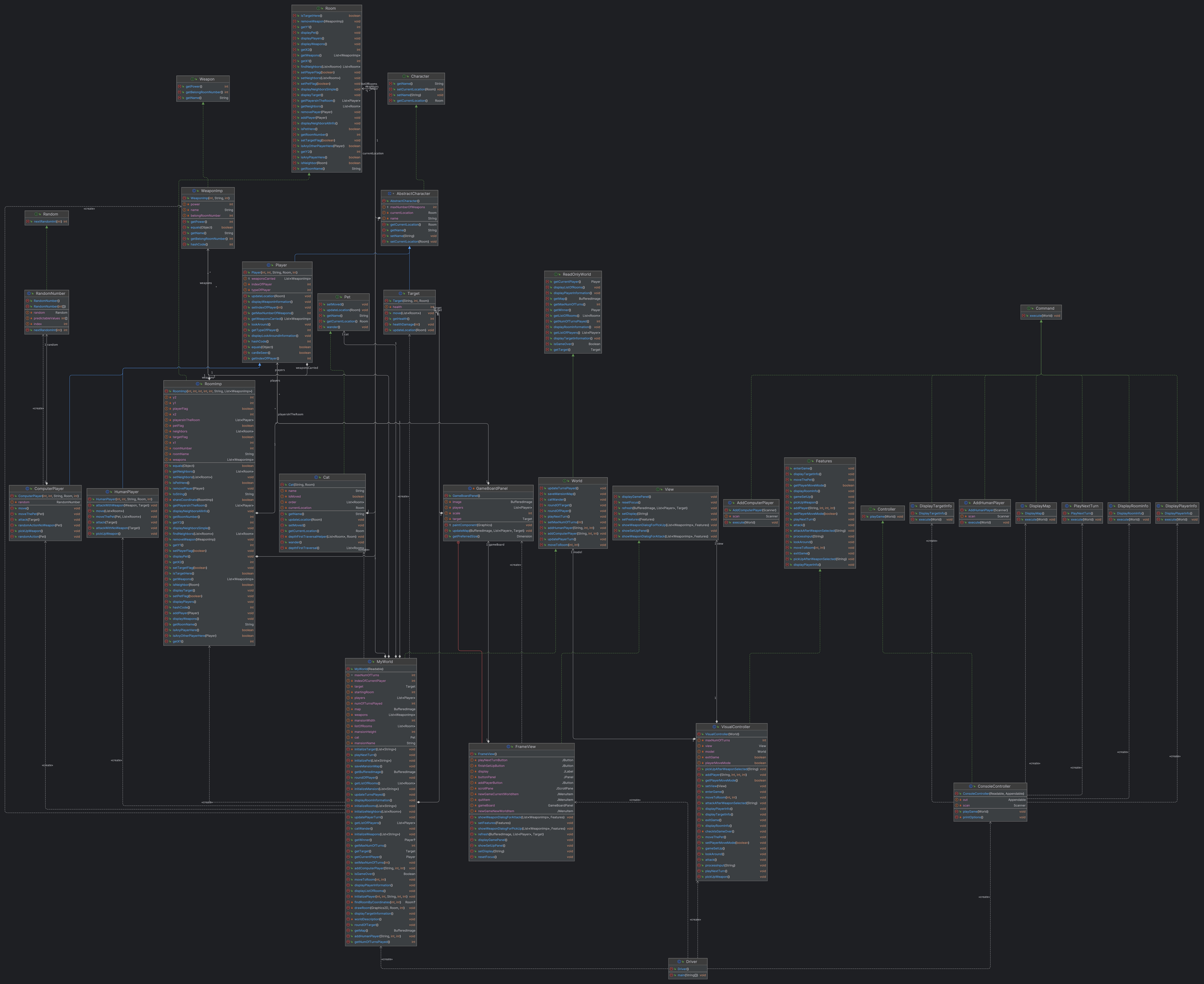
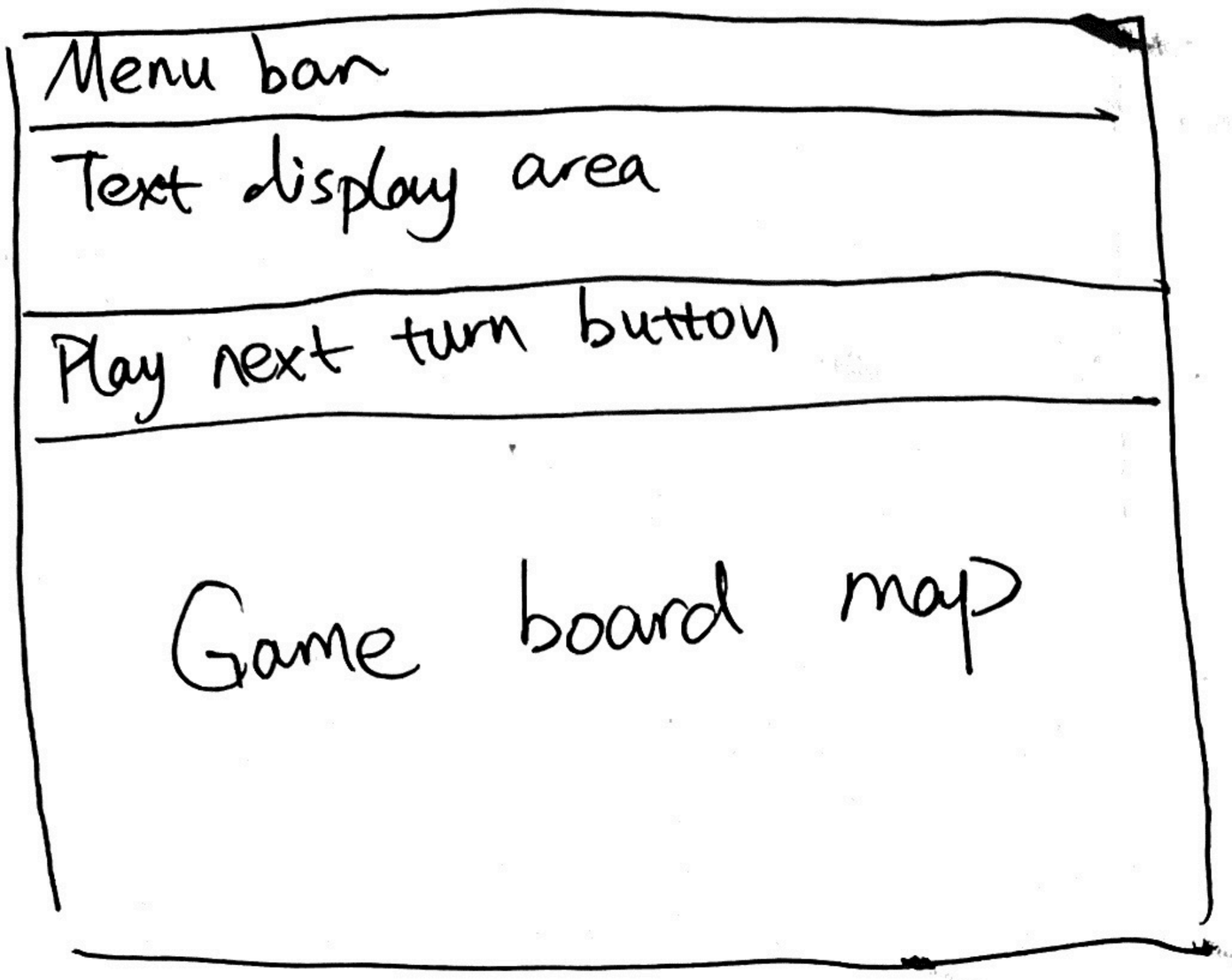
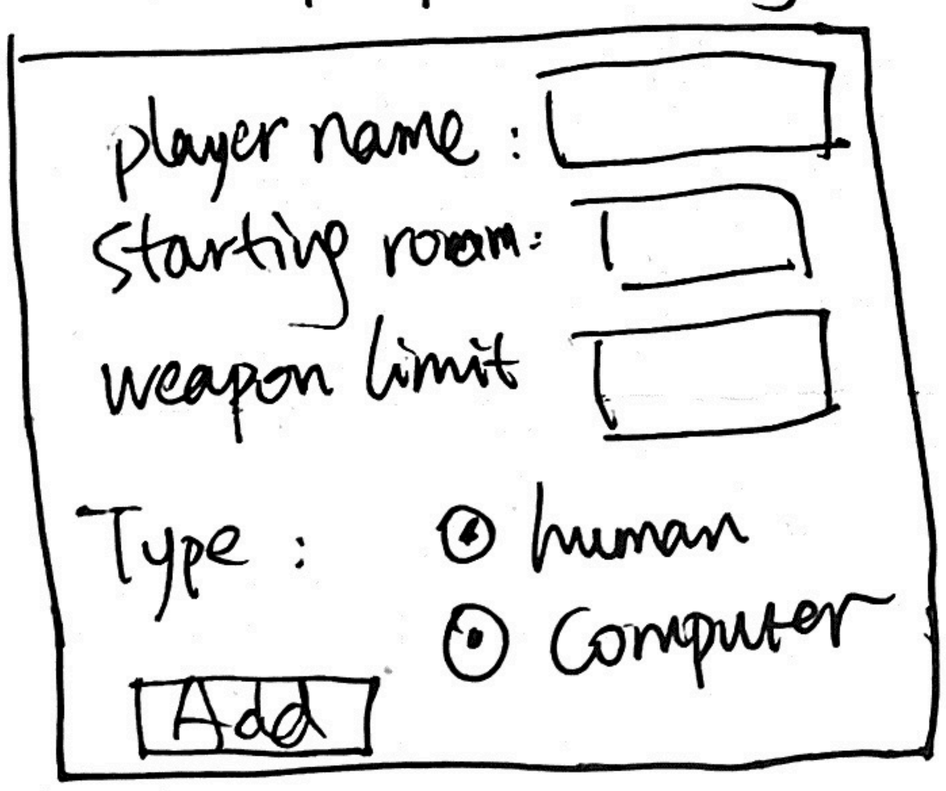

Room
isTargetHere() boolean
removeWeapon(WeaponImp) void
getY1() int
displayPet() void
displayPlayers() void
displayWeapons() void
getX2() int
getWeapons() List<WeaponImp>
getX1() int
findNeighbors(List<Room>) List<Room>
setPlayerFlag(boolean) void
setNeighbors(List<Room>) void
setPetFlag(boolean) void
displayNeighborsSimple() void
displayTarget() void
getPlayersInTheRoom() List<Player>
getNeighbors() List<Room>
removePlayer(Player) void
addPlayer(Player) void
displayNeighborsAllInfo() void
isPetHere() boolean
getRoomNumber() int
setTargetFlag(boolean) void
isAnyOtherPlayerHere(Player) boolean
getY2() int
isAnyPlayerHere() boolean
isNeighbor(Room) boolean
getRoomName() String
listOfRooms
neighbors
1..*
1
currentLocation
Weapon
getPower() int
getBelongRoomNumber() int
getName() String
Character
getName() String
setCurrentLocation(Room) void
setName(String) void
getCurrentLocation() Room
WeaponImp
WeaponImp(int, String, int)
power int
name String
belongRoomNumber int
getPower() int
equals(Object) boolean
getName() String
getBelongRoomNumber() int
hashCode() int
«create»
Random
nextRandomInt(int) int
AbstractCharacter
AbstractCharacter()
maxNumberOfWeapons int
currentLocation Room
name String
getCurrentLocation() Room
getName() String
setName(String) void
setCurrentLocation(Room) void
Player
Player(int, int, String, Room, int)
weaponsCarried List<WeaponImp>
indexOfPlayer int
typeOfPlayer int
updateLocation(Room) void
displayWeaponInformation() void
setIndexOfPlayer(int) void
getMaxNumberOfWeapons() int
getWeaponsCarried() List<WeaponImp>
lookAround() void
getTypeOfPlayer() int
displayLookAroundInformation() void
hashCode() int
equals(Object) boolean
canBeSeen() boolean
getIndexOfPlayer() int
weapons
*
RandomNumber
RandomNumber()
RandomNumber(int[])
random Random
predictableValues int[]
index int
nextRandomInt(int) int
Pet
setMoved() void
updateLocation(Room) void
getName() String
getCurrentLocation() Room
wander() void
Target
Target(String, int, Room)
health int
move(List<Room>) void
getHealth() int
healthDamage(int) void
updateLocation(Room) void
target
1 cat
1 random
weapons
weaponsCarried
players
ReadOnlyWorld
getCurrentPlayer() Player
displayListOfRooms() void
displayPlayerInformation() void
getMap() BufferedImage
getMaxNumOfTurns() int
getWinner() Player
getListOfRooms() List<Room>
getNumOfTurnsPlayed() int
displayRoomInformation() void
getListOfPlayers() List<Player>
displayTargetInformation() void
isGameOver() Boolean
getTarget() Target
Command
execute(World) void
RoomImp
RoomImp(int, int, int, int, int, String, List<WeaponImp>)
y2 int
y1 int
playerFlag boolean
x2 int
playersInTheRoom List<Player>
petFlag boolean
neighbors List<Room>
targetFlag boolean
x1 int
roomNumber int
roomName String
weapons List<WeaponImp>
equals(Object) boolean
getNeighbors() List<Room>
setNeighbors(List<Room>) void
isPetHere() boolean
removePlayer(Player) void
toString() String
shareCoordinate(RoomImp) boolean
getPlayersInTheRoom() List<Player>
displayNeighborsAllInfo() void
getRoomNumber() int
getY2() int
displayNeighborsSimple() void
findNeighbors(List<Room>) List<Room>
removeWeapon(WeaponImp) void
getY1() int
setPlayerFlag(boolean) void
displayPet() void
getX2() int
setTargetFlag(boolean) void
isTargetHere() boolean
getWeapons() List<WeaponImp>
isNeighbor(Room) boolean
displayTarget() void
setPetFlag(boolean) void
displayPlayers() void
hashCode() int
addPlayer(Player) void
displayWeapons() void
getRoomName() String
isAnyPlayerHere() boolean
isAnyOtherPlayerHere(Player) boolean
getX1() int
playersInTheRoom
ComputerPlayer
ComputerPlayer(int, int, String, Room, int)
random RandomNumber
move() void
moveThePet(Pet) void
attack(Target) void
randomActionNoWeapon(Pet) void
pickUpWeapon() void
randomAction(Pet) void
HumanPlayer
HumanPlayer(int, int, String, Room, int)
attackWithWeapon(Weapon, Target) void
move(List<Room>) void
moveThePet(Pet, List<Room>) void
attack(Target) void
attackWithNoWeapon(Target) void
pickUpWeapon() void
Cat
Cat(String, Room)
name String
isMoved boolean
order List<Room>
currentLocation Room
getName() String
updateLocation(Room) void
setMoved() void
getCurrentLocation() Room
depthFirstTraversalHelper(List<Room>, Room) void
wander() void
depthFirstTraversal() List<Room>
GameBoardPanel
GameBoardPanel()
image BufferedImage
players List<Player>
scale int
target Target
paintComponent(Graphics) void
updateMap(BufferedImage, List<Player>, Target) void
getPreferredSize() Dimension
1 gameBoard
World
updateTurnsPlayed() void
saveMansionMap() void
catWander() void
roundOfTarget() void
roundOfPlayer() void
playNextTurn() void
setMaxNumOfTurns(int) void
addHumanPlayer(String, int, int) void
addComputerPlayer(String, int, int) void
updatePlayerTurn() void
moveToRoom(int, int) void
1 model
View
displayGamePanel() void
resetFocus() void
refresh(BufferedImage, List<Player>, Target) void
setDisplay(String) void
setFeatures(Features) void
showWeaponDialogForPickUp(List<WeaponImp>, Features) void
showSetUpPanel() void
showWeaponDialogForAttack(List<WeaponImp>, Features) void
1 view
AddComputerPlayer
AddComputerPlayer(Scanner)
scan Scanner
execute(World) void
Features
enterGame() void
displayTargetInfo() void
movePlayerMoveMode() void
displayRoomInfo() void
gameSetUp() void
pickUpWeapon() void
addPlayer(String, int, int, int) void
setPlayerMoveMode(boolean) void
playNextTurn() void
attack() void
attackAfterWeaponSelected(String) void
processInput(String) void
lookAround() void
moveToRoom(int, int) void
exitGame() void
pickUpAfterWeaponSelected(String) void
displayPlayerInfo() void
Controller
playGame(World) void
DisplayTargetInfo
DisplayTargetInfo()
execute(World) void
AddHumanPlayer
AddHumanPlayer(Scanner)
scan Scanner
execute(World) void
DisplayMap
DisplayMap()
execute(World) void
PlayNextTurn
PlayNextTurn()
execute(World) void
DisplayRoomInfo
DisplayRoomInfo()
execute(World) void
DisplayPlayerInfo
DisplayPlayerInfo()
execute(World) void
«create»
MyWorld
MyWorld(Readable)
maxNumOfTurns int
indexOfCurrentPlayer int
target Target
startingRoom int
players List<Player>
numOfTurnsPlayed int
map BufferedImage
weapons List<WeaponImp>
mansionWidth int
listOfRooms List<Room>
mansionHeight int
cat Pet
mansionName String
initializeTarget(List<String>) void
playNextTurn() void
initializePet(List<String>) void
saveMansionMap() void
getBufferedImage() BufferedImage
roundOfPlayer() void
getListOfRooms() List<Room>
initializeMansion(List<String>) void
updateTurnsPlayed() void
displayRoomInformation() void
initializeRooms(List<String>) void
initializeNeighbors(List<Room>) void
updatePlayerTurn() void
getListOfPlayers() List<Player>
catWander() void
initializeWeapons(List<String>) void
getWinner() Player?
getMaxNumOfTurns() int
getTarget() Target
getCurrentPlayer() Player
setMaxNumOfTurns(int) void
addComputerPlayer(String, int, int) void
isGameOver() Boolean
moveToRoom(int, int) void
displayPlayerInformation() void
displayListOfRooms() void
initializePlayer(int, int, String, int, int) void
findRoomByCoordinates(int, int) Room?
drawRoom(Graphics2D, Room, int) void
displayTargetInformation() void
worldDescription() void
roundOfTarget() void
getMap() BufferedImage
addHumanPlayer(String, int, int) void
getNumOfTurnsPlayed() int
FrameView
FrameView()
playNextTurnButton JButton
finishSetUpButton JButton
display JLabel
buttonPanel JPanel
addPlayerButton JButton
scrollPane JScrollPane
newGameCurrentWorldItem JMenuItem
quitItem JMenuItem
gameBoard GameBoardPanel
newGameNewWorldItem JMenuItem
showWeaponDialogForAttack(List<WeaponImp>, Features) void
setFeatures(Features) void
showWeaponDialogForPickUp(List<WeaponImp>, Features) void
refresh(BufferedImage, List<Player>, Target) void
displayGamePanel() void
showSetUpPanel() void
setDisplay(String) void
resetFocus() void
VisualController
VisualController(World)
maxNumOfTurns int
view View
model World
exitGame boolean
playerMoveMode boolean
pickUpAfterWeaponSelected(String) void
addPlayer(String, int, int, int) void
getPlayerMoveMode() boolean
setView(View) void
enterGame() void
moveToRoom(int, int) void
attackAfterWeaponSelected(String) void
displayPlayerInfo() void
displayTargetInfo() void
exitGame() void
displayRoomInfo() void
checkIsGameOver() void
moveThePet() void
setPlayerMoveMode(boolean) void
gameSetUp() void
lookAround() void
attack() void
processInput(String) void
playNextTurn() void
pickUpWeapon() void
ConsoleController
ConsoleController(Readable, Appendable)
out Appendable
scan Scanner
playGame(World) void
printOptions() void
Driver
Driver()
main(String[]) void

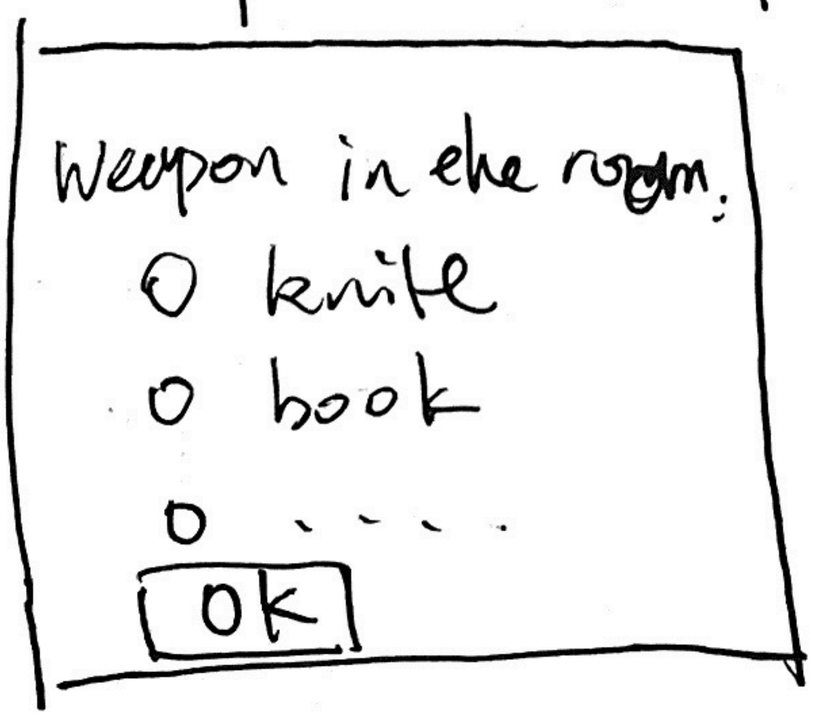
Main Window



Add player dialog



Weapon Selection panel



Testing Plan for Milestone 4

VisualController Class/Features Interface

In general, create a mock model and mock view to do the test for controller. Besides, conduct UAT tests for views.

1. Exit Game:

- Scenario: user chooses to exit the game
- Steps: call exitGame() method
- Expected Behavior:
    - exitGame flag is set to true
    - Game exits

2. Game Setup:

- Scenario: starting a new game and setting up players
- Steps:
    - call gameSetUp() method
    - simulate user adding players.
    - call enterGame() method
- Expected Behavior:
    - welcome message is displayed
    - after adding players, the game starts, and the corresponding message is passed to view.
- Scenario: adding more than 10 players should be unsuccessful
- Steps:
    - call gameSetUp() method
    - simulate user adding the 11th player
- Expected Behavior:
    - make sure the corresponding message is passed to view.

3. Play Next Turn:

- Scenario: simulate a human player's turn.
- Steps:

    - call playNextTurn() method.
    - simulate player actions (e.g., moving, picking up weapons, attacking).
- Expected Behavior:
    - player actions are passed to the view
    - check if the game ends under different conditions (e.g., max turns reached, exit game).

4. Attack (Human Player):

- Scenario: simulate a human player attacking
- Steps:
    - call attack() method.
    - simulate player choosing weapons
    - simulate the attack
- Expected Behavior:
    - if no weapon, check if the poking happens.
    - if there are weapons, check if the selected weapon is passed to the controller correctly, and check if attack is performed correctly in the model
    - check if the target's health is deducted when attack is successful and the information is passed to the view

5. Pick Up Weapon (Human Player):

- Scenario: simulate a human player picking up a weapon.
- Steps:
    - call pickUpWeapon() method.
    - simulate player choosing a weapon.
    - simulate the pickup
- Expected Behavior:
    - if there is no weapon, check if the inventory has changed.
    - if there are weapons, check if the weapon is added to the player's inventory.
    - check if weapon removed from the room after being picked up.
    - check if information is delivered correctly to the view.

6. Move to Room (Human Player):

- Scenario: simulate a human player moving to a neighboring room.
- Steps:
    - call moveToRoom() method with coordinates of a neighboring room.
- Expected Behavior:
    - check if the player's location is updated.
    - check if the display shows the new player location.

7. Look Around:

- Scenario: simulate looking around.
- Steps: call lookAround() method.
- Expected Behavior:
    - check if the view shows information about nearby rooms.

8. Player Move Mode:

- Scenario: check player move mode.
- Steps:
    - call setPlayerMoveMode(true) and setPlayerMoveMode(false).
- Expected Behavior:
    - check if the player move mode flag is correctly set and unset.

9. Display Information for Room:

- Scenario: display information for a room.
- Steps:
    - create a mock room.
    - call displayRoomInfo() with the mock room.
    - call again
- Expected Behavior:
    - information about the room is displayed in the UI.
    - check that the displayed information matches the expected details of the mock room.
    - check if the displayed information is as expected when called again

10. Display Information for Valid Player:

- Scenario: display information for a player.
- Steps:
    - create a mock player

    - call displayPlayerInfo() with the mock player
    - call gain
- Expected Behavior:
    - information about the player is displayed in the UI
    - check that the displayed information matches the expected details of the mock player
    - check if the displayed information is as expected when called again

11. Display Information for Valid Target:

- Scenario: display information for the target.
- Steps:
    - create a mock target
    - call displayTargetInfo() with the mock target.
    - call again
- Expected Behavior:
    - information about the target is displayed in the UI.
    - check that the displayed information matches the expected details of the mock target.
    - check if the displayed information is as expected when called again

World Interface

In order to be compatible with the GUI, several methods should be broken down to methods with single functionality. Should conduct unit tests for the newly implemented methods. For example:

1. Testing Plan for moveToRoom(int x, int y) Method

- Scenario: move to a neighbor room
- Steps:
    - set up a game environment with a player in a specific room.
    - call moveToRoom(x, y) with the coordinates of a neighboring room.
- Expected Behavior:
    - the player's location is updated to the specified neighboring room.
    - verify that the player's current location matches the expected room after the move.
- Scenario: move to a non-neighbor room

- Steps:
    - set up a game environment with a player in a specific room.
    - call moveToRoom(x, y) with the coordinates of a non-neighboring room.
- Expected Behavior:
    - the player's location remains unchanged.
    - verify that the player's current location is still the initial room after the method call.

2. Testing Plan for findRoomByCoordinates(int x, int y) Method

- Scenario: find room with coordinates (x, y) that is within the boundaries of a room
- Steps:
    - set up a game environment with list of rooms.
    - call findRoomByCoordinates(x, y) with the coordinates within the boundaries of a specific room.
- Expected Behavior:
    - the method returns the correct room
- Scenario: find room with coordinates (x, y) that do not fall within the boundaries of any room
- Steps:
    - set up a game environment with known rooms.
    - call findRoomByCoordinates(x, y) with coordinates outside the boundaries of all rooms.
- Expected Behavior:
    - the method should return null, indicating that there is no room containing the specified coordinates.