha”, “Kitchen”, 3, null)`	worldMap(worldSpec) - Player Name (Harsha) - Chosen room (Kitchen) - Item carry limit (3) - Player type (null)	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Moving The Player to Neighbor Room. `moveThePlayerToNeighborName(“Armory”)`	worldMap(worldSpec) - Room Name (Armory)	The current player room should be Armory.
Moving The Player to Neighbor Room. (Not a neighbor) `moveThePlayerToNeighborName(“cellar”)	worldMap(worldSpec) - Room Name (cellar)	IllegalArgumentException
Moving The Player to Neighbor Room. (Not a valid room name) `moveThePlayerToNeighborName(null)	worldMap(worldSpec) - Room Name (null)	IllegalArgumentException
Moving The Player to Neighbor Room. (Room name is not available in the world) `moveThePlayerToNeighborName(“balcony”)	worldMap(worldSpec) - Room Name (balcony)	IllegalArgumentException
Moving The Player to Neighbor Room. (No neighbors available for that room) `moveThePlayerToNeighborName(“balcony”)	worldMap(worldSpec)	IllegalArgumentException
Getting current turn in the game. `getPlayerForCurrentTurn()`	worldMap(worldSpec)	“Harsha”

Testing WorldMapImpl class	Input	Expected Value
Getting current turn in the game. `getPlayerForCurrentTurn()`	worldMap(worldSpec)	“John”
Getting current turn in the game. `getPlayerForCurrentTurn()`	worldMap(worldSpec)	“Computer 1”
Pick up a weapon for the current player. `pickUpWeapon(“Knife”)`	worldMap(worldSpec) - Weapon name (Knife)	Players carrying weapons list should have knife added to it and weapons list of room should remove this weapon.
Pick up a weapon for the current player. (Weapon name is not available in the current room). `pickUpWeapon(“Chain Saw”)`	worldMap(worldSpec) - Weapon name (Chain Saw)	IllegalArgumentException
Pick up a weapon for the current player. (Invalid weapon name) `pickUpWeapon(“”)`	worldMap(worldSpec) - Weapon name ()	IllegalArgumentException

Testing WorldMapImpl class	Input	Expected Value
Pick up a weapon for the current player. (weapon not available in the room and world) `pickUpWeapon("Machete")`	worldMap(worldSpec) - Weapon name (Machete)	IllegalArgumentException
Getting the player description. (Human) `getPlayerDetails()`	worldMap(worldSpec)	Player Details: Name = Harsha, Current Room = "Store Room", Carrying weapons List = ["Knife", "Chain Saw"].
Getting the player description. (Computer) `getPlayerDetails()`	worldMap(worldSpec)	Player Details: Name = Computer 1, Current Room = "Bed Room", Carrying weapons List = ["Table Lamp"].
Player looking around from the current room. (Human) `lookAround()`	worldMap(worldSpec)	Look around for Harsha: Current room = "Store Room", Neighbors = ["Kitchen", "Dining Room"]

Testing WorldMapImpl class	Input	Expected Value
Player looking around from the current room. (Computer) `lookAround()`	worldMap(worldSpec)	Look around for Computer 1: Current room = “Bed Room”, Neighbors = [“Gym Room”, “Laundry Room”]
Check if the player is a computer player or not. `isComputerPlayer(“Harsha”)`	worldMap(worldSpec)	false
Check if the player is a computer player or not. `isComputerPlayer(“Computer1”)`	worldMap(worldSpec)	true
Check if the player is a computer player or not. (Invalid player name). `isComputerPlayer(“”)`	worldMap(worldSpec)	IllegalArgumentException
Check if the player is a computer player or not. (Player name is not available). `isComputerPlayer(“Smith”)`	worldMap(worldSpec)	IllegalArgumentException

Testing for RoomImpl Class:

Testing RoomImpl class	Input	Expected Value
Get name	RoomImpl(“Kitchen”, 16, 3, 21, 10)	"Kitchen"
Get upper left X coordinate	RoomImpl(“Kitchen”, 16, 3, 21, 10)	16
Get upper left Y coordinate	RoomImpl (“Kitchen”, 16, 3, 21, 10)	3
Get lower right X coordinate	RoomImpl (“Kitchen”, 16, 3, 21, 10)	21
Get lower right Y coordinate	RoomImpl (“Kitchen”, 16, 3, 21, 10)	10
Equals same object	RoomImpl (“Kitchen”, 16, 3, 21, 10)	True
Equals different objects with same values	RoomImpl (“Kitchen”, 16, 3, 21, 10)	True
Equals different values	RoomImpl (“Kitchen”, 16, 3, 21, 10) RoomImpl (“Kitchen”, 28, 11, 21, 9)	False

Testing RoomImpl class	Input	Expected Value
Constructor disallows negative upper left row	RoomImpl (“Kitchen”, -16, 3, 21, 10)	IllegalArgumentException
Constructor disallows negative upper left column	RoomImpl (“Kitchen”, 16, -3, 21, 10)	IllegalArgumentException
Constructor disallows negative lower right row	RoomImpl (“Kitchen”, 16, 3, -21, 10)	IllegalArgumentException
Constructor disallows negative lower right column	RoomImpl (“Kitchen”, 16, 3, 21, -10)	IllegalArgumentException
Constructor disallows name as null values	RoomImpl (null, 16, 3, 21, 10)	IllegalArgumentException
toString method test	RoomImpl (“Kitchen”, 16, 3, 21, 10)	“Room Details: (Name: Kitchen, Upper left X: 16, Upper left Y: 3, Lower right X: 21, Lower right Y: 10)”
Get List of weapons available in the room. `getWeaponList()`	RoomImpl (“Kitchen”, 16, 3, 21, 10)	[“Sharp Knife”, “Crepe Pan”]
Get all the rooms which are neighbors to the Kitchen. `getNeighbors()`	RoomImpl (“Kitchen”, 16, 3, 21, 10)	[“Dining Room”, “Bedroom”, “Drawing Room”, “Family Room”]
Remove the weapon from the current room. `removeWeaponFromRoom(“Knife”)`	RoomImpl (“Kitchen”, 16, 3, 21, 10)	Should remove the weapon by name Knife from Kitchen Room.

Testing RoomImpl class	Input	Expected Value
Remove the weapon from the current room. (If Weapon is not available) `removeWeaponFromRoom("Knife")`	RoomImpl ("Kitchen", 16, 3, 21, 10)	IllegalArgumentException.

Testing for WeaponImpl Class:

Testing WeaponImpl class	Input	Expected Value
Get name	WeaponImpl("Chain Saw", 4, 2)	"Chain Saw"
Get damage value	WeaponImpl("Chain Saw", 4, 2)	4
Get room index	WeaponImpl("Chain Saw", 4, 2)	2
Constructor disallows negative damage	WeaponImpl("Chain Saw", -4, 2)	IllegalArgumentException
Constructor disallows negative room index	WeaponImpl("Chain Saw", 4, -2)	IllegalArgumentException
Constructor disallows room index greater than number of rooms	WeaponImpl("Chain Saw", 4, 100)	IllegalArgumentException
Equals same object	WeaponImpl("Chain Saw", 4, 2)	True

Testing WeaponImpl class	Input	Expected Value
Equals different objects with same values	WeaponImpl(“Chain Saw”, 4, 2)	True
Equals different values	WeaponImpl(“Chain Saw”, 4, 2) WeaponImpl(“Knife”, 2, 1)	False
toString method test	WeaponImpl(“Chain Saw”, 4, 2)	“Weapon Details: (Name: Chain Saw, Damage value: 4, Room Index: 2)”

Testing for TargetImpl Class:

Testing TargetImpl class	Input	Expected Value
Get name	TargetImpl("Doctor Lucky ", 50, 3)	"Doctor Lucky"
Get health	TargetImpl("Doctor Lucky ", 50, 3)	50
Get current position	TargetImpl("Doctor Lucky ", 50, 3)	3
Constructor disallows invalid target name	TargetImpl(null, 50, 3)	IllegalArgumentException
Constructor disallows negative Health	TargetImpl("Doctor Lucky ", -50, 3)	IllegalArgumentException
Constructor disallows Negative current position	TargetImpl("Doctor Lucky ", 50, -3)	IllegalArgumentException
Constructor disallows current position which is greater than number of rooms	TargetImpl("Doctor Lucky ", 50, 300)	IllegalArgumentException
Constructor disallows if health is zero	TargetImpl("Doctor Lucky ", 0, 3)	IllegalArgumentException
toString method test	TargetImpl("Doctor Lucky ", 50, 3)	“Target Details: (Name: Doctor Lucky, health: 50, Current position: 3)”
Move target to the next step. `moveNextStep()`	TargetImpl("Doctor Lucky ", 50, 3)	Target Current position values should be 4
Moving the target to step zero because maximum steps are done. `moveTargetToStepZero()`	TargetImpl("Doctor Lucky ", 50, 20)	Target Current position values should be 0

Testing TargetImpl class	Input	Expected Value
Equals same object	TargetImpl("Doctor Lucky ", 50, 20)	True
Equals different objects with same values	TargetImpl("Doctor Lucky ", 50, 20)	True
Equals different values	TargetImpl("Doctor Lucky ", 50, 20) TargetImpl("Monster ", 80, 3)	False