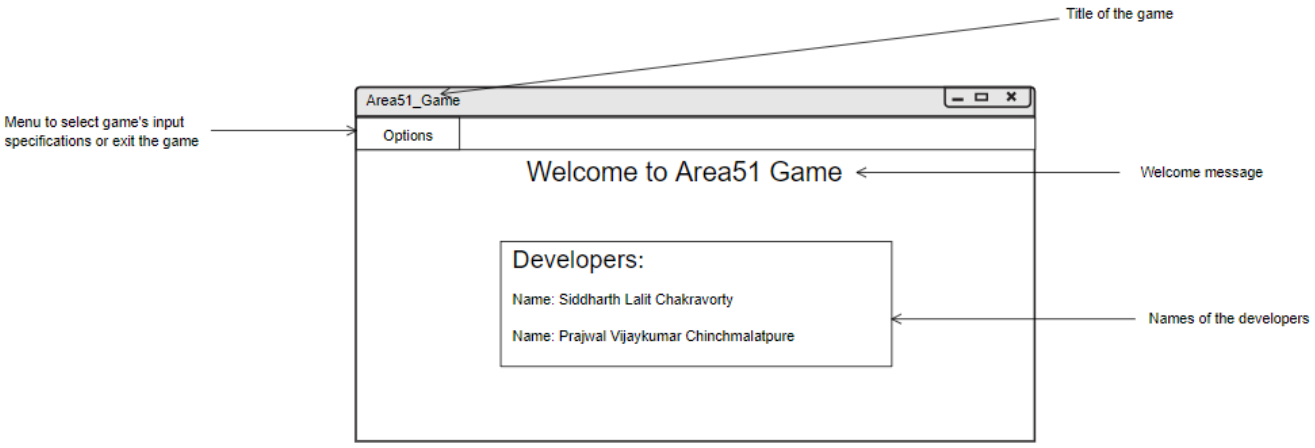
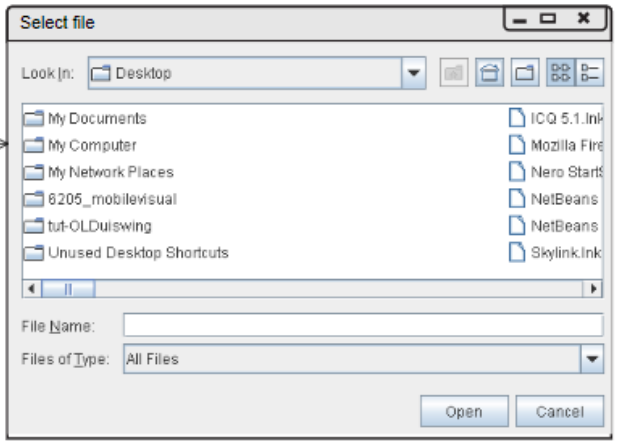
Screen 1- Welcome screen

1.



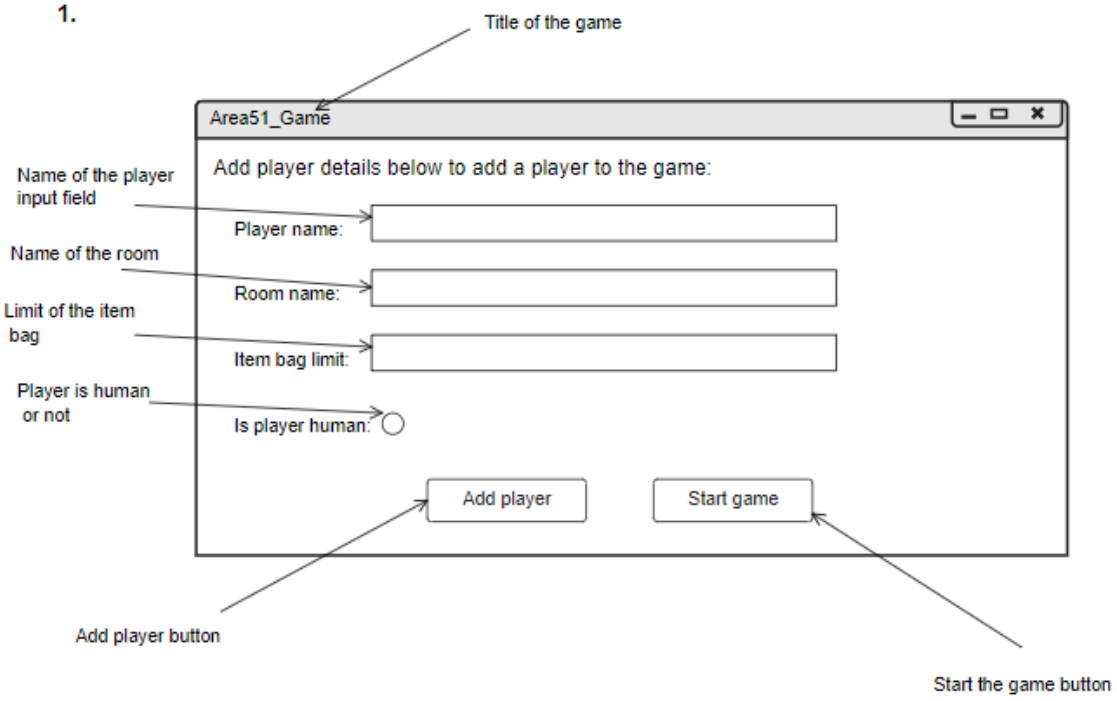
2.

3.

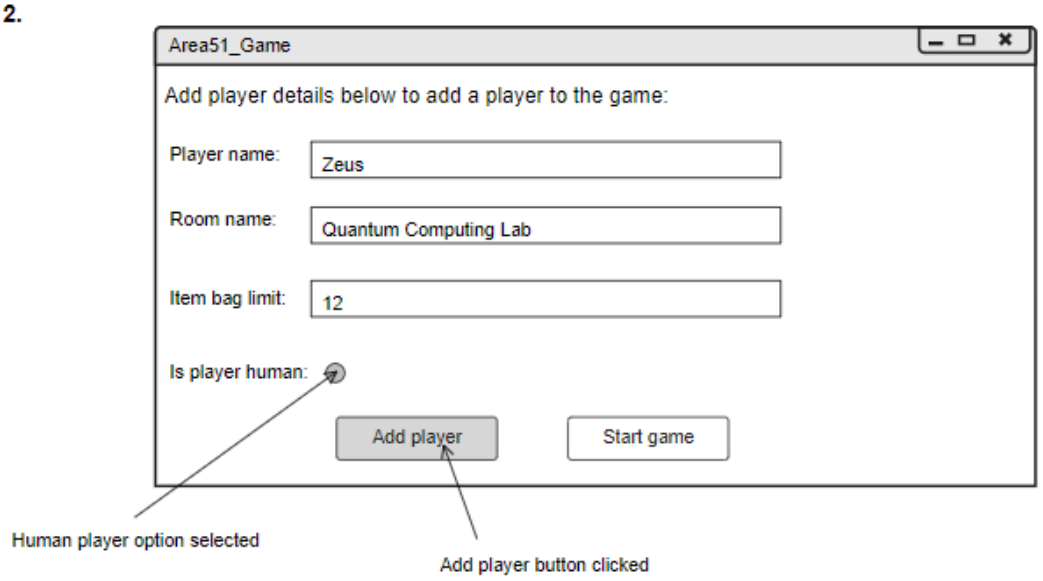


Screen 2- Player add screen

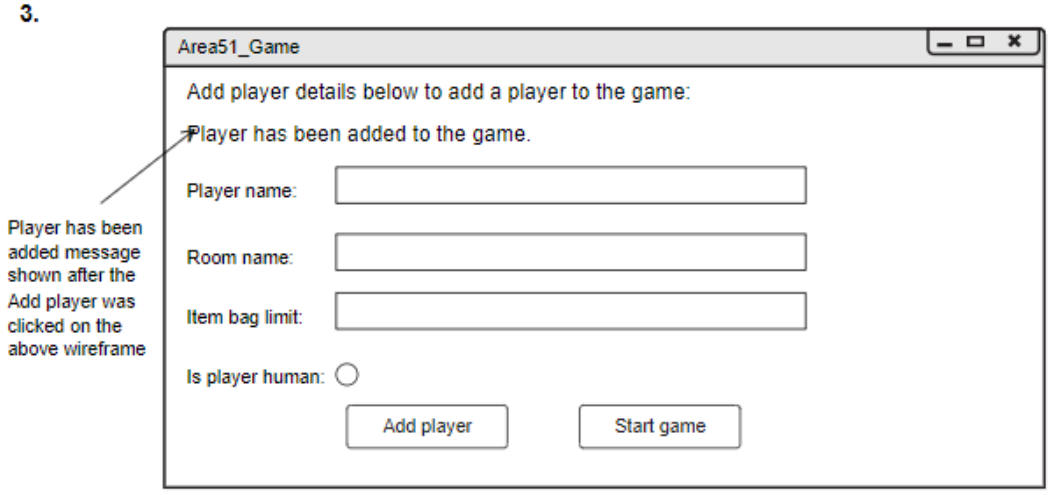
1.



2.

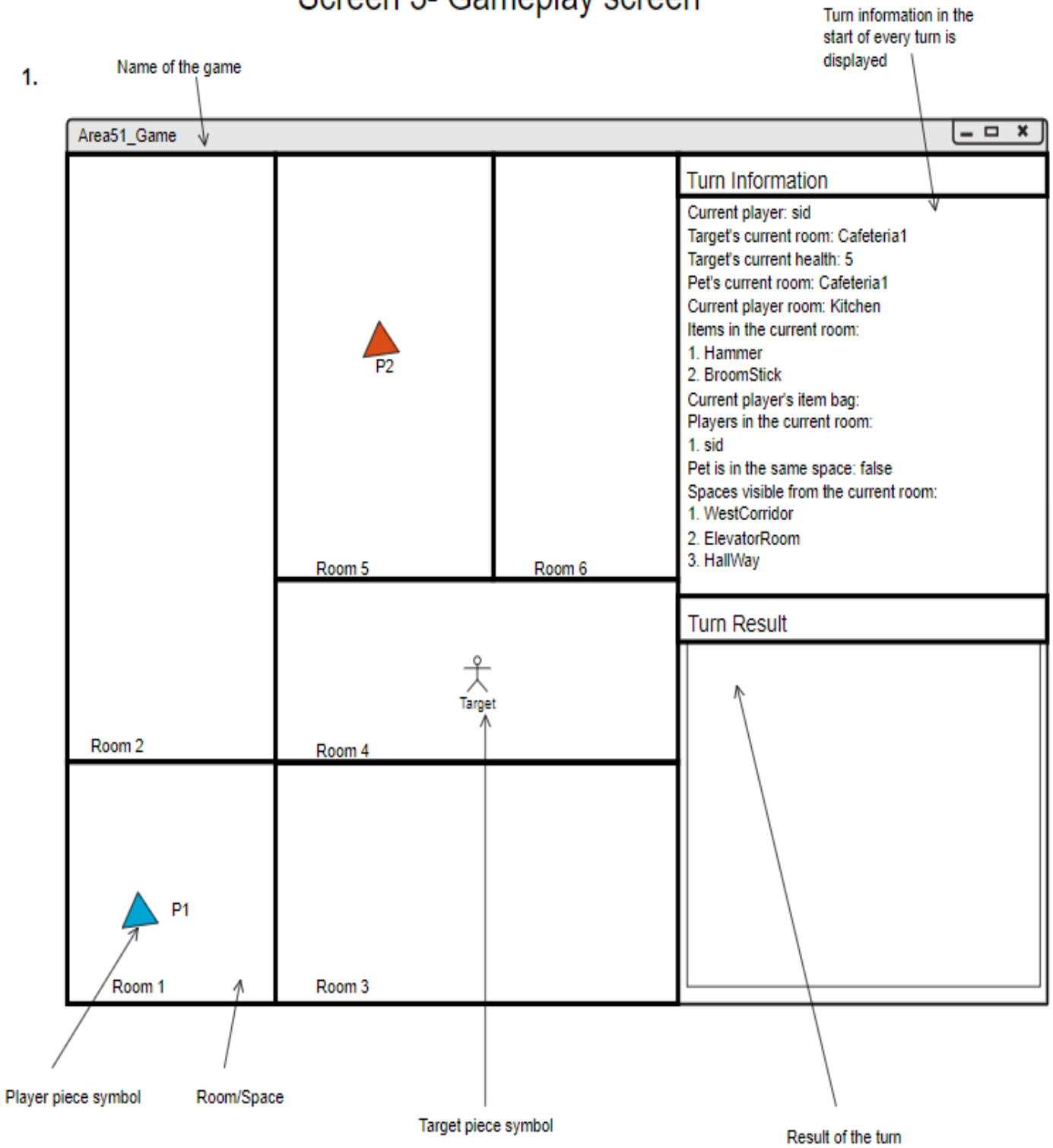


3.

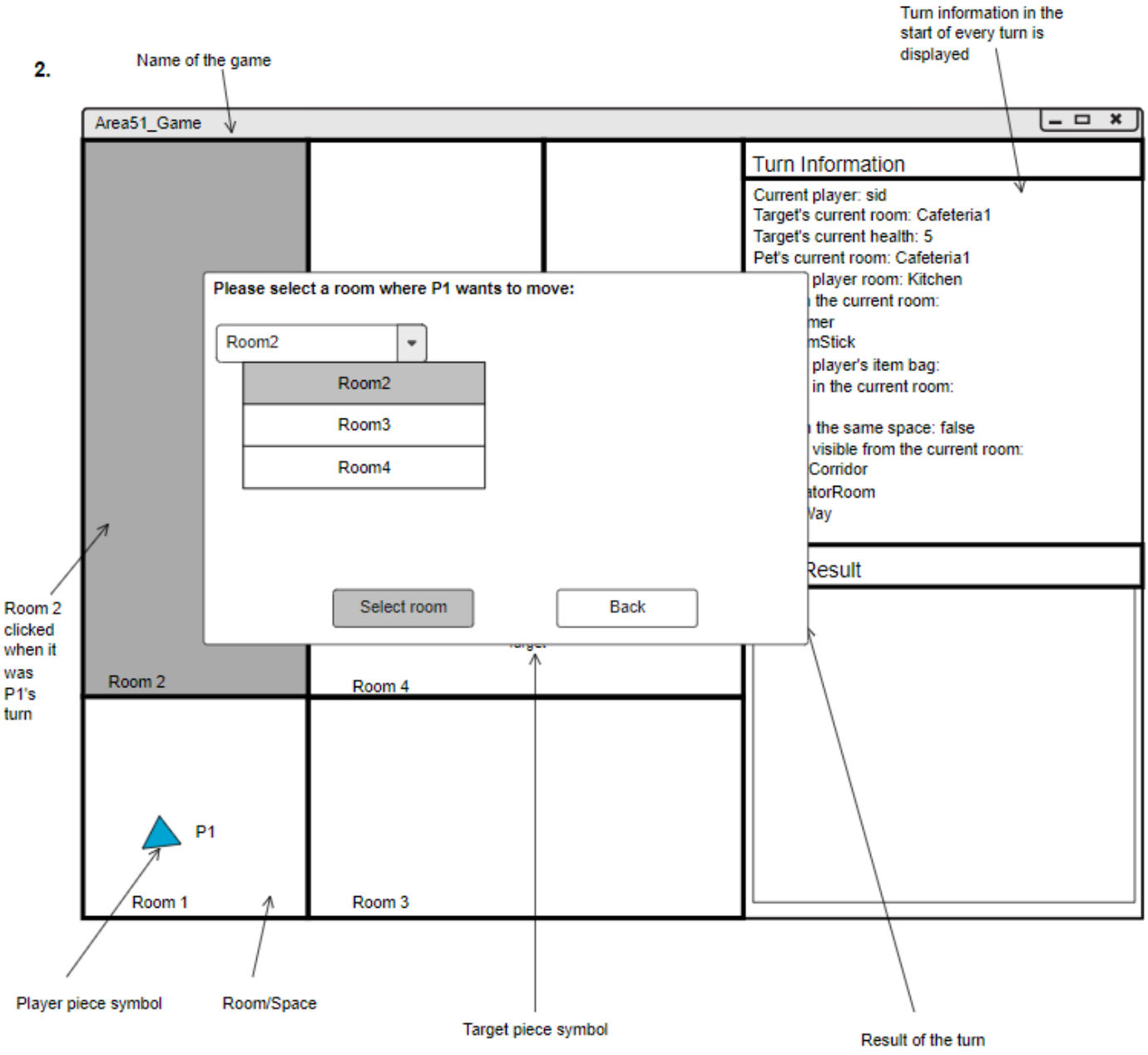


Screen 3- Gameplay screen

1.



2.


Name of the game
Turn information in the start of every turn is displayed
Area51_Game
Turn Information
Current player: sid
Target's current room: Cafeteria1
Target's current health: 5
Pet's current room: Cafeteria1
player room: Kitchen
the current room:
mer
mStick
player's item bag:
in the current room:
the same space: false
visible from the current room:
Corridor
torRoom
Vay
Result
Please select a room where P1 wants to move:
Room2
Room2
Room3
Room4
Select room
Back
Room 2 clicked when it was P1's turn
Room 2
Room 4
P1
Room 1
Room 3
Player piece symbol
Room/Space
Target piece symbol
Result of the turn

MAIN

Driver

+ main(args: String[])

CONTROLLER

<<interface>> Features

+ executeAddPlayerPiece(name: String, roomName: String, itemBagLimit: int, isHuman: boolean): String
+ executeMovePlayerPiece(roomName: String): String
+ executePickItem(item: Item): boolean
+ executePlayerLookAround(): String
+ executePlayerDescription(playerName: String: String
+ executePlayerAttack(itemName: String): String
+ executeMovePet(roomName: String): String
+ generateBaseImage(): BufferedImage
+ notifyAddPlayerPiece(): String
+ notifyMovePlayerPiece(): String
+ notifyPickItem(): boolean
+ notifyPlayerDescription(): String
+ notifyPlayerAttack(): String
+ notifyMovePet(): String
+ restartGame()
+ quitGame()

<<interface>> Controller

+ start()
+ setView(view: View)

ControllerImpl

- world: World
- gamePlayCommandMap: Map<String, Function<String, Command>
- view: View

+ ControllerImpl(world: World)
+ start(): int
+ setView(view: View)
+ executeAddPlayerPiece(name: String, roomName: String, itemBagLimit: int, isHuman: boolean): String
+ executeMovePlayerPiece(roomName: String): String
+ executePickItem(itemName: String): String
+ executePlayerLookAround(): String
+ executePlayerDescription(playerName: String): String
+ executePlayerAttack(itemName: String): String
+ executeMovePet(roomName: String): String
+ generateBaseImage(): BufferedImage
+ notifyAddPlayerPiece(): String
+ notifyMovePlayerPiece(): String
+ notifyPickItem(): String
+ notifyPlayerDescription(): String
+ notifyPlayerAttack(): String
+ notifyMovePet(): String
+ restartGame()
+ quitGame()

uses

CreateGraphicalWorld

+ CreateGraphicalWorld()
+ execute(world: World)

AddPlayerPiece

- name: String
- roomName: String
- itemBagLimit: int
- isHuman: boolean

+ AddPlayerPiece(name: String, roomName: String, itemBagLimit: int, isHuman: boolean): String
+ execute(world: World)

VIEW

View interface is how the Controller will see the view.

renderOverlayPieces will get coordinates of all pieces of board from the controller. The view will simply render the pices at the co-ordinates mentioned by the read only model.

<<interface>> View

+ setFeatures(feature: Feature)
+ renderOverlayPices(List<List<Integer, Integer>>)
+ displayTurnInformation()
+ displayMoveResult()
+ displayPlayerAddAcknowledgement()
+ displayPlayerAddExceptionMessage()
+ displayFileUploadExceptionMessage()
+ displayPlayerLimitReachedMessage()
+ displayPlayerDescription()
+ displayPlayerCarriedItems()
+ displayRoomItems()
+ getCurrentPanelNumber()
+ displayValidNeighbours()

uses

<<interface>> Command

+ execute(world: World)

PlayerAttack

- itemName: String

+ PlayerAttack(itemName : String)
+ execute(world: World)

PlayerMove

- roomName: String

+ PlayerMove(roomName : String)
+ execute(world: World)

<<interface>> GameOverBox

+ renderGameResult()

MovePet

- roomName : String

+ MovePet(roomName: String)
+ execute(world: World)

GetRoomInformation

- roomName: String

+ GetRoomInformation(roomName: string)
+ execute(world: World)

DisplayPlayerDescription

- playerName: String

+ DisplayPlayerDescription(playerName: String)
+ execute(world: World)

PlayerLookAround

+ PlayerLookAround()
+ execute(world: World)

PlayerPickItem

- itemName: String

+ PlayerPickItem(itemName : String)
+ execute(world: World)

ViewImpl

CurrentEventName will be sent to the Controller. On the basis of this name, the controller will decide if it needs additional data for the event (Eg. Item name for attack command). For doing this; a pop up dialogue box will be rendered with a drop down menu of item list.

- currentEventName: String
- moveResult: String
- welcomePanel: WelcomePanel
- addPlayerPanel: AddPlayerPanel
- gamePlayPanel: GamePlayPanel
- frame: JFrame
- currentPanelNumber: int
- model: ReadOnlyWorld
- featuresController: Features

+ ViewImpl(world: ReadOnlyWorld)
+ setFeatures(feature: Features)
- updateCurrentEventName()
+ getCurrentPanelNumber()
+ switchPanels()
+ showWelcomePanel()
+ showInitilizePanel()
+ showGamePlay()

GameOverBoxImpl

- exitButton : JButton
- restartButton : JButton
- gameResultMessage : JLabel

+ GameOverBoxImpl()
- getGameResult()
+ renderGameResult()
- closePopup(actionEvent: ActionEvent)

1 1 1

<<interface>> WelcomePanel

+ setFeatures(feature: Feature)

<<interface>> AddPlayerPanel

+ setFeatures(feature: Feature)

<<interface>> GamePlayPanel

+ setFeatures(feature: Feature)

WelcomePanelImpl

- optionsMenu : JMenu
- menuBar : JMenuBar
- gameTitle : JLabel
- developerName1Label: JLabel
- developerName2Label: JLabel
- developerHeader : JLabel
- CELL_DIM : int = 80
- OFFSET : int = 80
- FONT_SIZE : int = 40
- featuresController: Features

+ WelcomePanelImpl()
+ setFeatures(features: Features)
+ displayFileUploadExceptionMessage()

AddPlayerPanelImpl

- addPlayerButton : JButton
- StartGameButton : JButton
- userInstruction : JLabel
- playerName : JLabel
- inputPlayerName : JTextField
- roomName : JLabel
- inputRoomName : JTextField
- bagLimit : JLabel
- inputBagLimit : JTextField
- playerHuman : JLabel
- inputPlayerHuman : JRadioButton
- CELL_DIM : int = 80
- OFFSET : int = 80
- FONT_SIZE : int = 40
- featuresController: Features

+ AddPlayerPanelImpl()
+ setFeatures(features: Features)
+ displayPlayerAddAcknowledgement()
+ displayPlayerAddExceptionMessage()
+ displayPlayerLimitReachedMessage()

All these display methods will be called from the controller and view will decide how to display a component by appropriately defining these methods.

GamePlayPanelImpl

- CELL_DIM : int = 80
- OFFSET : int = 80
- FONT_SIZE : int = 40
- baseWorldImage : BufferedImage
- overlayPieceButtonList : List<JButton>
- turnInformationTitle : JLabel
- turnInformation : JLabel
- turnResultTitle : JLabel
- turnResult : JLabel
- submitButton : JButton
- featuresController: Features

+ GamePlayPanelImpl()
+ setFeatures(features: Features)
+ renderOverlayPices(List<List<Integer, Integer>)
+ displayTurnInformation()
+ displayMoveResult()
+ displayPlayerDescription()
+ displayPlayerCarriedItemsDialogBox(playerItems : List<String>)
+ displayRoomItemsDialogBox(roomItems : List<String>)
+ displayValidNeighboursDialogBox(validNeighbours : List<String>)

MODEL

<<interface>> ReadOnlyWorld

+ getTurnInformation(): String
+ getRoomsList(): List<Room>
+ getPlayerList(): List<Player>
+ getBaseImage(): BufferedImage
+ getTargetPiece(): Target
+ getOverlayPieceCoordinates(): List<List<Integer, Integer>>
+ getMoveResult(): String
+ getPlayerDescription(): String
+ getOverlayPieceCoordinates(): List<List<Integer, Integer>>

WorldInitializer

- fileReader: Readable
- worldInitializer: WorldInitializer
- area51: World
- maxTurns: int

- WorldInitializer(fileReader: Readable, maxTurns: int)
+ getInstance(fileReader: Readable, maxTurns: int): WorldInitializer
+ generateWorld(): World

initializes

<<interface>> World

+ addPlayerPiece(name: String, roomName: String, itemBagLimit: Integer, isHuman: boolean): String
+ movePlayerPiece(roomName: String): String
+ playerPickUpItem(itemName: String): String
+ playerLookAround(): String
+ getPlayerDescription(playerName: String: String
+ isCurrentTurnHuman(): boolean
+ getPlayerList(): List<Player>
+ getMaxTurns(): Integer
+ getRows(): Integer
+ getColumns(): Integer
+ getNumberOfTurns(): Integer
+ displayRoomInformation(roomName: String): String
+ playComputerPlayer(option: Integer, flagUseRandom: boolean): String
+ playerAttack(itemName: String): String
+ movePet(roomName: String): String
+ updateMoveResult()
+ updateOverlayPieces()
+ getOverlayPieceCoordinates(): List<List<Integer, Integer>>
+ updateInputSpecsFilePath(filePath: String)
+ generateBaseImage()

ImageDisplayer

- bufferedImage: BufferedImage
- graphics: Graphics
- scaleUp: int
- world: World
- roomsList: List<Room>

+ ImageDisplayer(world: World, scaleUp: int)
- paintComponent(g: Graphics)
+ draw()
+ generate(): BufferedImage

uses

GenerateRandom

- randomInt: Integer
- random: Random

+ GenerateRandom()
+ GenerateRandom(randomInt: int)
+ getRandomInt(range: int)

<<interface>> Room

+ calculateNeighborsAndSave(roomsList: List<Room>)
+ getNeighbors(): List<Room>
+ getItemList(): List<Item>
+ getLeftTopRow(): int
+ getLeftTopCol(): int
+ getRoom(): int
+ getRightTopRow(): int
+ getRightTopCol(): int
+ getName(): String
+ addPlayerToRoom(Player player)
+ removePlayerFromRoom(Player player)
+ removeItemFromRoom(item: Item)
+ getPlayersInRoom(): List<Player>
+ statusChangePetInRoom(isInRoom: boolean)
+ getIsPetPresent(): boolean

1..* contains 1

WorldImpl

- rows: Integer
- columns: Integer
- worldName: String
- roomsList: List<Rooms>
- itemList: List<Item>
- playerList: List<Player>
- targetPiece: Player
- maxTurns: Integer
- numberOfTurns: Integer
- currentTurnListIndex: Integer
- pet: Pet
- imageDisplayer: ImageDisplayer
- overlayPieceCoordinates: List<List<Integer, Integer>>
- moveResult: String
- inputSpecsFilePath: String
- baseImage: BufferedImage

+ World(rows:Integer, columns:Integer, worldName:String, roomsList:List<Room>, itemList: List<Item>, targetName:String, targetHealth:Integer, petName: String, maxTurns: Integer)
- doRoomsOverlap(room1: Room, room2: Room): boolean
- areItemsRoomIndicesValid(): boolean
- allRoomsLesserDimThanWorld(): boolean
+ getTargetPiece() : Target
+ getRows(): Integer
+ getRoomsList(): List<Rooms>
+ getCols(): Integer
+ getWorldName(): String
+ toString(): String
+ updateMoveResult()
+ updateInputSpecsFilePath(filepath: String)
+ getTurnInformation(): String
+ addPlayerPiece(name: String, roomName: String, itemBagLimit: int, isHuman: boolean): String
+ movePlayerPiece(roomName: String): String
+ playerPickUpItem(itemName: String): String
+ playerLookAround(): String
+ displayRoomInformation(roomName: String): String
+ playComputerPlayer(option: Integer, flagUseRandom: boolean) : String
+ getPlayerDescription(playerName: String): String
+ getPlayerList(): List<Player>
+ isCurrentTurnHuman(): boolean
+ getMaxTurns(): Integer
+ getNumberOfTurns(): Integer
+ playerAttack(itemName: String): String
- canSeePlayer(player1: Player, player2: Player): Boolean
+ movePet(roomName: String): String
+ getBaseImage(): BufferedImage
+ updateOverlayPieces()
- generateRandomInCustomRange(start:Integer, end:Integer): Integer
+ generateBaseImage()
+ getBaseImage(): BufferedImage
+ getOverlayPieceCoordinates(): List<List<Integer, Integer>>
+ getMoveResult(): String

1 contains 1..*

uses

RoomImpl

- number: int
- name: String
- leftTopRow: int
- leftTopCol: int
- rightBottomRow: int
- rightBottomCol: int
- neighborRooms: List<Room>
- itemsList: List<Item>
- globalItemsList: List<Item>
- globalRoomsList: List<Room>
- playersInRoom: List<Player>
- isPetPresent: boolean

+ Room(number: int, name: String, leftTopRow: int, leftTopCol: int, rightBottomRow: int, rightBottomCol: int, itemList: List<Item>)
+ calculateNeighborsAndSave(roomsList: List<Room>)
- generateItemList(itemList: List<Item>)
+ getNeighbors(): List<Room>
+ getItemList(): List<Item>
+ getLeftTopRow(): int
+ getLeftTopCol(): int
+ getRoom(): int
+ getRightTopRow(): int
+ getRightTopCol(): int
+ getName(): String
+ equals(o: Object): boolean
+ hashCode(): int
+ toString(): String
+ addPlayerToRoom(Player player)
+ removePlayerFromRoom(Player player)
+ removeItemFromRoom(item: Item)
+ getPlayersInRoom(): List<Player>
+ statusChangePetInRoom(isInRoom: boolean)
+ getIsPetPresent(): boolean

stays in 1

<<interface>> Pet

+ getCurrentRoom(): Room
+ getName(): String
+ movePet(room: Room): String
+ movePetDFS(): Room

PetImpl

- name: String
- currentRoom: Room
- roomsList: List<Room>

+ PetImpl(name: String, currentRoom: Room, roomsList: List<Room>)
+ getCurrentRoom(): Room
+ getName(): String
+ movePet(room: Room): String
+ movePetDFS(): Room

1 contains 1

<<interface>> Player

+ getCurrentRoom(): Room
+ getName(): String
+ getNumber(): int
+ getIsHuman(): boolean
+ getItemList(): List<Item>
+ pickItem(item: Item): boolean
+ getSpiteLevel(): int
+ getItemBagLimit(): int
+ lookAround(): String
+ move(room: Room): String
+ removeItemFromBag(item: Item): boolean

contains 1..*

<<interface>> Target

+ getCurrentRoom(): Room
+ getName(): String
+ getNumber(): int
+ getHealth(): int
+ move()
+ receiveAttack(damage: int)

TargetImpl

- name: String
- number: int
- currentRoom: Room
- roomsList: List<Room>
- health: int

+ Target(number: int, name: String, health: int, roomsList: List<Room>)
+ getHealth(): int
+ getCurrentRoom(): Room
+ move()
+ getName(): String
+ getNumber(): int
+ toString(): String
+ receiveAttack(damage: int)

contains

PlayerImpl

- itemList: List<Item>
- currentRoom: Room
- spiteLevel: int
- itemBagLimit: int
- isHuman: boolean
- name: String
- number: int

+ Player(number: int, name: String, currentRoom: Room, itemBagLimit: int, isHuman: boolean)
+ getItemList(): List<Item>
+ getCurrentRoom(): Room
+ pickItem(item: Item): boolean
+ getSpiteLevel(): int
+ lookAround(): String
+ getIsHuman(): boolean
+ getItemBagLimit(): int
+ move(room: Room): String
+ getName(): String
+ getNumber(): int
+ toString(): String
+ removeItemFromBag(item: Item): boolean

1 1

contains 0..* 1

<<interface>> Item

+ getContainingRoomIndex(): int
+ getName(): String
+ getId(): int
+ getDamage(): int

0..* carries 0..*

ItemImpl

- id: int
- name: String
- damage: int
- containingRoomIndex: int

+ Item(id: int, name: String, damage: int, containingRoomIndex: int)
+ getContainingRoomIndex(): int
+ getName(): String
+ getId(): int
+ getDamage(): int
+ toString(): String

uses

TESTING PLAN FOR MILESTONE 4

Note: **The below tests are Milestone 4 specifically**, the tests for Milestone 3, 2 and 1 are followed post this test plan for Milestone 4.

MODEL Test

Testing getMoveResult()	**Input**	**Expected Value**
Testing if we get correct string representation denoting the status of the move that was made e.g. playerAttack.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.playerAttack(10)	Correct string representation returned denoting the attack status and target’s current health.

Testing updateInputSpecsFilePath()	**Input**	**Expected Value**
Testing if updateInputSpecsFilePath() actually sets the inputSpecsFilePath field to the passed value.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.updateInputSp ecsFilePath(<file_path>)	The inputSpecsFilePath field is updated to the value sent.
Testing if it throws IllegalArgumentException when filePath string is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.updateInputSp ecsFilePath(“”)	IllegalArgumentException is thrown.

Testing if it throws IllegalArgumentException when filePath string is null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.updateInputSpecsFilePath(null)	IllegalArgumentException is thrown.

Testing updateOverlayPieces()	**Input**	**Expected Value**
Testing if updateOverlayPieces actually sets the correct values for the overlayPieceCooridinates field.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.updateOverlayPieceCoordinates()	The overlayPieceCoordinates field is set to appropriate values where for each player, their coordinates lie within the boundaries of their current room.

Testing getOverlayPieceCoordinates()	**Input**	**Expected Value**
Testing if getOverLayPiece coordinates actually return the correct set of values from the overlayPieceCooridinates field.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.updateOverlayPieceCoordinates() worldImpl.getOverlayPieceCoordinates()	The overlayPieceCoordinates field is set to appropriate values where for each player, their coordinates lie within the boundaries of their current room. This function returns the valid set of outputs for each player’s coordinates.

Testing generateBaseImage()	**Input**	**Expected Value**
Testing if generateBaseImage() actually generates a correct representation of the spaces/rooms in the world.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”,	Correct representation of the spaces/rooms in the world is stored as a BufferedImage in the baseImage field.

	100, 1) worldImpl.generateBaseImage()	

Testing getBaseImage()	**Input**	**Expected Value**
Testing if getBaseImage() actually returns the correct representation of the spaces/rooms in the world.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) worldImpl.generateBaseImage()	The correct representation of the spaces/rooms in the world is stored as a BufferedImage in the baseImage field. This function returns the value stored in the baseImage field of the world model.

CONTROLLER Test

Note: We are going to make use of a mock model and a mock view here to test the controller in an isolated way. The mock model implements the World interface, same as what our model (WorldImpl) does. The mock view implements the View interface, same as that out view (ViewImpl) does.

1. ControllerImpl Test

Testing Construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the model world passed is null.	ControllerImpl(null)	IllegalArgumentException is thrown.

Testing valid construction when model passed is not null.	ControllerImpl(mockWorld)	Object created, hashCode() returns a value.

Testing start()	**Input**	**Expected Value**
Testing if start functions returns the next panel number when the view calls it where start will get the current panel number from the view and return an incremented value telling the view to show the next panel.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.start()	Returns the next panel number when the view calls it where start will get the current panel number from the view and return an incremented value telling the view to show the next panel. We call here controller Impl to simulate the call from view here.

Testing setView()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the view passed is null.	ControllerImpl(mockWorld) controllerImpl.setView(null)	IllegalArgumentException is thrown.
Testing valid construction when the model passed is not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView)	Function works as expected and the mock view passed is initialised as controller's view field.

Testing executeAddPlayerPiece()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the name is null.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(null, "Lab",	IllegalArgumentException is thrown.

	12, true)	
Testing if IllegalArgumentException is thrown when the name is an empty string.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“”, “Lab”, 12, true)	IllegalArgumentException is thrown.
Testing valid String is returned when name is not null and everything else is valid.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“yeti”,”Lab”, 12, true)	Valid string returned denoting mockWorld’s function was called with the given parameters.
Testing if IllegalArgumentException is thrown when the room name is null.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“yeti”, null, 12, true)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“yeti”,””, 12, true)	IllegalArgumentException is thrown.
Testing valid String is returned when room name is not null and everything else is valid.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“yeti”, “Lab”, 12, true)	Valid string returned denoting mockWorld’s function was called with the given parameters.
Testing if IllegalArgumentException is thrown when itemBagLimit is <=0.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“yeti”, “Lab”, -1, true)	IllegalArgumentException is thrown.
Testing valid String is returned when itemBagLimit > 0 and everything else is valid.	ControllerImpl(mockWorld) controllerImpl.executeAddPlayerPiece(“yeti”, “Lab”, 12, true)	Valid string returned denoting mockWorld’s function was called with the given parameters.
Testing valid string is returned when all parameters are valid.	ControllerImpl(mockWorld)	Valid string returned denoting mockWorld’s function was called with

	controllerImpl.executeAddPlayerPiece(“yeti”, “Lab”, 12, true)	the given parameters.

Testing executeMovePlayerPiece()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the room name is null.	ControllerImpl(mockWorld) controllerImpl.executeMovePlayerPiece(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	ControllerImpl(mockWorld) controllerImpl.executeMovePlayerPiece(“”)	IllegalArgumentException is thrown.
Testing valid String is returned when room name is not null and everything is valid.	ControllerImpl(mockWorld) controllerImpl.executeMovePlayerPiece(”Lab”)	Valid string returned denoting mockWorld’s function was called with the given parameters.

Testing executePickItem()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the item name is null.	ControllerImpl(mockWorld) controllerImpl.executePickItem(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the item name is an empty string.	ControllerImpl(mockWorld)	IllegalArgumentException is thrown.

	controllerImpl.executePickItem(“”)	
Testing valid String is returned when item name is not null and everything is valid.	ControllerImpl(mockWorld) controllerImpl.executePickItem(“Laser gun”)	Valid string returned denoting mockWorld’s function was called with the given parameters.

Testing executePlayerLookAround()	**Input**	**Expected Value**
Testing valid String is returned when denoting the function was called in the mock model.	ControllerImpl(mockWorld) controllerImpl.executePlayerLookAround()	Valid string returned denoting mockWorld’s function was called with the given parameters.

Testing executePlayerDescription()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the player name is null.	ControllerImpl(mockWorld) controllerImpl.executePlayerDescription(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the player name is an empty string.	ControllerImpl(mockWorld) controllerImpl.executePlayerDescription(“”)	IllegalArgumentException is thrown.
Testing valid String is returned when the player name is not null and everything is valid.	ControllerImpl(mockWorld) controllerImpl.executePlayerDescription(“sid”)	Valid string returned denoting mockWorld’s function was called with the given parameters.

Testing executePlayerAttack()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the item name is null.	ControllerImpl(mockWorld) controllerImpl.executePlayerAttack(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the item name is an empty string.	ControllerImpl(mockWorld) controllerImpl.executePlayerAttack(“”)	IllegalArgumentException is thrown.
Testing valid String is returned when item name is not null and everything is valid.	ControllerImpl(mockWorld) controllerImpl.executePlayerAttack(“Laser gun”)	Valid string returned denoting mockWorld’s function was called with the given parameters.

Testing executeMovePet()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the room name is null.	ControllerImpl(mockWorld) controllerImpl.executeMovePet(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	ControllerImpl(mockWorld) controllerImpl.executeMovePet(“”)	IllegalArgumentException is thrown.
Testing valid String is returned when room name is not null and everything is valid.	ControllerImpl(mockWorld) controllerImpl.executeMo	Valid string returned denoting mockWorld’s function was called with the given parameters.

	vePet(”Lab”)	

Testing generateBaseImage()	**Input**	**Expected Value**
Testing if call to generateBaseImage returns a string denoting the mock model’s generateBaseImage() was called.	ControllerImpl(mockWorld) controllerImpl.generateBaseImage()	Valid string returned denoting mockWorld’s generateBaseImage() function was called.

Testing notifyAddPlayerPiece()	**Input**	**Expected Value**
Testing if the call to notifyAddPlayerPiece simulates the mockView notifying the controller that add player piece functionality was requested by the user and a String is returned from this function denoting the next action that the view is supposed to take.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.notifyAddPlayerPiece()	String is returned from this function denoting the next action that the view (mock view here) is supposed to take.

Testing notifyMovePlayerPiece()	**Input**	**Expected Value**
Testing if the call to notifyMovePlayerPiece simulates the mockView notifying the controller that move player piece functionality was requested by the user and a String is returned from this function denoting the next action that the view is supposed to take.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.notifyMovePlayerPiece()	String is returned from this function denoting the next action that the view (mock view here) is supposed to take.

Testing notifyPickItem()	**Input**	**Expected Value**
Testing if the call to notifyPickItem simulates the mockView notifying the controller that pick item functionality was requested by the user and a String	ControllerImpl(mockWorld) controllerImpl.setView(mo	String is returned from this function denoting the next action that the view (mock view here) is supposed to take.

is returned from this function denoting the next action that the view is supposed to take.	ckView) controllerImpl.notifyPickItem()	

Testing notifyPlayerDescription()	**Input**	**Expected Value**
Testing if the call to notifyPlayerDescription simulates the mockView notifying the controller that the get player description functionality was requested by the user and a String is returned from this function denoting the next action that the view is supposed to take.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.notifyPlayerDescription()	String is returned from this function denoting the next action that the view (mock view here) is supposed to take.

Testing notifyPlayerAttack()	**Input**	**Expected Value**
Testing if the call to notifyPlayerAttack simulates the mockView notifying the controller that player attack functionality was requested by the user and a String is returned from this function denoting the next action that the view is supposed to take.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.notifyPlayerAttack()	String is returned from this function denoting the next action that the view (mock view here) is supposed to take.

Testing notifyMovePet()	**Input**	**Expected Value**
Testing if the call to notifyMovePet simulates the mockView notifying the controller that the move pet functionality was requested by the user and a String is returned from this function denoting the next action that the view is supposed to take.	ControllerImpl(mockWorld) controllerImpl.setView(mockView)	String is returned from this function denoting the next action that the view (mock view here) is supposed to take.

	controllerImpl.notifyMovePet()	

Testing restartGame()	**Input**	**Expected Value**
Testing if call to restartGame() actually re-initializes the state of the mockView and mockModel.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.restartGame()	Re-initialises the state of the mock model, mock view and the game as a whole.

Testing quitGame()	**Input**	**Expected Value**
Testing if the call to quitGame() actually shuts down the code with a normal exit code.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) controllerImpl.quitGame()	Shuts down the code with a normal exit code.

2. CreateGraphicalWorld Test

Testing construction()	**Input**	**Expected Value**
Testing valid construction.	CreateGraphicalWorld()	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**

Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) CreateGraphicalWorld() createGraphicalWorld.execute(mockModel)	Calls the mock model's generateBaseImage.
Testing if IllegalArgumentException is thrown when the world model passed is null.	CreateGraphicalWorld() createGraphicalWorld.execute(null)	IllegalArgumentException is thrown.

3. AddPlayerPiece Test

Testing construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the name is null.	AddPlayerPiece(null, "QuatumComputingLab", 12, true)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the name is an empty string.	AddPlayerPiece("", "QuatumComputingLab", 12, true)	IllegalArgumentException is thrown.
Testing valid construction when name is not null and everything else is valid.	AddPlayerPiece("sid", "QuatumComputingLab", 12, true)	Object created, hashCode() returns a value.
Testing if IllegalArgumentException is thrown when the room name is null.	AddPlayerPiece("sid", null, 12, true)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	AddPlayerPiece("sid", "", 12, true)	IllegalArgumentException is thrown.
Testing valid construction when room name is not null and everything else is valid.	AddPlayerPiece("sid", "QuatumComputingLab", 12, true)	Object created, hashCode() returns a value.
Testing if IllegalArgumentException is thrown when itemBagLimit is <=0.	AddPlayerPiece("sid", "QuatumComputingLab", -1, true)	IllegalArgumentException is thrown.

Testing valid construction when itemBagLimit > 0 and everything else is valid.	AddPlayerPiece(“sid”, “QuatumComputingLab”, 12, true)	Object created, hashCode() returns a value.
Testing valid construction when all parameters are valid.	AddPlayerPiece(“sid”, “QuatumComputingLab”, 12, true	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) AddPlayerPiece(“sid”, “QuatumComputingLab”, 12, true) addPlayerPiece.execute(mockModel)	Calls the mock model’s addPlayerPiece() function.
Testing if IllegalArgumentException is thrown when the world model passed is null.	AddPlayerPiece(“sid”, “QuatumComputingLab”, 12, true) addPlayerPiece.execute(null)	IllegalArgumentException is thrown.

4. PlayerAttack Test

Testing construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the item name is null.	PlayerAttack(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the item name is an empty string.	PlayerAttack(“”)	IllegalArgumentException is thrown.

Testing valid construction when item name is not null and everything else is valid.	PlayerAttack(“Laser Gun”)	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) PlayerAttack(“Laser Gun”) addPlayerPiece.execute(mockModel)	Calls the mock model’s playerAttack() function.
Testing if IllegalArgumentException is thrown when the world model passed is null.	PlayerAttack(“Laser Gun”) addPlayerPiece.execute(null)	IllegalArgumentException is thrown.

5. PlayerMove Test

Testing construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the room name is null.	PlayerMove(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	PlayerMove(“”)	IllegalArgumentException is thrown.
Testing valid construction when room name is not null and everything else is valid.	PlayerMove(“QuantumComputingLab”)	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld)	Calls the mock model’s movePlayerPiece() function.

	controllerImpl.setView(mockView) PlayerMove(“QuantumComputingLab”) playerMove.execute(mockModel)	
Testing if IllegalArgumentException is thrown when the world model passed is null.	PlayerMove(“QuantumComputingLab”) playerMove.execute(null)	IllegalArgumentException is thrown.

6. PlayerPickItem Test

Testing construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the item name is null.	PlayerPickItem(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the item name is an empty string.	PlayerPickItem(“”)	IllegalArgumentException is thrown.
Testing valid construction when item name is not null and everything else is valid.	PlayerPickItem(“Laser Gun”)	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) PlayerPickItem(“Laser Gun”)	Calls the mock model’s playerPickUpItem() function.

	playerPickItem.execute(mockModel)	
Testing if IllegalArgumentException is thrown when the world model passed is null.	PlayerPickItem(“Laser Gun”) playerPickItem.execute(null)	IllegalArgumentException is thrown.

7. PlayerLookAround Test

Testing construction()	**Input**	**Expected Value**
Testing object creation when the constructor called in a valid way.	PlayerLookAround()	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) PlayerLookAround() playerLookAround.execute(mockModel)	Calls the mock model’s playerLookAround() function.
Testing if IllegalArgumentException is thrown when the world model passed is null.	PlayerLookAround() playerLookAround.execute(null)	IllegalArgumentException is thrown.

8. DisplayPlayerDescription Test

Testing construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the player name is null.	DisplayPlayerDescription(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the player name is an empty string.	DisplayPlayerDescription(“”)	IllegalArgumentException is thrown.
Testing valid construction when the player name is not null and everything else is valid.	DisplayPlayerDescription(“sid”)	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) DisplayPlayerDescription(“sid”) displayPlayerDescription. execute(mockModel)	Calls the mock model’s getPlayerDescription() function.
Testing if IllegalArgumentException is thrown when the world model passed is null.	DisplayPlayerDescription(“sid”) displayPlayerDescription. execute(null)	IllegalArgumentException is thrown.

9. GetRoomDescription Test

Testing construction()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the room name is null.	GetRoomDescription(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	GetRoomDescription(“”)	IllegalArgumentException is thrown.
Testing valid construction when the room name is not null and everything else is valid.	GetRoomDescription(“QuantumComputingLab”)	Object created, hashCode() returns a value.

Testing execute()	**Input**	**Expected Value**
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) GetRoomDescription(“sid”) getRoomDescription.execute(mockModel)	Calls the mock model’s displayRoomDescription() function.
Testing if IllegalArgumentException is thrown when the world model passed is null.	GetRoomDescription(“sid”) getRoomDescription.execute(nulll)	IllegalArgumentException is thrown.

10. MovePet Test

Testing construction()	Input	Expected Value
Testing if IllegalArgumentException is thrown when the room name is null.	MovePet(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when the room name is an empty string.	MovePet(“”)	IllegalArgumentException is thrown.
Testing valid construction when the room name is not null and everything else is valid.	MovePet(“QuantumComputingLab”)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing when the world model passed is valid and not null.	ControllerImpl(mockWorld) controllerImpl.setView(mockView) MovePet(“HallWay”) movePet.execute(mockModel)	Calls the mock model’s movePet() function.
Testing if IllegalArgumentException is thrown when the world model passed is null.	MovePet(“HallWay”) movePet.execute(null)	IllegalArgumentException is thrown.

View Test

1. ViewImpl Test

Testing construction()	Input	Expected Value
Testing if IllegalArgumentException is thrown when the readOnlyWorld model is null.	ViewImpl(null)	IllegalArgumentException is thrown.
Testing valid construction when the readOnlyWorld model is not null and everything is valid.	ViewImpl(readOnlyWorld)	Object created, hashCode() returns a value.

Testing setFeatures()	Input	Expected Value
Testing if IllegalArgumentException is thrown when the features controller is null.	ViewImpl(readOnlyWorld) viewImpl.setFeatures(null)	IllegalArgumentException is thrown.
Testing valid construction when the features controller is not null and everything is valid.	viewImpl.setFeatures(featureController)	featuresController field of the object is set.

2. GameOverBoxImpl Test

Testing construction()	Input	Expected Value
Testing valid construction.	GameOverBoxImpl()	Object created, hashCode() returns a value.

3. WelcomePanelImpl Test

Testing construction()	Input	Expected Value
Testing valid construction.	WelcomePanelImpl()	Object created, hashCode() returns a value.

Testing setFeatures()	Input	Expected Value
Testing if IllegalArgumentException is thrown when the features controller is null.	WelcomePanelImpl() welcomePanelImpl.setFeatures(null)	IllegalArgumentException is thrown.
Testing valid construction when the features controller is not null and everything is valid.	WelcomePanelImpl() welcomePanelImpl.setFeatures(featureController)	featuresController field of the object is set.

4. AddPlayerPanelImpl Test

Testing construction()	Input	Expected Value
Testing valid construction.	AddPlayerPanelImpl()	Object created, hashCode() returns a value.

Testing setFeatures()	Input	Expected Value
Testing if IllegalArgumentException is thrown when the features controller is null.	AddPlayerPanelImpl() addPlayerPanelImpl.setFeatures(null)	IllegalArgumentException is thrown.
Testing valid construction when the features controller is not null and everything is valid.	AddPlayerPanelImpl() addPlayerPanelImpl.setFeatures(featureController)	featuresController field of the object is set.

5. GamePlayPanelImpl Test

Testing construction()	Input	Expected Value
Testing valid construction.	GamePlayPanelImpl()	Object created, hashCode() returns a value.

Testing setFeatures()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when the features controller is null.	GamePlayPanelImpl() gamePlayPanelImpl.setFeatures(null)	IllegalArgumentException is thrown.
Testing valid construction when the features controller is not null and everything is valid.	GamePlayPanelImpl() gamePlayPanelImpl.setFeatures(featureController)	featuresController field of the object is set.

TESTING PLAN FOR MILESTONE 3 SPECIFIC FEATURES

Note: The below tests are for Milestone 3 specific features; milestone 1 and 2 testing is followed post the test plan for milestone 3.

MODEL Test

1. WorldImpl

Testing construction()	Input	Expected Value
Testing construction whether it throws IllegalArgumentException when petName == null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, null, 100, 1)	IllegalArgumentException is thrown.
Testing construction whether it throws IllegalArgumentException when petName is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “”, 100, 1)	IllegalArgumentException is thrown.
Testing construction when petName is valid.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1)	Object created, hashCode() returns a value.
Testing construction when target health passed is 0.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 0, 1)	IllegalArgumentException is thrown.

Testing construction when target health passed is negative.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, -1, 1)	IllegalArgumentException is thrown.

Testing getTurnInformation()	**Input**	**Expected Value**
Testing whether it throws IllegalStateException when numberOfTurns >= maxTurns of the game.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, true) world.playerLookAround() world.playerLookAround() world.getTurnInformation()	IllegalStateException is thrown.
Testing if correct String value is returned for a human player turn after a human player turn.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, true) world.addPlayerPiece(Name, roomName, itemBagLimit, true) world.getTurnInformation()	Expected value is returned for a human player turn after a human player turn.
Testing whether correct information about the target’s current location and	WorldImpl(21, 22, “Area51”,	Expected value is returned with information about the target’s current

the information about the current room and its items is returned.	<List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.addPlayerPiece(Name2, roomName2, itemBagLimit2, true) world.getTurnInformation() 	location and the information about the current room and its items.
Testing whether getTurnInformation displays correct target health value after target receives an attack.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) TargetImpl(0,”alien”, 100, <List of rooms>) world.attack(“poke”) world.getTurnInformation() 	Correct string representation is returned reflecting the damage made by the attack on the target.
Testing whether getTurnInformation does not displays the neighbour which have the pet in the start of every turn.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.getTurnInformation() 	Correct string representation is returned which skips the neighbour room of the current room that has the pet while displaying neighbour rooms at the start of every turn.
Testing whether getTurnInformation displays correct pet’s room after the pet moves.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.movePet(<room_name>) world.getTurnInformation() 	Correct string representation that displays correct pet’s room after the pet moves.

Testing whether getTurnInformation shows game ending when target health becomes <= 0 and shows the correct winner with human-human players.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 1) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.addPlayerPiece(Name2, roomName2, itemBagLimit2, true) world.attack(“poke”) world.getTurnInformation()	IllegalStatException is thrown.
Testing whether getTurnInformation shows game ending when target health becomes <= 0 and shows the correct winner with human-computer players.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 1) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.addPlayerPiece(Name2, roomName2, itemBagLimit2, false) world.attack(“poke”) world.getTurnInformation()	IllegalStatException is thrown..
Testing whether getTurnInformation does not display the item that was used by the player in it’s item bag after that item was used for an attack by the player during the turn of the player.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.pickItem(<item_name>	String returned which gives correct information about the turn and does not display the item that was used by player1.

	world.getTurnInformation() world.attack(<item_name>) world.getTurnInformation()	
Testing whether getTurnInformation shows game ending when Target health becomes 0 and attack is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 10) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.attack(“poke”) world.getTurnInformation()	IllegalStateException is thrown.
Testing whether getTurnInformation shows game ending when Target health becomes 0 and movePet is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 10) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.attack(“poke”) world.movePet(<room_name> world.getTurnInformation()	IllegalStateException is thrown.
Testing whether getTurnInformation shows game ending when Target health becomes 0 and playerLookAround is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 10) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true)	IllegalStateException is thrown.

	world.attack(“poke”) world.playerLookAround() world.getTurnInformation() 	
Testing whether getTurnInformation shows game ending when Target health becomes 0 and playerMove is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 10) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.attack(“poke”) world.playerMove(<room_name> world.getTurnInformation() 	IllegalStateException is thrown.
Testing whether getTurnInformation shows game ending when Target health becomes 0 and playerPickItem is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 1, 10) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.attack(“poke”) world.playerPickItem(<item_name> world.getTurnInformation() 	IllegalStateException is thrown.

Testing playComputerPlayer()	**Input**	**Expected Value**
Testing for the computer to use the most damaging item it has and attacking as a first move when the target character is in the same space and they cant be seen by other players.	WorldImpl(21, 22, “Area51”,<List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”,	Computer attempts an attack on the Target character with the most damaging item it has. The item is removed from the world,

Testing it here as getTurnInformation() actually calls and coordinates the call to playComputerPlayer()	100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, false) world.getTurnInformation() 	the room and the computer player's bag. Health of the target decreases.
Testing if poke attack is used if no item in the bag and the computer player cannot be seen by any other player and target in the same room.	WorldImpl(21, 22, "Area51",<List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, false) world.getTurnInformation() 	String representing that poke attack was used on the target and target's health decreases by 1.
Testing if first move is always attack if target in the same room and no other players can see the computer player.	WorldImpl(21, 22, "Area51",<List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, false) world.getTurnInformation() 	String representing that the computer player attacked and created damage on target's health.
Testing if attack is always made when the pet and the target are in the same room as the computer player.	WorldImpl(21, 22, "Area51",<List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, false) world.getTurnInformation() 	String representing that the computer player attacked and created damage on target's health.

Testing for game ending when target health becomes 0 because of the computer player attack.	WorldImpl(21, 22, “Area51”,<List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, false) world.getTurnInformation() world.getTurnInformation()	IllegalStateException with string message that game has ended already and cpu player is the winner.
Testing if item is removed from the world after it was used for an attempt by the computer player’s attack.	WorldImpl(21, 22, “Area51”,<List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(Name, roomName, itemBagLimit, false) world.getTurnInformation() world.getTurnInformation()	Item is not present with the player after use and is depicted in the string from getTurnInformation().

Testing playerLookAround()	**Input**	**Expected Value**
Testing whether PlayerLookAround() gives a correct string of neighbour rooms that can be seen from the current room the player is currently in.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerLookAround()	String consisting of neighbour rooms, information about its current space, items in it and players in it. Also, it would omit displaying information on a room that is occupied by the pet if such a room is a neighbour of the current room.
Testing playerLookAround() when the pet is in one of the neighbouring rooms.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>,	String consisting of neighbour rooms except the one that has the pet in it, information about its current space, items in it and players in it.

	“YetiAlien”, “yeti”, 100, 30) world.playerLookAround()	
Testing playerLookAround() when the pet is in none of the neighbouring rooms.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerLookAround()	String consisting of neighbour rooms information about its current space, items in it and players in it.
Testing target moves after playerLookAround().	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerLookAround()	Target moves and current room index is 1 greater than before in a cyclic manner.
Testing when MaxTurns is reached already and playLookAround is called as the next move.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.playerLookAround()	IllegalStateException is thrown.

Testing displayRoomInformation()	**Input**	**Expected Value**
Testing displayRoomInformation when room name == null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(null)	IllegalArgumentException is thrown.
Testing displayRoomInformation when room name is an empty String.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”,	IllegalArgumentException is thrown.

	100, 30) world.displayRoomInformation(“”)	
Testing displayRoomInformation when room name is valid and pet is present in the room.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(<valid_room_name>)	Returns proper and expected String representation for the room as same as toString() of the room. The string representation shows pet’s presence in the room as true.
Testing displayRoomInformation when room name is valid and pet is not present in the room.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(<valid_room_name>)	Returns proper and expected String representation for the room as same as toString() of the room. The string representation shows pet’s presence in the room as false.
Testing displayRoomInformation when the room with that name is not present in the world.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(<invalid_room_name>)	IllegalArgumentException is thrown.

Testing playerAttack()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when itemName is null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(null)	IllegalArgumentException is thrown.

Testing if IllegalArgumentException is thrown when itemName is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“”)	IllegalArgumentException is thrown.
Testing if a poke attack from the player reduces the target’s health.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) h1 = target.getHealth() world.playerAttack(“Laser Gun”) h2 = target.getHealth()	h1=h2-1
Testing attack fails when an attack is made but was seen by other players.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	String representing attack failure as it was seen by other players.
Testing if an attack from the player using a specific item actually reduces the target’s health by that amount.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) h1 = target.getHealth() world.playerAttack(“Laser Gun”) h2 = target.getHealth()	h1>h2 and the decrease in health of the target is equal to the damage of the item used.
Testing if an attack from the player using a specific item removes the item from the player’s item bag.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	Item removed from the player’s item bag.

Testing if an attack from the player using a specific itemName such that the item does not exist in the player's bag throws IllegalArgumentException.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.playerAttack(<invalid_item_name>)	IllegalArgumentException is thrown.
Testing if an attack from the player using a specific itemName such that the item does not exist in the world throws IllegalArgumentException.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.playerAttack(<invalid_item_name)	IllegalArgumentException is thrown.
Testing if the game ends when an attack kills the target.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.playerAttack("Laser Gun")	IllegalStateException representing game ending.
Testing if the game ends after multiple attacks on the target and kills it. Since IllegalStateException is thrown here, we would catch the exception and get its message to see if the correct winner is announced in the string and the game has ended.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.addPlayerPiece(Name1, roomName1, itemBagLimit1, true) world.addPlayerPiece(Name2, roomName2, itemBagLimit2, true) world.playerAttack("Laser Gun") world.playerAttack("Knife") world.getTurnInformation()	IllegalStateException is thrown.

Testing for a successful attack when the pet is in the same room as the player and the target as the pet makes the room invisible to the other players in neighbouring rooms.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	String representing attack was successful.
Testing for an unsuccessful attack when the pet is in the same room as the player and the target but there is another player in the room.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	String representing unsuccessful attempt as the attempt was seen by other players.
Testing target moves to the next room after playerAttack().	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	Target moves and current room index is 1 greater than before in a cyclic manner.
Testing when MaxTurns is reached already and playerAttack() is called as the next move.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	IllegalStateException is thrown.
Testing whether turn count increases by one when playerAttack() is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	The count for turn increases by 1.
Testing if an attack fails when the target is not in the same room as the player.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>,	IllegalArgumentException is thrown.

	“YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”)	
Testing if item is removed from the world and subsequently player’s item bag after attack fails i.e other players saw you attack.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.playerAttack(“Laser Gun”) player.toString()	String representation of the player does not contain the item anymore in the bag after the unsuccessful attack.

Testing movePet()	**Input**	**Expected Value**
Testing if IllegalArgumentException is thrown when roomName is null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePet(null)	IllegalArgumentException is thrown.
Testing if IllegalArgumentException is thrown when roomName is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePet(“”)	IllegalArgumentException is thrown.
Testing for IllegalArgumentException being thrown when the room name passed is such that no such room exists in the world.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePet(<invalid_room_name>)	IllegalArgumentException is thrown.
Testing if target moves after movePet().	WorldImpl(21, 22,	Target moves and current room index

	“Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePet(“Lab”)	is 1 greater than before in a cyclic manner.
Testing when MaxTurns has reached already and movePet() is called as the next move.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePet(“Lab”)	IllegalStateException is thrown.
Testing whether turn count increases by one when movePet() is called.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePet(“Lab”)	The count for turn increases by 1.

2. TargetImpl

Testing receiveAttack()	**Input**	**Expected Value**
Testing if the health of the Target object decreases when received an attack from one of the players.	PlayerImpl(4, “Jedi”, <List of items>) TargetImpl(0,”alien”, 100, <List of rooms>) hlt1=target.getHealth() jedi.attack(<damage_amount>) alien.receiveAttack(<damage_amount>) hlt2=target.getHealth()	hlt2 < hlt1
Testing if IllegalArgumentException is	PlayerImpl(4, “Jedi”, <List	IllegalArgumentException is thrown.

thrown if damage trying to be made is negative.	of items>) TargetImpl(0,”alien”, 100, <List of rooms>) player.attack(-1) target.receiveAttack(-1)	
Testing if IllegalArgumentException is thrown if damage trying to be made is equal to zero.	PlayerImpl(4, “Jedi”, <List of items>) TargetImpl(0,”alien”, 100, <List of rooms>) player.attack(0) target.receiveAttack(0)	IllegalArgumentException is thrown.

Testing toString()	**Input**	**Expected Value**
Testing if the string representation of the target represents the correct value of health after an attack is received.	PlayerImpl(4, “Jedi”, <List of items>) TargetImpl(0,”alien”, 100, <List of rooms>) jedi.attack() alien.receiveAttack(<damage_amount>) alien.toString()	Correct String representation reflecting the change in target’s health after an attack was received.

3. PlayerImpl

Testing lookAround()	**Input**	**Expected Value**
Testing whether lookAround() gives back the correct list of neighbour rooms.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.lookAround()	Correct String representation of list of neighbour rooms except that room that has the pet in it if any such room is a neighbour of the current room.
Testing whether turn count increases by one when lookAround() is called.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	The count for turn increases by 1.

	jedi.lookAround()	
Testing target moves after lookAround().	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.lookAround()	Target moves and current room index is 1 greater than before in a cyclic manner.
Testing lookAround() when pet is in one of the neighbours.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.lookAround()	Correct String representation of list of all neighbour rooms except that room that has the pet in it.
Testing lookAround() when pet is in none of the neighbours rooms.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.lookAround()	Correct String representation of list of all neighbour rooms.
Testing lookAround() when pet is in the same room.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.lookAround()	Correct String representation of list of all neighbour rooms

4. RoomImpl

Testing toString()	**Input**	**Expected Value**
Testing whether toString() returns the correct value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.toString()	Correct string representation of the room which includes room name, items in the room, neighbouring rooms, players in the room and also the pet if it is currently present in the room.
Testing toString() in whether it returns false for pet’s presence in the room when there is no pet in the room.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.toString()	Correct string representation of the room which includes room name, items in the room, neighbouring rooms, players in the room and also the pet’s presence as false.
Testing toString() in whether it returns true for pet’s presence when there is a pet present in the room.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.toString()	Correct string representation of the room which includes room name, items in the room, neighbouring rooms, players in the room and also the pet’s presence as true.
Testing toString() when pet was not in the room before but was moved to the current room, room’s toString() should be able to return the correct string value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) PetImpl(“godzilla”, currentRoom, <List of rooms>)	Correct string representation is returned that specifies the correct status of the pet even after moving the pet to this room.

	room.statusChangePetInRoom(false) room.toString() room.statusChangePetInRoom(true) room.toString()	
Testing toString() when pet was in the room before but was moved to another room, room's toString() should be able to return the correct string value.	RoomImpl(10,"EastLab", 0, 5, 6, 8, <List of items>) PetImpl("godzilla", currentRoom, <List of rooms>) room.statusChangePetInRoom(true) room.toString() room.statusChangePetInRoom(false) room.toString()	Correct string representation is returned that specifies the correct status of the pet even after moving the pet to another room.

5. PetImpl

Testing Construction()	Input	Expected Value
Testing whether IllegalArgumentException is thrown when name is null.	PetImpl(null, currentRoom, <List of rooms>)	IllegalArgumentException is thrown.
Testing whether IllegalArgumentException is thrown when name is an empty String.	PetImpl("",currentRoom, <List of rooms>)	IllegalArgumentException is thrown.
Testing whether IllegalArgumentException is	PetImpl("", null, <List of rooms>)	IllegalArgumentException is thrown.

thrown when currentRoom is null.		
Testing whether IllegalArgumentException is thrown when List of rooms passed is null.	PetImpl(“”,currentRoom, null)	IllegalArgumentException is thrown.
Testing whether IllegalArgumentException is thrown when the size of List of Rooms is 0.	PetImpl(“”,currentRoom, <List of 0 rooms>)	IllegalArgumentException is thrown.
Testing for successful construction when all parameters passed are valid.	PetImpl(“godzilla”, currentRoom, <List of rooms>)	Object created, hashCode() returns a value.

Testing getCurrentRoom()	Input	Expected Value
Testing whether getCurrentRoom returns a valid room when called.	PetImpl(“godzilla”, currentRoom, <List of rooms>)	Valid Room is returned which is the current room of the pet.
Testing whether room at index 0 of the global room list is returned when the game starts.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “alien”, “Yeti”, 100, 30) PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.getCurrentRoom()	Room at index 0 of the global room list is returned as the pet starts with the target.

Testing getName()	Input	Expected Value
Testing whether getName returns the name of the pet when called.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.getName()	Name of the pet is returned which should match the string value sent in the input specifications while running the project.

Testing movePet()	Input	Expected Value
Testing whether IllegalArgumentException is thrown when room is null.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.movePet(null)	IllegalArgumentException is thrown.
Testing whether IllegalArgumentException is thrown when the room passed is not present in the roomslist.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.movePet(“Galaxy”)	IllegalArgumentException is thrown.
Testing whether movePet actually moves the pet to the specified room.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.movePet(“Lab”)	Pet is moved to the specific room mentioned, this is reflected in toString() of the room as well as the pet's current room is updated.

Testing movePetDFS()	Input	Expected Value
Testing whether the pet moves to the next nearest unvisited neighbour room in the next call to this function.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.movePetDFS() pet.getCurrentRoom() pet.movePetDFS() pet.getCurrentRoom()	Pet moves to the next neighbour which is unvisited.
Testing whether the pet moves back to the parent room when all neighbours of the room are visited.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.movePetDFS() pet.getCurrentRoom()	Pet moves back to the parent room when all its child rooms are visited.
Testing whether the pet starts from room 0 in the roomsList when all the rooms are visited.	PetImpl(“godzilla”, currentRoom, <List of rooms>) pet.movePetDFS() pet.getCurrentRoom() pet.movePetDFS() pet.getCurrentRoom()	Pet’s current room returns the first room after all the rooms in the roomsList are visited.
Testing whether the pet starts from room 0 in the roomsList	PetImpl(“godzilla”, currentRoom, <List of	Pet’s get current room returns the 0th room in the roomsList when the first

when the first time it's called.	rooms>) pet.movePetDFS() pet.getCurrentRoom()	time its called.
Testing whether the pet starts DFS from the current room next after an operation of movePet() to the current Room was made.	PetImpl("godzilla", currentRoom, <List of rooms>) pet.movePet(room1) pet.movePetDFS() pet.getCurrentRoom()	The pet moves to the next room in the DFS order i.e. the next univisited neighbour of the room that the pet was moved to after the movePet function is called.

CONTROLLER Test

1. ControllerImpl

Testing start()	Input	Expected Value
Testing start if a valid mock world is passed and it successfully starts the game.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when mock world == null, IllegalArgumentException is thrown.	ControllerImpl(in, out) controller.start(null)	IllegalArgumentException is thrown.
Testing when addPlayerPiece command is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing when displayPlayerDescription is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing when playerLookAround is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when playerPickItem is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.

Testing start when playerMove is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when getRoomInformation is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when invalid command name is sent.	ControllerImpl(in, out) controller.start(mockWorldImpl)	The program asks for the input again.
Testing start when invalid inputs sent to command and the loop continues asking for next command.	ControllerImpl(in, out) controller.start(mockWorldImpl)	The program loops through asking for command inputs again.
Testing start when maxTurns has reached, game Quits.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Game quits gracefully.
Testing start to check if the players take turns in the order they were inserted into the game.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Players take turns in order they were inserted in the game.
Testing start to check if computer player added after a player actually plays automatically after player's turn.	ControllerImpl(in, out) controller.start(mockWorldImpl)	CPU player added after a player actually plays automatically after player's turn.
Testing start that calling methods that do not make a turn actually do not modify the turn being displayed.	ControllerImpl(in, out) controller.start(mockWorldImpl)	The turn displayed remains fixed when those command are used that actually do not constitute a turn.
Testing start when movePet is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid string specifying the status of the move operation.
Testing start when movePet is passed an invalid room name.	ControllerImpl(in, out) controller.start(mockWorldImpl)	The program loops through asking for command inputs again and the appendable string for the controller specifies that invalid room name was passed.
Testing start when attack is called in a valid way.	ControllerImpl(in, out) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid string specifying the status of the player attack operation.

Testing start when attack is passed an invalid item name.	ControllerImpl(in, out) controller.start(mockWorldImpl)	The program loops through asking for command inputs again and the appendable string for the controller specifies that invalid item name was passed.

2. MovePet

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	MovePet(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan == null.	MovePet(scan, out)	IllegalArgumentException is thrown.
Testing construction when scan != null.	MovePet(scan, out)	Object created, hashCode() returns a value.
Testing construction when out == null.	MovePet(scan, out)	IllegalArgumentException is thrown.
Testing construction when out != null.	MovePet(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null and all valid parameters are passed.	MovePet(scan, out) movePet.excecute(mockModel)	Expected String value of parameters passed and thus received by the mock model's movePet() function.
Testing execute when mock model == null.	MovePet(scan, out) movePet.excecute(null)	IllegalArgumentException is thrown.
Testing execute when roomName sent is an invalid room name.	MovePet(scan, out) movePet.excecute(mockModel)	IllegalArgumentException is thrown.

3. Player Attack

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	PlayerAttack(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan == null.	PlayerAttack(scan, out)	IllegalArgumentException is thrown.
Testing construction when scan != null.	PlayerAttack(scan, out)	Object created, hashCode() returns a value.
Testing construction when out == null.	PlayerAttack(scan, out)	IllegalArgumentException is thrown.
Testing construction when out != null.	PlayerAttack(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null and all valid parameters are passed.	PlayerAttack(scan, out) playerAttack.excecute(mock Model)	Expected String value of parameters passed and thus received by the mock model's attack() function.
Testing execute when mock model == null.	PlayerAttack(scan, out) playerAttack.excecute(null)	IllegalArgumentException is thrown.
Testing when invalid item name is sent in the prompt.	PlayerAttack(scan, out) playerAttack.excecute(mock Model)	Prints to the appendable out the string specifying invalid item name passed.

Testing plan for Milestone 1 and 2

MODEL TEST

1. WorldImpl

Testing Construction()	Input	Expected Value
Length of items list >= 0	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Testing number of rows > 0.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Testing number of columns > 0.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Testing number of columns <= 0.	WorldImpl(21, 0, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	IllegalArgumentException is thrown
Testing number of rows <= 0.	WorldImpl(-1, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	IllegalArgumentException is thrown
Length of rooms list >= 1	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 21 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Length of rooms list < 1	WorldImpl(21, 22, “Area51”, <List of 0 rooms>,	IllegalArgumentException is thrown.

	<List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	
Target name == null	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, null, “yeti”, 100, 30)	IllegalArgumentException is thrown.
Target name != null	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Target health >=1	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Target health < 1	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, -100, 30)	IllegalArgumentException is thrown
Testing non-overlapping rooms	WorldImpl(21, 22, “Area51”, <List of 24 VALID rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Testing overlapping rooms.	WorldImpl(21, 22, “Area51”, <List of 24 INVALID rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	IllegalArgumentException is thrown
Testing if items in items list are in invalid room indices.	WorldImpl(21, 22, “Area51”, <List of 24 VALID rooms>, <List of 25 invalid items>, “YetiAlien”, “yeti”,	IllegalArgumentException is thrown

	100, 30)	
Testing if items in items list are in valid room indices.	WorldImpl(21, 22, “Area51”, <List of 24 VALID rooms>, <List of 25 invalid items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.
Is room size <= world size	WorldImpl(21, 22, “Area51”, <List of 24 inavlid rooms>, <List of 25 valid items>, “YetiAlien”, “yeti”, 100, 30)	IllegalArgumentException is thrown
If maxTurns < 1	WorldImpl(21, 22, “Area51”, <List of 24 inavlid rooms>, <List of 25 valid items>, “YetiAlien”, “yeti”, 100, -1)	IllegalArgumentException is thrown
If maxTurns >= 1	WorldImpl(21, 22, “Area51”, <List of 24 inavlid rooms>, <List of 25 valid items>, “YetiAlien”, “yeti”, 100, 30)	Object created, hashCode() returns a value.

Testing toString()	**Input**	**Expected Value**
Testing toString() to return correct String representation of the WorldImpl object we create.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) Area51.toString()	21 22 Area51

Testing getRows()	**Input**	**Expected Value**
Testing whether getRows() returns the correct number of rows.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	21

Testing getRoomsList()	**Input**	**Expected Value**
Testing whether getRoomsList() returns the correct list of rooms.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	List of rooms.

Testing getCols()	**Input**	**Expected Value**
Testing whether getCols() returns the correct number of columns.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	22

Testing getWorldImplName()	**Input**	**Expected Value**
Testing whether getWorldImplName() returns the correct name of the world.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	“Area51”

Testing addPlayerPiece()	**Input**	**Expected Value**
Testing whether addPlayerPiece() adds the PlayerPiece object to the end of playerList.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	The new playerPiece is added to the playerList.

	world.addPlayerPiece("sid", "lab", 10, true)	
Testing when adding an already existing playerName will throw IllegalArgumentException.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.addPlayerPiece(<already_exisiting_player_name>, "lab", 10, true)	IllegalArgumentException is thrown.
Testing when playerName == targetName.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.addPlayerPiece(<target_name>, "lab", 10, true)	IllegalArgumentException is thrown.
Testing when playerName is null, IllegalArgumentException is thrown.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.addPlayerPiece(null, "lab", 10, true)	IllegalArgumentException is thrown.
Testing when playerName is an empty string.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.addPlayerPiece("", "lab", 10, true)	IllegalArgumentException is thrown.
Testing when room name does not exist in the roomsList will throw IllegalArgumentException.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) world.addPlayerPiece("sid", <non-existing	IllegalArgumentException is thrown.

	roomName>, 10, true)	
Testing when the room name is valid.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, “lab”, 10, true)	The new playerPiece is added to the playerList.
Testing when the room name is null. IllegalArgumentException is thrown.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, null, 10, true)	IllegalArgumentException is thrown.
Testing when the room name is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, “”l, 10, true)	IllegalArgumentException is thrown.

Testing playComputerPlayer()	**Input**	**Expected Value**
Testing whether the computerPlayer moves as intended when the ‘pseudo random’ value i.e. option is set.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, “”l, 10, true) world.addPlayerPiece(“cpu”, “”l, 10, false) world.playComputerPlayer(1, false)	The computer player performs playerLookAround() and send s the appropriate String value.
Testing whether playComputerPlayer throws IllegalArgumentException when option < 0.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>,	IllegalArgumentException is thrown.

	“YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, “”l, 10, true) world.addPlayerPiece(“cpu”, “”l, 10, false) world.playComputerPlayer(-1, false)	
Testing whether playComputerPlayer throws IllegalArgumentException when option > 2.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, “”l, 10, true) world.addPlayerPiece(“cpu”, “”l, 10, false) world.playComputerPlayer(3, false)	IllegalArgumentException is thrown.
Testing whether playComputerPlayer works as expected when option >=0 and option <=2.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.addPlayerPiece(“sid”, “”l, 10, true) world.addPlayerPiece(“cpu”, “”l, 10, false) world.playComputerPlayer(1, false)	The computer player performs playerLookAround() and sends the appropriate String value.
Testing when playComputerPlayer() throws IllegalStateException when numberOfTurns > maxTurns.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.addPlayerPiece(“sid”, “”l, 10, true) world.playerLookAround() world.playComputerPlayer(	IllegalStateException is thrown.

	1, false)	

Testing movePlayerPiece()	**Input**	**Expected Value**
Testing whether movePlayerPiece() moves the player to the correct room.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(“lab”)	The room “lab” should contain “sid” player.
Testing invalid roomName is entered. IllegalArgumentException is thrown.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(<invalid_room_name>)	IllegalArgumentException is thrown.
Testing when roomName is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(“”)	IllegalArgumentException is thrown.
Testing when roomName is null. IllegalArgumentException is thrown.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(null)	IllegalArgumentException is thrown.
Testing when moving a non-existing player piece, IllegalArgumentException is thrown.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(“lab”)	IllegalArgumentException is thrown.
Testing when a room passed is not a neighbour of the current room.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”,	IllegalArgumentException is thrown.

	100, 30) world.movePlayerPiece(“LabEast”)	
Testing target moves after movePlayerPiece().	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(“lab”)	Target moves and current room index is 1 greater than before in a cyclic manner.
Testing when MaxTurns is reached already and movePlayerPiece is called as the next move.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.movePlayerPiece(“lab”)	IllegalStateException is thrown.
Testing when the player moves to the correct room, the room has the player.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(“lab”)	The new room contains the player.
Testing when the player moves to the correct room, the previous room removes the player.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.movePlayerPiece(“lab”)	The previous room does not contain the player.

Testing playerPickItem()	**Input**	**Expected Value**
Testing whether playerPickItem() adds the item to the players itemList() when the correct item name is sent.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	Adds the item to the player's item list. Returns the string that the item was added successfully to the player’s bag.

	Area51.playerPickItem("sid", "gun")	
Testing whether playerPickItem() throws IllegalArgumentException when the item name is wrong.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) Area51.playerPickItem("sid", <wrong_item_name>)	IllegalArgumentException is thrown
Testing when the item name is an empty string.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) Area51.playerPickItem("sid", "")	IllegalArgumentException is thrown
Testing when itemName is null. IllegalArgumentException is thrown.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) Area51.playerPickItem("sid", null)	IllegalArgumentException is thrown.
Testing when the player's itemBagLimit still has space to accommodate more items.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 30) Area51.playerPickItem("sid", "gun")	Item added to players itemList. Returns the string that the item was added successfully to the player's bag.
Testing when MaxTurns is reached already and player pick item is called as the next move.	WorldImpl(21, 22, "Area51", <List of 24 rooms>, <List of 25 items>, "YetiAlien", "yeti", 100, 1) Area51.playerPickItem("gun")	IllegalStateException is thrown.

Testing when the player's itemBagLimit does not have space to accommodate more items.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) Area51.playerPickItem(“sid”, “gun”)	Returns string that picking up item was not successful.
Testing when the item does not exist in the room.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) Area51.playerPickItem(“sid”, <item_not_in_room>)	IllegalArgumentException is thrown.
Testing target moves after playerPickItem().	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) area51.playerPickItem(“sid”, <item_in_room>)	Target moves and current room index is 1 greater than before in a cyclic manner.

Testing playerLookAround()	**Input**	**Expected Value**
Testing whether PlayerLookAround() gives a correct string of neighbour rooms that can be seen from the current room the player is currently in.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.lookAround()	String consisting of neighbour rooms, information about its current space, items in it and players in it.
Testing target moves after playerLookAround().	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.lookAround(player)	Target moves and current room index is 1 greater than before in a cyclic manner.

Testing when MaxTurns is reached already and playLookAround is called as the next move.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) world.lookAround(player)	IllegalStateException is thrown.

Testing getTurnInformation()	**Input**	**Expected Value**
Testing whether it throws IllegalStateException when numberOfTurns >= maxTurns of the game.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) addPlayerPiece(Name, roomName, itemBagLimit, true) world.lookAround() world.lookAround() getTurnInformation()	IllegalStateException is thrown.
Testing if correct String value is returned for a human player turn after a human player turn.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1) addPlayerPiece(Name, roomName, itemBagLimit, true) addPlayerPiece(Name, roomName, itemBagLimit, true) getTurnInformation()	Expected value is returned for a human player turn after a human player turn.
Testing whether correct information about the target’s current location and the information about the current room and its items is returned.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 1)	Expected value is returned with information about the target’s current location and the information about the current room and its items.

	addPlayerPiece(Name, roomName, itemBagLimit, true) addPlayerPiece(Name, roomName, itemBagLimit, true) getTurnInformation()	

Testing getPlayerDescription()	**Input**	**Expected Value**
Testing whether getPlayerDescription() returns correct description of player when correct name parameter is sent.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.getPlayerDescription(“sid”)	Correct player description.
Testing whether getPlayerDescription() throws IllegalArgumentException when incorrect player name is sent.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.getPlayerDescription(<non_existing_player_name>)	IllegalArgumentException is thrown.
Testing when playerName is null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.getPlayerDescription(null)	IllegalArgumentException is thrown.
Testing when playerName is an empty string.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30)	IllegalArgumentException is thrown.

	world.getPlayerDescription(“”)	

Testing isCurrentTurnHuman()	**Input**	**Expected Value**
Testing whether isCurrentTurnHuman() returns the correct value of true when the current turn is of a human.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.isCurrentTurnHuman()	Gives back true if the current turn is human.

Testing displayRoomInformation()	**Input**	**Expected Value**
Testing displayRoomInformation when room name == null.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(null)	IllegalArgumentException is thrown.
Testing displayRoomInformation when room name is an empty String.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(“”)	IllegalArgumentException is thrown.
Testing displayRoomInformation when room name is valid.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(<valid_room_name>)	Returns proper and expected String representation for the room as same as toString() of the room.

Testing displayRoomInformation when the room with that name is not present in the world.	WorldImpl(21, 22, “Area51”, <List of 24 rooms>, <List of 25 items>, “YetiAlien”, “yeti”, 100, 30) world.displayRoomInformation(<invalid_room_name>)	IllegalArgumentException is thrown.

Testing playComputerPlayer() indirectly	**Input**	**Expected Value**
Testing target moves after playComputerPlayer().	playComputerPlayer(2, false)	Target moves and current room index is 1 greater than before in a cyclic manner.

Testing attack()	**Input**	**Expected Value**
Testing if the player attacks and the health of the target decreases.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) TargetImpl(0,”alien”, 100, <List of rooms>) hlt1=target.getHealth() jedi.attack() alien.receiveAttack() hlt2=target.getHealth()	hlt2<hlt1

2. RoomImpl

Testing Construction()	Input	Expected Value
Number < 0	RoomImpl(-1,”EastLab”, 1, 1, 10, 10, <List of items>)	IllegalArgumentException is thrown
Number >=0	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Name==null	RoomImpl(10, null, 1, 1, 10, 10, <List of items>)	IllegalArgumentException is thrown.
Name is an empty string.	RoomImpl(10, “”l, 1, 1, 10, 10, <List of items>)	IllegalArgumentException is thrown.
Name!=null	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Left Top Row<0	RoomImpl(10,”EastLab”, 1, 1, -1, 10, <List of items>)	IllegalArgumentException is thrown
Left Top Column<0	RoomImpl(10,”EastLab”, 1, 1, 10, -4, <List of items>)	IllegalArgumentException is thrown
Left Top Row>=0	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Left Top Column>=0	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Right Bottom Row<0	RoomImpl(10,”EastLab”, 1, 1, -10, 10, <List of items>)	IllegalArgumentException is thrown
Right Bottom Column<0	RoomImpl(10,”EastLab”, 1, 1, 10, -10, <List of items>)	IllegalArgumentException is thrown
Right Bottom Row>=1	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Right Bottom Column>=1	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Right Bottom Row <= Left Top Row	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of	IllegalArgumentException is thrown

	items>)	
Right Bottom Row>Left Top Row	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Right Bottom Column<=Left Top Column	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	IllegalArgumentException is thrown
Right Bottom Column>Left Top Column	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Length of item list>=0	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.
Item list == null	RoomImpl(10,”EastLab”, 1, 1, 10, 10, null)	IllegalArgumentException is thrown
All valid parameters passed	RoomImpl(10,”EastLab”, 1, 1, 10, 10, <List of items>)	Object created, hashCode() returns a value.

Testing calculateNeighborsAndSave()	**Input**	**Expected Value**
roomsList.size() < 1	calculcateNeighborsAndSave(<List of 0 rooms>)	IllegalArgumentException is thrown.
roomsList == null	calculcateNeighborsAndSave(<List of 0 rooms>)	IllegalArgumentException is thrown.
All valid parameters passed.	calculcateNeighborsAndSave(<List of 20 rooms>)	Object created, hashCode() returns a value.

Testing toString()	**Input**	**Expected Value**
Testing whether toString() returns the correct value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.toString()	Correct string representation of the room.

Testing getNeighbors()	**Input**	**Expected Value**
Testing if 2 rooms are neighbours i.e. they	RoomImpl(10,”EastLab”,	Room1’s neighbour is room2.

share a wall.	0, 5, 6, 8, <List of items> RoomImpl(11,”AlienRoom”, 0, 8, 6, 11, <List of items>)	
Testing if 2 rooms are not neighbours i.e. they don’t share a wall.	RoomImpl(10,”EastLab”, 0, 0, 4, 3, <List of items> RoomImpl(11,”AlienRoom”, 29, 17, 40, 21, <List of items>)	0 size neighborsList for room1.

Testing getItemList()	**Input**	**Expected Value**
Testing if getItemList returns a non-null list of Item.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>)	Returns List<Item> which is not null.

Testing equals()	**Input**	**Expected Value**
Testing equals() to test if 2 rooms are equal based on their room name.	RoomImpl(10,”EastLab”, 1, 2, 10, 20, <List of items> RoomImpl(10,”EastLab”, 1, 2, 10, 20, <List of items>)	true
Testing equals() to test if 2 rooms are not equal based on their room name.	RoomImpl(10,”EastLab”, 1, 2, 10, 20, <List of items> RoomImpl(12,”WestLab”, 1, 2, 10, 20, <List of items>)	false

Testing getLeftTopRow()	**Input**	**Expected Value**
Testing whether getLeftTopCol() returns correct leftTopRow value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>)	Correct leftTopRow value.

	room.getLeftTopRow()	

Testing getLeftTopCol()	**Input**	**Expected Value**
Testing whether getLeftTopCol() returns correct leftTopCol value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.getLeftTopCol()	Correct leftTopCol value.

Testing getRoom()	**Input**	**Expected Value**
Testing if getRoom() returns the room number.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.getRoom()	Return room number.

Testing getRightBottomRow()	**Input**	**Expected Value**
Testing whether getRightBottomRow() returns correct rightBottomRow value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.getrightBottomRow( )	Correct rightBottomRow value.

Testing getRightBottomCol()	**Input**	**Expected Value**
Testing whether getRightBottomCol() returns the correct rightBottomCol value.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.getRightBottomCol( )	Correct rightBottomCol value.

Testing getName()	**Input**	**Expected Value**
Testing whether getName() returns the correct name.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.getName()	Correct room name value returned.

Testing addPlayerToRoom()	**Input**	**Expected Value**
Testing whether adding null player throws IllegalArgumentException.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.addPlayerToRoom(null)	IllegalArgumentException is thrown.
Testing whether a valid player gets added to playersInSpaceList.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.addPlayerToRoom(<valid_player_piece>)	Valid player gets added to the playersInspaceList.

Testing removePlayerFromRoom()	**Input**	**Expected Value**
Testing whether removing non-existing player throws IllegalArgumentException.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.removePlayerFrom Room(<non_existing_play er>)	IllegalArgumentException is thrown.
Testing whether removing from empty list throws IllegalStateException	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.removePlayerFrom Room(<valid_player_piec e>)	IllegalStateException is thrown.
Testing whether a valid player gets removed from the playersInSpace List.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.removePlayerFrom Room(<valid_player_piec e>)	Valid player gets removed from the playersInspaceList.
Testing when playerPiece is null.	RoomImpl(10,”EastLab”, 0, 5, 6, 8, <List of items>) room.removePlayerFrom Room(null)	IllegalArgumentException is thrown.

Testing removeItemFromRoom()	**Input**	**Expected Value**

Testing whether removing an invalid i.e. non existing item throws Illegal ArgumentException.	RoomImpl(10,"EastLab", 0, 5, 6, 8, <List of items>) room.removeItemsFromRoom(<invalid_name>)	IllegalArgumentException is thrown.
Testing when null is sent as item. IllegalArgumentException is thrown.	RoomImpl(10,"EastLab", 0, 5, 6, 8, <List of items>) room.removeItemsFromRoom(null)	IllegalArgumentException is thrown.
Testing whether removing from empty item list throws IllegalStateException	RoomImpl(10,"EastLab", 0, 5, 6, 8, <List of items>) room.removeItemsFromRoom(<valid_item_name>)	IllegalStateException is thrown.
Testing whether a valid item gets removed from the itemsList.	RoomImpl(10,"EastLab", 0, 5, 6, 8, <List of items>) room.removeItemsFromRoom(<valid_Item_name>)	Valid Item gets removed from the itemsList.

Testing getPlayersInRoom()	**Input**	**Expected Value**
Testing whether it returns the copy of players in the room list.	RoomImpl(10,"EastLab", 0, 5, 6, 8, <List of items>) room.getPlayersInRoom()	Copy of the players in the room list is returned.

3. ItemImpl

Testing Construction()	Input	Expected Value
Id >= 0	ItemImpl(1, "laser", 4, 12)	Object created, hashCode() returns a value.
Id < 0	ItemImpl(-1, "laser", 4, 12)	IllegalArgumentException is thrown
name==null	ItemImpl(1, null, 4, 12)	IllegalArgumentException is thrown
name!=null	ItemImpl(1, "laser", 4, 12)	Object created, hashCode() returns a value.

Name is an empty string.	ItemImpl(1, “laser”, 4, 12)	IllegalArgumentException is thrown
damage>0	ItemImpl(1, “laser”, 4, 12)	Object created, hashCode() returns a value.
damage<=0	ItemImpl(1, “laser”, 4, -8)	IllegalArgumentException is thrown
containingRoomIndex>=0	ItemImpl(1, “laser”, 4, 12)	Object created, hashCode() returns a value.
containingRoomIndex<0	ItemImpl(1, “laser”, 4, 12)	IllegalArgumentException is thrown

Testing getContainingRoomIndex()	**Input**	**Expected Value**
Testing if getContainingRoomIndex() returns a non negative integer.	ItemImpl(1, “laser”, 4, 12)	12

Testing getName()	**Input**	**Expected Value**
Testing if getName() returns a non null String value.	ItemImpl(1, “laser”, 4, 12)	laser

Testing getDamage()	**Input**	**Expected Value**
Testing if getDamage() returns a positive integer.	ItemImpl(1, “laser”, 4, 12)	4

Testing toString()	**Input**	**Expected Value**
Testing toString() to return correct String representation of the Item object we create.	ItemImpl(1, “laser”, 4, 12) laser.toString()	12 4 laser

4. PlayerImpl

Testing Construction()	**Input**	**Expected Value**
number>=0	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Object created, hashCode() returns a value.
number<0	PlayerImpl(-1, “Jedi”,	IllegalArgumentException is thrown

	<Valid_Room>, 10, true)	
name==null	PlayerImpl(4, null, Valid_Room>, 10, true)	IllegalArgumentException is thrown
Name is an empty string.	PlayerImpl(4, “”, <Valid_Room>, 10, true)	IllegalArgumentException is thrown
name!=null	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Object created, hashCode() returns a value.
room == null	PlayerImpl(4, “Jedi”, null, 10, true)	IllegalArgumentException is thrown
room != null	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Object created, hashCode() returns a value.
itemBagLimit < 0	PlayerImpl(4, “Jedi”, <Valid_Room>, -1, true)	IllegalArgumentException is thrown
itemBagLimit >= 0	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Object created, hashCode() returns a value.

Testing getCurrentRoom()	**Input**	**Expected Value**
Testing if getCurrentRoom() returns a non null Room.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Copy of current room object.

Testing getItemList()	**Input**	**Expected Value**
Testing if getItemList() returns a list of items.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Copy of <List of items>

Testing move()	**Input**	**Expected Value**
Testing when the room is null.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.move(null)	IllegalArgumentException is thrown.
Testing when the room is not a neighbour of the current room.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.move(<invalid_room>)	IllegalArgumentException is thrown.
Testing move when valid parameters are passed.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.move(<valid_room>)	Player moves to the <valid_room>. player.getCurrentRoom().getName() will give the name of the

		<valid_room>.

Testing pickItem()	**Input**	**Expected Value**
Testing if the player picks the item from the item list of the room in which the player is present. The Item is valid.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.pickItem(“gun”)	Player picks the item from the item list of the room in which the player is present. true is returned.
Testing if IllegalArgumentException is thrown for invalid item i.e. item non-existing in the room.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.pickItem(<invalid_item>)	IllegalArgumentException is thrown.
Testing when the item is null, IllegalArgumentException is thrown.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.pickItem(null)	IllegalArgumentException is thrown.
Testing when item bag limit is reached and player trying to add item.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.pickItem(“gun”)	false
Testing when the item bag limit has not reached and there is space to accommodate the item.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.pickItem(“gun”)	Player picks the item from the item list of the room in which the player is present. true is returned.

Testing getItemBagLimit()	**Input**	**Expected Value**
Testing if getItemBagLimitt() returns the item bag limit for the player.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.getItemBagLimit()	Correct bag limit value.

Testing lookAround()	**Input**	**Expected Value**
Testing whether lookAround() gives back the correct list of neighbour rooms.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	Correct String representation of list of neighbour rooms.

	jedi.lookAround()	

Testing getName()	**Input**	**Expected Value**
Testing if getName() returns a valid and expected string.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	“jedi”

Testing getNumber()	**Input**	**Expected Value**
Testing if getNumber() returns a non negative integer.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true)	4

Testing getIsHuman()	**Input**	**Expected Value**
Testing whether getIsHuman() gives back whether the current player is human or computer player.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.getIsHuman()	Give back true.

Testing removeItemFromBag()	**Input**	**Expected Value**
Testing whether the function removes the item successfully when there is at least 1 item.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.removeItemFromBag(<valid_item>)	Remove the item from the bag. Can be seen in itemList. true is returned.
Testing when there is no such item in the bag.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.removeItemFromBag(<invalid_item>)	IllegalArgumentException is thrown.
Testing when the item is null	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.removeItemFromBag(null)	IllegalArgumentException is thrown.
Testing when the item list of the player is empty.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.removeItemFromBag(<val	false is returned.

	id_item>)	

Testing toString()	Input	Expected Value
Testing toString() to return correct String representation of the Player object we create.	PlayerImpl(4, “Jedi”, <Valid_Room>, 10, true) jedi.toString()	Player name: Jedi Player current room: lab Player items: <List of items>

5. TargetImpl

Testing Construction()	Input	Expected Value
number>=0	TargetImpl(0,”alien”, 100, <List of rooms>)	Object created, hashCode() returns a value.
number<0	TargetImpl(-1,”alien”, 100, <List of rooms>)	IllegalArgumentException is thrown
name==null	TargetImpl(0,null, 100, <List of rooms>)	IllegalArgumentException is thrown
name!=null	TargetImpl(0,”alien”, 100, <List of rooms>)	Object created, hashCode() returns a value.
Name is an empty string.	TargetImpl(0,””, 100, <List of rooms>)	IllegalArgumentException is thrown
roomList !=null	TargetImpl(0,”alien”, 100, <List of rooms>)	Object created, hashCode() returns a value.
health<0	TargetImpl(0,”alien”, -10, <List of rooms>)	IllegalArgumentException is thrown
health>=0	TargetImpl(0,”alien”, 100, <List of rooms>)	Object created, hashCode() returns a value.
Target Starts at Room at 0th index in the roomsList. We equate it to the index of Room 0.	TargetImpl(0,”alien”, 100, <List of rooms>)	They are equal.

Testing getName()	**Input**	**Expected Value**
Testing if getName() returns a valid and expected string.	TargetImpl(0,”alien”, 100, <List of rooms>)	“alien”

Testing getNumber()	**Input**	**Expected Value**
Testing if getNumber() returns a non negative integer.	TargetImpl(0,”alien”, 100, <List of rooms>)	0

Testing getHealth()	**Input**	**Expected Value**
Testing if getHealth() returns a non negative integer.	TargetImpl(0,”alien”, 100, <List of rooms>)	Current health value >=0.

Testing getCurrentRoom()	**Input**	**Expected Value**
Testing if getCurrentRoom() returns a non null Room.	TargetImpl(0,”alien”, 100, <List of rooms>)	Current room object.

Testing move()	**Input**	**Expected Value**
Testing if the target moves to the next room based on the indices from roomList.	TargetImpl(0,”alien”, 100, <List of rooms>) alien.move()	Target moves to the next room based on the indices from roomList.

Testing toString()	**Input**	**Expected Value**
Testing toString() to return correct String representation of the Target object we create.	TargetImpl(0,”alien”, 100, <List of rooms>) alien.toString()	100 alien

Testing receiveAttack()	**Input**	**Expected Value**
Testing if the health of the Target object decreases when received an attack from one of the players.	PlayerImpl(4, “Jedi”, <List of items>) TargetImpl(0,”alien”, 100,	hlt2<hlt1

	<List of rooms> hlt1=target.getHealth() jedi.attack() alien.receiveAttack() hlt2=target.getHealth()	

6. GenerateRandom

Testing Construction()	Input	Expected Value
Testing whether using the parameterized constructor will initialise the random variable with the passed value.	GenerateRandom(2) generateRandom.getRandomInt(3)	Object created, hashCode() returns a value. getRandomInt() will send out the correct value passed before int the constructor.

Testing getRandomInt()	Input	Expected Value
Testing whether getRandomInt() returns the expected value or not.	GenerateRandom(2) generateRandom.getRandomInt(3)	getRandomInt() will send out the correct value passed before int the constructor.

CONTROLLER TEST

Note:
We are going to make use of a mock model here to test the controller in an isolated way. The mock model implements the GameWorldImpl interface, same as what our model (WorldImpl) does.

1. ControllerImpl

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	ControllerImpl(in, out)	Object created, hashCode() returns a value.
Testing when in == null, IllegalArgumentException is thrown.	ControllerImpl(null, out)	IllegalArgumentException is thrown.
Testing when in != null, object gets created.	ControllerImpl(in, out)	Object created, hashCode() returns a value.
Testing when out == null, IllegalArgumentException is thrown.	ControllerImpl(in,null)	IllegalArgumentException is thrown.
Testing when out != null, object gets created.	ControllerImpl(in, out)	Object created, hashCode() returns a value.

Testing start()	Input	Expected Value
Testing start if a valid mock world is passed and it successfully starts the game.	ControllerImpl(in, out, commandMap, 30) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when mock world == null, IllegalArgumentException is thrown.	ControllerImpl(in, out, commandMap,30) controller.start(null)	IllegalArgumentException is thrown.
Testing when addPlayerPiece command is called in a valid way.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing when displayPlayerDescription is called in a valid way.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing when playerLookAround is	ControllerImpl(in, out,	Prints successfully to the appendable

called in a valid way.	commandMap,30) controller.start(mockWorldImpl)	out the valid strings.
Testing start when playerPickItem is called in a valid way.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when playerMove is called in a valid way.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when getRoomInformation is called in a valid way.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Prints successfully to the appendable out the valid strings.
Testing start when invalid command name is sent.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	The program asks for the input again.
Testing start when invalid inputs sent to command and the loop continues asking for next command.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	The program loops through asking for command inputs again.
Testing start when maxTurns has reached, game Quits.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Game quits gracefully.
Testing start to check if the players take turns in the order they were inserted into the game.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	Players take turns in order they were inserted in the game.
Testing start to check if computer player added after a player actually plays automatically after player’s turn.	ControllerImpl(in, out, commandMap,30) controller.start(mockWorldImpl)	CPU player added after a player actually plays automatically after player’s turn.
Testing start that calling methods that do not make a turn actually	ControllerImpl(in, out, commandMap,30)	The turn displayed remains fixed when those command are used that actually

do not modify the turn being displayed.	controller.start(mockWorldImpl)	do not constitute a turn.

2. DisplayWorldImplInformation

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	DisplayWorldImplInformation(out)	Object created, hashCode() returns a value.
Testing for IllegalArgumentException when Appendable out == null.	DisplayWorldImplInformation(in, null)	IllegalArgumentException is thrown.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	DisplayWorldImplInformation() displayInformation.excecute(mockModel)	Expected string data from mockModel's toString() method to out. Also displays the correct unique code that was passed when constructing the mock model object.
Testing execute when mock model == null.	DisplayWorldImplInformation() displayInformation.excecute(null)	IllegalArgumentException is thrown.

3. AddPlayerPiece

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	AddPlayerPiece(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan == null.	AddPlayerPiece(scan, out)	IllegalArgumentException is thrown.
Testing construction when scan != null.	AddPlayerPiece(scan, out)	Object created, hashCode() returns a value.
Testing construction when out ==	AddPlayerPiece(scan, out)	IllegalArgumentException is thrown.

null.		
Testing construction when out != null.	AddPlayerPiece(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	AddPlayerPiece(scan, out) addPlayerPiece.excecute(mockModel)	Expected String value of parameters passed and thus received by the mock model's addPlayerPiece() function.
Testing execute when mock model == null.	AddPlayerPiece(scan, out) addPlayerPiece.excecute(null)	IllegalArgumentException is thrown.
Testing execute when player name in the inputstream is invalid in the sense it is null or empty string.	AddPlayerPiece(scan, out) addPlayerPiece.excecute(null)	IllegalArgumentException is thrown.
Testing execute when room name in the inputstream is invalid in the sense it is null or empty string.	AddPlayerPiece(scan, out) addPlayerPiece.excecute(null)	IllegalArgumentException is thrown.
Testing execute when item bag limit in input stream is negative i.e. its an invalid value.	AddPlayerPiece(scan, out) addPlayerPiece.excecute(null)	IllegalArgumentException is thrown.

4. MovePlayerPiece

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	MovePlayerPiece(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan == null.	MovePlayerPiece(scan, out)	IllegalArgumentException is thrown.
Testing construction when scan != null.	MovePlayerPiece(scan, out)	Object created, hashCode() returns a value.
Testing construction when out ==	MovePlayerPiece(scan, out)	IllegalArgumentException is thrown.

null.		
Testing construction when out != null.	MovePlayerPiece(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	MovePlayerPiece(scan, out) movePlayerPiece.excecute(mockModel)	Expected String value of parameters passed and thus received by the mock model's addPlayerPiece() function.
Testing execute when mock model == null.	MovePlayerPiece(scan, out) movePlayerPiece.excecute(mockModel)	IllegalArgumentException is thrown.
Testing execute when room name in the inputstream is invalid in the sense it is null or empty string.	MovePlayerPiece(scan, out) movePlayerPiece.excecute(mockModel)	IllegalArgumentException is thrown.

5. PlayerPickItem

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	PlayerPickItem(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan == null.	PlayerPickItem(scan, out)	IllegalArgumentException is thrown.
Testing construction when scan != null.	PlayerPickItem(scan, out)	Object created, hashCode() returns a value.
Testing construction when out == null.	PlayerPickItem(scan, out)	IllegalArgumentException is thrown.
Testing construction when out != null.	PlayerPickItem(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	PlayerPickItem(scan, out) playerPickItem.excecute(mockModel)	Expected String value of parameters passed and thus received by the mock model's addPlayerPiece() function.

Testing execute when mock model == null.	PlayerPickItem(scan, out) playerPickItem.excecute(mockModel)	IllegalArgumentException is thrown.
Testing execute when item name in the inputstream is invalid in the sense it is null or empty string.	PlayerPickItem(scan, out) playerPickItem.excecute(mockModel)	IllegalArgumentException is thrown.

6. PlayerLookAround

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	PlayerLookAround(out)	Object created, hashCode() returns a value.
Testing for IllegalArgumentException when Appendable out == null.	PlayerLookAround(out)	IllegalArgumentException is thrown.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	PlayerLookAround(out) playerLookAround.execute(mockModel)	Expected string data from mockModel's toString() method to out. Also displays the correct unique code that was passed when constructing the mock model object.
Testing execute when mock model == null.	PlayerLookAround(out) playerLookAround.execute(null)	IllegalArgumentException is thrown.

7. DisplayPlayerDescription

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	DisplayPlayerDescription(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan	DisplayPlayerDescription(sc	IllegalArgumentException is thrown.

== null.	an, out)	
Testing construction when scan != null.	DisplayPlayerDescription(scan, out)	Object created, hashCode() returns a value.
Testing construction when out == null.	DisplayPlayerDescription(scan, out)	IllegalArgumentException is thrown.
Testing construction when out != null.	DisplayPlayerDescription(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	DisplayPlayerDescription(scan, out) displayPlayerDescription.execute(mockModel)	Expected String value of parameters passed and thus received by the mock model's addPlayerPiece() function.
Testing execute when mock model == null.	DisplayPlayerDescription(scan, out) displayPlayerDescription.execute(mockModel)	IllegalArgumentException is thrown.
Testing execute when player name in the inputstream is invalid in the sense it is null or empty string.	DisplayPlayerDescription(scan, out) displayPlayerDescription.execute(mockModel)	IllegalArgumentException is thrown.

8. DisplayRoomInformation

Testing Construction()	Input	Expected Value
Testing if an object gets created when all valid inputs are passed.	DisplayRoomInformation(scan, out)	Object created, hashCode() returns a value.
Testing construction when scan == null.	DisplayRoomInformation(scan, out)	IllegalArgumentException is thrown.
Testing construction when scan != null.	DisplayRoomInformation(scan, out)	Object created, hashCode() returns a value.
Testing construction when out == null.	DisplayRoomInformation(scan, out)	IllegalArgumentException is thrown.

Testing construction when out != null.	DisplayRoomInformation(scan, out)	Object created, hashCode() returns a value.

Testing execute()	Input	Expected Value
Testing execute when mock model != null.	DisplayRoomInformation(scan, out) displayRoomInformation.execute(mockModel)	Expected String value of parameters passed and thus received by the mock model's addPlayerPiece() function.
Testing execute when mock model == null.	DisplayRoomInformation(scan, out) displayRoomInformation.execute(mockModel)	IllegalArgumentException is thrown.
Testing execute when room name in the inputstream is invalid in the sense it is null or empty string.	DisplayRoomInformation(scan, out) displayRoomInformation.execute(mockModel)	IllegalArgumentException is thrown.

Which model we chose and why:

We decided to go with Siddharth's model because of the following reasons:

→ The selected model required least refactoring.

→ The selected model has been tested thoroughly proving that not only that our code CAN work but that it WILL ALWAYS work.

→ The selected model is completely housing the game logic in the model and not in controller.

→ The selected model has followed all the best practices of Style and Manual grading to perfection.

→ The selected model has a fully updated and easy to understand UML.

→ The selected model has the most clearly defined facade and can most easily be separated from the implementation of the controller in the previous milestone.

→ The selected model has high performance in completeness and correctness parameters.

→ The design decisions of the selected model are fully intuitive, making for a better user experience.