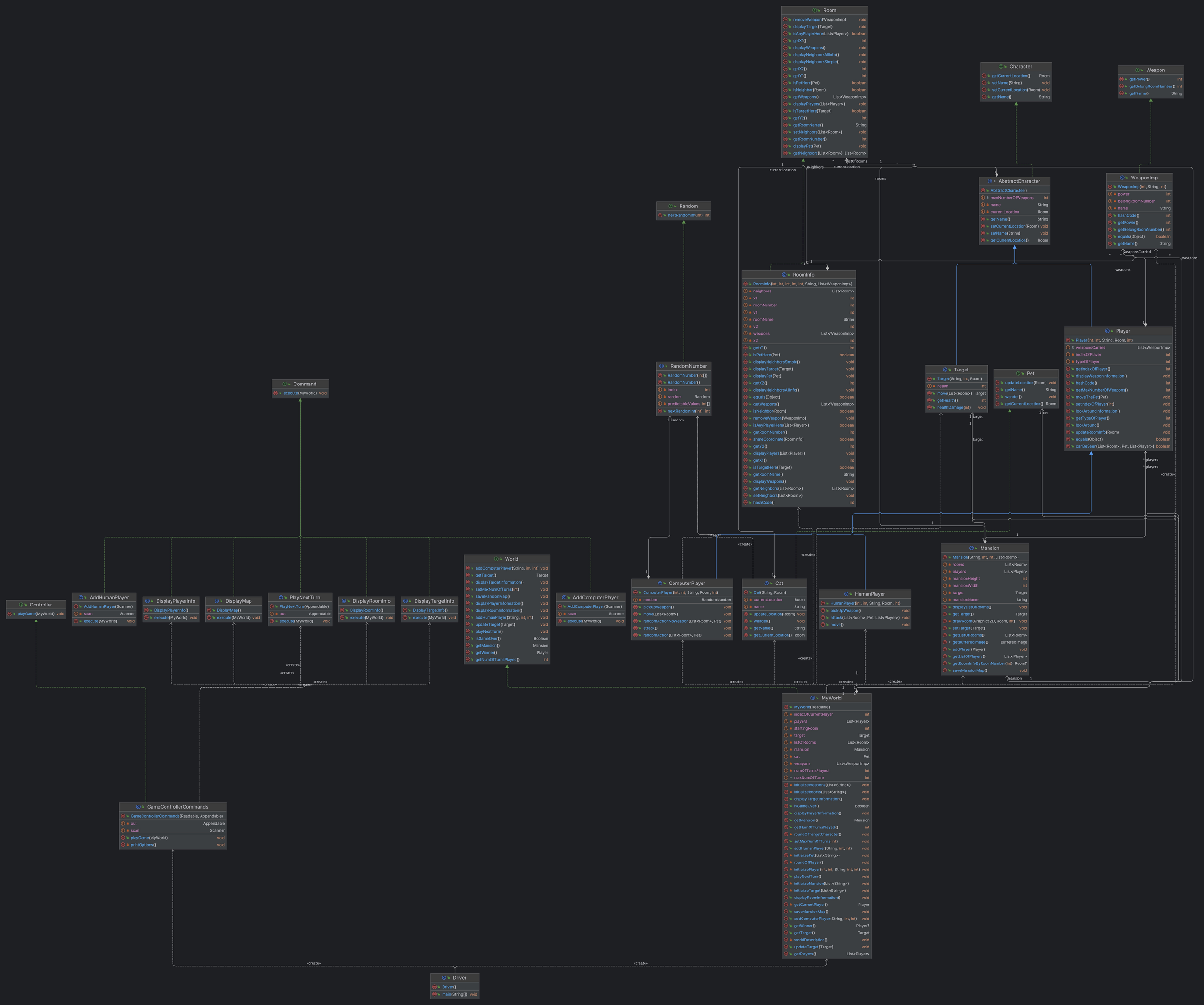
Room
removeWeapon(WeaponImp) void
displayTarget(Target) void
isAnyPlayerHere(List<Player>) boolean
getX1() int
displayWeapons() void
displayNeighborsAllInfo() void
displayNeighborsSimple() void
getX2() int
getY1() int
isPetHere(Pet) boolean
isNeighbor(Room) boolean
getWeapons() List<WeaponImp>
displayPlayers(List<Player>) void
isTargetHere(Target) boolean
getY2() int
getRoomName() String
setNeighbors(List<Room>) void
getRoomNumber() int
displayPet(Pet) void
getNeighbors(List<Room>) List<Room>
Character
getCurrentLocation() Room
setName(String) void
setCurrentLocation(Room) void
getName() String
Weapon
getPower() int
getBelongRoomNumber() int
getName() String
neighbors
currentLocation
listOfRooms
rooms
AbstractCharacter
AbstractCharacter()
maxNumberOfWeapons int
name String
currentLocation Room
getName() String
setCurrentLocation(Room) void
setName(String) void
getCurrentLocation() Room
WeaponImp
WeaponImp(int, String, int)
power int
belongRoomNumber int
name String
hashCode() int
getPower() int
getBelongRoomNumber() int
equals(Object) boolean
getName() String
weaponsCarried
weapons
Random
nextRandomInt(int) int
RandomNumber
RandomNumber(int[])
RandomNumber()
index int
random Random
predictableValues int[]
nextRandomInt(int) int
RoomInfo
RoomInfo(int, int, int, int, int, String, List<WeaponImp>)
neighbors List<Room>
x1 int
roomNumber int
y1 int
roomName String
y2 int
weapons List<WeaponImp>
x2 int
getY1() int
isPetHere(Pet) boolean
displayNeighborsSimple() void
displayTarget(Target) void
displayPet(Pet) void
getX2() int
displayNeighborsAllInfo() void
equals(Object) boolean
getWeapons() List<WeaponImp>
isNeighbor(Room) boolean
removeWeapon(WeaponImp) void
isAnyPlayerHere(List<Player>) boolean
getRoomNumber() int
shareCoordinate(RoomInfo) boolean
getY2() int
displayPlayers(List<Player>) void
getX1() int
isTargetHere(Target) boolean
getRoomName() String
displayWeapons() void
getNeighbors(List<Room>) List<Room>
setNeighbors(List<Room>) void
hashCode() int
Player
Player(int, int, String, Room, int)
weaponsCarried List<WeaponImp>
indexOfPlayer int
typeOfPlayer int
getIndexOfPlayer() int
displayWeaponInformation() void
hashCode() int
getMaxNumberOfWeapons() int
moveThePet(Pet) void
setIndexOfPlayer(int) void
lookAroundInformation() void
getTypeOfPlayer() int
lookAround() void
updateRoomInfo(Room) void
equals(Object) boolean
canBeSeen(List<Room>, Pet, List<Player>) boolean
Target
Target(String, int, Room)
health int
move(List<Room>) Target
getHealth() int
healthDamage(int) void
Pet
updateLocation(Room) void
getName() String
wander() void
getCurrentLocation() Room
target
cat
players
Command
execute(MyWorld) void
World
addComputerPlayer(String, int, int) void
getTarget() Target
displayTargetInformation() void
setMaxNumOfTurns(int) void
saveMansionMap() void
displayPlayerInformation() void
displayRoomInformation() void
addHumanPlayer(String, int, int) void
updateTarget(Target) void
playNextTurn() void
isGameOver() Boolean
getMansion() Mansion
getWinner() Player
getNumOfTurnsPlayed() int
Mansion
Mansion(String, int, int, List<Room>)
rooms List<Room>
players List<Player>
mansionHeight int
mansionWidth int
target Target
mansionName String
displayListOfRooms() void
getTarget() Target
drawRoom(Graphics2D, Room, int) void
setTarget(Target) void
getListOfRooms() List<Room>
getBufferedImage() BufferedImage
addPlayer(Player) void
getListOfPlayers() List<Player>
getRoomInfoByRoomNumber(int) Room?
saveMansionMap() void
mansion
ComputerPlayer
ComputerPlayer(int, int, String, Room, int)
random RandomNumber
pickUpWeapon() void
move(List<Room>) void
randomActionNoWeapon(List<Room>, Pet) void
attack() void
randomAction(List<Room>, Pet) void
Cat
Cat(String, Room)
currentLocation Room
name String
updateLocation(Room) void
wander() void
getName() String
getCurrentLocation() Room
HumanPlayer
HumanPlayer(int, int, String, Room, int)
pickUpWeapon() void
attack(List<Room>, Pet, List<Players>) void
move() void
Controller
playGame(MyWorld) void
AddHumanPlayer
AddHumanPlayer(Scanner)
scan Scanner
execute(MyWorld) void
DisplayPlayerInfo
DisplayPlayerInfo()
execute(MyWorld) void
DisplayMap
DisplayMap()
execute(MyWorld) void
PlayNextTurn
PlayNextTurn(Appendable)
out Appendable
execute(MyWorld) void
DisplayRoomInfo
DisplayRoomInfo()
execute(MyWorld) void
DisplayTargetInfo
DisplayTargetInfo()
execute(MyWorld) void
AddComputerPlayer
AddComputerPlayer(Scanner)
scan Scanner
execute(MyWorld) void
GameControllerCommands
GameControllerCommands(Readable, Appendable)
out Appendable
scan Scanner
playGame(MyWorld) void
printOptions() void
MyWorld
MyWorld(Readable)
indexOfCurrentPlayer int
players List<Player>
startingRoom int
target Target
listOfRooms List<Room>
mansion Mansion
cat Pet
weapons List<WeaponImp>
numOfTurnsPlayed int
maxNumOfTurns int
initializeWeapons(List<String>) void
initializeRooms(List<String>) void
displayTargetInformation() void
isGameOver() Boolean
displayPlayerInformation() void
getMansion() Mansion
getNumOfTurnsPlayed() int
roundOfTargetCharacter() void
setMaxNumOfTurns(int) void
addHumanPlayer(String, int, int) void
initializePet(List<String>) void
roundOfPlayer() void
initializePlayer(int, int, String, int, int) void
playNextTurn() void
initializeMansion(List<String>) void
initializeTarget(List<String>) void
displayRoomInformation() void
getCurrentPlayer() Player
saveMansionMap() void
addComputerPlayer(String, int, int) void
getWinner() Player?
getTarget() Target
worldDescription() void
updateTarget(Target) void
getPlayers() List<Player>
Driver
Driver()
main(String[]) void
«create»

Testing Plan for Milestone 3

Player Class

1. **canBeSeen() Method Testing:**

- Condition: Testing with a pet in the current room, with or without a player in neighboring room.
    - Example: Set up a test where there is a pet in the current room and no player in neighboring room.
    - Expected: Verify that the method returns false.
    - Example: Set up a test where there is a pet in the current room and a player in neighboring room.
    - Expected: Verify that the method returns false.
- Condition: Testing with a player in neighboring rooms and no pet in the current room.
    - Example: Setup a test where there is a player in one of the neighboring rooms and no pet in the current room.
    - Expected: Verify that the method returns true.
- Condition: Testing with no player in neighboring rooms and no pet in the current room.
    - Example: Setup a test where there is no player in any neighboring rooms and no pet in the current room.
    - Expected: Verify that the method returns false.
- Condition: Testing with null inputs.
    - Example: Test the method with null inputs for rooms, pet, and players.
    - Expected: Verify that the method handles null inputs gracefully without throwing exceptions.

2. **moveThePet() Method Testing:**

- Condition: Testing the method with a Pet object in a specific location.
    - Example: Create a Pet object in a specific location, pass it to the method, and check its location after the method is called.
    - Expected: Verify that the Pet object is moved from its initial location and placed in a new location.

Pet Interface (Cat Class)

1. **Constructor Testing:**

• Condition: Testing a Cat object is created with the provided name and current location.

- Example: Instantiate the instance with a name and current location.
- Expected: Verify that a Cat object is created with the provided name and current location.

2. **wander() Method Testing:**

• Condition: Testing the method to ensure that the cat moves to a new location, follow the rule of depth-first traversal.

- Example: Invoke the wander method and check the cat's current location.
- Expected: Verify that the cat's current location is updated correctly after wandering.

3. **getName() Method Testing:**

• Condition: Testing if the method returns the correct name of the cat.

- Example: Create a Cat instance with a specific name and call the method.
- Expected: Ensure that the method returns the correct name of the cat.

4. **getCurrentLocation() Method Testing:**

• Condition: Testing if the method returns the current location of the cat.

- Example: Create a Cat instance and verify its current location.
- Expected: Ensure that the method returns the correct current location of the cat.

5. **updateLocation() Method Testing:**

• Condition: Testing if the method successfully updates the current location of the cat.

- Example: Update the cat's current location and then retrieve the updated location.
- Expected: Verify that the method correctly updates the current location of the cat.

6. **Testing the starting location:**

• Condition: Testing if the pet starts in room 0.

- Example: Get the room number after initialization.
- Expected: Verify that the room number is 0.

HumanPlayer Class

1. **attack() Method Testing:**

• Condition: Testing the attack method when the player has no weapons.

- Example: Call the method with an empty list of carried weapons.
- Expected: Ensure that the method informs the player about the lack of weapons and executes the "poke the target in the eye" action.

• Condition: Testing the attack method when the player has weapons and selects one to attack the target.

- Example: Simulate the scenario where the player has multiple weapons, selects one, and attacks the target.
- Expected: Verify that the method correctly displays the weapon options, allows the player to choose one, and executes the attack based on the chosen weapon.

• Condition: Testing the poke attack when the player chooses to poke the target in the eye.

- Example: Choose the poke attack option and evaluate the outcome.
- Expected: Verify that the method correctly assesses the visibility status of the player and makes damage adjustments accordingly.

• Condition: Testing if the method correctly removes the selected weapon from the player's inventory after an attack.

- Example: Perform an attack using a specific weapon and then check the player's remaining inventory.
- Expected: Ensure that the method removes the selected weapon from the player's inventory and updates the inventory accordingly.

• Condition: Testing the visibility check function to verify if the player can be seen by others.

- Example: Execute the method and evaluate the visibility status of the player.
- Expected: Verify that the method accurately determines the visibility status of the player based on the surrounding rooms, pets, and players.

• Condition: Testing the health damage functionality when the attack is successful.

- Example: Attack the target successfully using a specific weapon and check the target's health.
- Expected: Ensure that the method reduces the target's health based on the power of the selected weapon and the visibility status of the player.

ComputerPlayer Class

1. **attack() Method Testing:**

• Condition: Testing the attack method when the player has no weapons.

- Example: Call the method with an empty list of carried weapons.
- Expected: Ensure that the method and executes the "poke the target in the eye" action.

• Condition: Testing the attack method when the computer player has only one weapon.

- Example: Simulate the scenario where the computer player has one weapon and executes the attack.
- Expected: Verify that the method uses the available weapon to attack the target and correctly reduces the target's health based on the weapon's power.

• Condition: Testing the attack method when the computer player has multiple weapons.

- Example: Simulate the scenario where the computer player has multiple weapons and executes the attack.
- Expected: Ensure that the method selects the weapon with the highest power, uses it to attack the target, and correctly reduces the target's health based on the chosen weapon's power.

• Condition: Testing if the method correctly removes the selected weapon from the player's inventory after an attack.

- Example: Perform an attack using a specific weapon and then check the player's remaining inventory.
- Expected: Ensure that the method removes the selected weapon from the player's inventory and updates the inventory accordingly.

• Condition: Testing the health damage functionality when the attack is successful.

- Example: Execute the attack and check the target's health after the attack.
- Expected: Ensure that the method reduces the target's health based on the selected weapon's power and accurately reflects the damage inflicted by the computer player.

Room Interface

1. **isPetHere() Method Testing:**

• Condition: Testing the method when the pet is in the room.

- Example: Simulate the scenario where the pet is in the room and invoke the method.
- Expected: Verify that the method correctly returns true.

• Condition: Testing the method when the pet is not in the room.

- Example: Simulate the scenario where the pet is not in the room and execute the method.
- Expected: Ensure that the method returns false.

2. **displayPet() Method Testing:**

• Condition: Testing the method to confirm the accurate display of pet information.

- Example: Provide the pet as input and call the method to display the pet information in the room.
- Expected: Verify that the method prints the pet's name, the room number, and the room name correctly.

Refactored Method of MyWorld Class

1. **roundOfPlayer() Method Testing the new functionalities:**

• Condition: Testing the method behavior when the target is present in the current room.

- Example: Simulate the scenario where the target is in the current room and test the method's behavior.
- Expected: Verify that the method handles the presence of the target, prompts the player for appropriate actions, and facilitates the attack or other valid actions as necessary.

• Condition: Testing the method behavior when the target is not present in the current room.

- Example: Simulate the scenario where the target is not in the current room and assess the method's behavior.
- Expected: Ensure that the method handles the absence of the target appropriately, prompts the player for valid actions, and executes the available actions accordingly.

• Condition: Testing the method when it is the turn of a human player.

- Example: Simulate the scenario where a human player's turn is triggered and provide the required input accordingly.
- Expected: Verify that the method accurately prompts the player for action, handles user input, and executes the chosen action correctly.

• Condition: Testing the method when it is the turn of a computer player.

- Example: Simulate the scenario where a computer player's turn is triggered and observe its action.
- Expected: Ensure that the computer player has different choices when there is chance to attack or not. If can not be seen, computer player must attack, otherwise randomly chooses a valid action.

DisplayTargetInfo Command Class

1. **execute() Method Testing:**

• Condition: Testing the method with a valid World instance.

- Example: Initialize the command with a valid World instance and execute it.
- Expected: Verify that the method of the World class is executed without errors.

• Condition: Testing the method with a null World instance.

- Example: Initialize the command with a null World instance and execute it.
- Expected: Verify that the method gracefully handles the null input and does not cause any unexpected errors.