world.base

BasePet

- name: String
- spaceIndex: int

+ BasePet(name: String)
+ getName(): String
+ getSpaceIndex(): int
+ setSpaceIndex(index: int)

BaseSpace

- startX: int
- startY: int
- endX: int
- endY: int
- order: int
- name: String

+ BaseSpace(x1: int, y1: int, x2: int, y2: int, order: int, name: String)
+ getStart(): int[]
+ getEnd(): int[]
+ getOrder(): int
+ getName(): String
+ getNeighbors(): List<Space>
+ getOccupiers(): List<Player>
+ getWeapons(): List<BaseWeapon>
+ isExposed(): Boolean
+ toString(): String

BaseWeapon

- space: int
- damage: int
- name: String
- holder: String

+ BaseWeapon(s: int, d: int, str: String)
+ getSpaceIndex(): int
+ setHolder(holder: String)
+ getHolder(): String
+ getDamage(): int
+ getName(): String

BasePlayer

- order: int
- name: String
- spaceIndex: int
- type: PlayerType
- weaponLimit: Integer

+ BasePlayer(order: int, name: String, spaceIndex: int, type: PlayerType, limit: Integer)
+ getOrder(): int
+ getName(): String
+ setSpaceIndex(index: int)
+ getSpaceIndex(): int
+ getWeaponLimit(): Integer
+ getType(): PlayerType
+ addWeapon(weapon: Weapon): Boolean
+ getWeapons(): List<Weapon>

world.model

<<Interface>>
Pet

+ getName(): String
+ getSpaceIndex(): int
+ setSpaceIndex(index: int)

<<Interface>>
Space

+ getName(): String
+ getOrder(): int
+ getNeighbors(): List<Space>
+ getOccupiers(): List<Player>
+ getWeapons(): List<BaseWeapon>
+ isExposed(): Boolean

Target

- health: int
- name: String
- position: int

+ Target(health: int, name: String)
+ getHealth(): int
+ decreaseHealth(damage: int): int
+ getName(): String
+ getPosition(): int
+ setPosition(pos: int)

<<Interface>>
Player

+ getOrder(): int
+ getName(): String
+ setSpaceIndex(index: int)
+ getSpaceIndex(): int
+ getWeaponLimit(): Integer
+ getType(): PlayerType
+ addWeapon(weapon: Weapon): Boolean
+ getWeapons(): List<Weapon>

world.enums

<<enumeration>>
PlayerType

HUMAN
COMPUTER

world.context

Context

- map: Map<String, Object>

+ setM(m: Integer)
+ getM(): Integer
+ setN(n: Integer)
+ getN(): Integer
+ setWorldName(name: String)
+ getWorldName(): String
+ setTarget(target: Target)
+ getTarget(): Target
+ setPet(pet: Pet)
+ getPet(): Pet
+ setSpaces(spaces: List<BaseSpace>)
+ getSpaces(): List<BaseSpace>
+ setNeighborMap(map: Map<BaseSpace, List<BaseSpace>>)
+ getNeighborMap(): Map<BaseSpace, List<BaseSpace>>
+ setExposedSpaces(spaces: Set<Space>)
+ getExposedSpaces(): Set<Space>
+ setWeapons(weapons: List<BaseWeapon>)
+ getWeapons(): List<BaseWeapon>
+ setEvidences(evidences: List<BaseWeapon>)
+ getEvidences(): List<BaseWeapon>
+ setPlayers(players: List<Player>)
+ getPlayers(): List<Player>

ContextBuilder

+ build(in: Readable): Context
- initMatrixAndValidateSpaces(m: int, n: int, spaces: List<BaseSpace>): int[][]
- initNeighborMapAndSpaces(m: int, n: int, matrix: int[][], spaces: List<BaseSpace>, context Context)
- getNeighborsFormMatrix(space: BaseSpace, spaces: List<BaseSpace>, m: int, n: int, matrix: int[][]): List<BaseSpace>

ContextHolder

- holder: ThreadLocal<Context>

+ set(ctx: Context)
+ get(): Context
+ remove()

world

World

+ getAllSpaces(context: Context): List<String>
+ getNeighbors(context: Context, space: String): List<String>
+ getNeighbors(context: Context, index: Integer): List<String>
+ getSpace(context: Context, name: String): Space
+ getSpace(context: Context, index: Integer): Space
+ getTargetPosition(context: Context): Space
+ moveTarget(context: Context): Space
+ getAllPlayers(context: Context): List<Player>
+ getPlayer(context: Context, index: int): Player
+ addPlayer(context: Context, player: Player): Boolean
+ movePlayer(player: Player, space: Space)
+ pickUp(player: Player, weapon: BaseWeapon)
+ getGraphicalImage(context: Context): BufferedImage
+ showGraphicalImage(directory: String)

controller

<<Interface>>
WorldController

+ playGame(context: Context)

AbstractWorldController

- turn: Integer
- maxPlayerAmount: Integer
- gameEngine: GameEngine
- serviceTemplate: ServiceTemplate

determineFlowForComputer(ctx: Context, player: Player): Flow
+ setTurn(turn: Integer)
+ setMaxPlayerAmount(amount: Integer)

GameEngine

- count: int
- currentPlayer: int
- turnAmount: int
- gameOver: Boolean
- context: Context

+ GameEngine(context: Context, turnAmount: int)
+ getCurrentPlayer(): Player
+ getTurn(): Integer
+ gameOver(): Boolean
+ updateState(): Boolean

WorldGuiController

- view: WorldView
- mouseListener: MouseClickListener

- configureButtonListener()
- configureKeyboardListener()
- configureMouseClickListener()
+ setView(view: WorldView)
+ setMouseListener(listener: MouseClickListener)
+ newGame()
+ restartGame()
+ quitGame()

WorldConsoleController

- in: Readable
- out: Appendable
- scan: Scanner

+ setIn(in: Readable)
+ setOut(out: Appendable)
- createPlayer(ctx: Context, order: int)
- start(ctx: Context)
- displayPaylerDetail(ctx: Context, player: Player)

flowengine

ServiceTemplateImpl<T extends BaseRequest, R extend BaseResult>

- bizFlowExecutor: BizFlowExecutor

- validateRequest(request: T)
- initFlowContext(ctx: FlowContext, flowName: String, request: T, result: R)
- assembleSuccessResult(ctx: FlowContext)
- assembleFailResult(e: Exception)
+ execute(request: T, flowName: String, callback: BaseProcessCallBack): R

<<Interface>>
BizFlowExecutor

+ startProcess(context: FlowContext)

BizFlowExecutorImpl

- flowEnrichersMap: Map<String, List<String>>
- flowValidatorsMap: Map<String, List<String>>
- flowActionsMap: Map<String, List<String>>
- enricherMap: Map<String, Enricher>
- validatorMap: Map<String, Validator>
- actionMap: Map<String, Action>

- getEnrichers(flow: String)
- executeEnricher(enrichers: List<Enricher>, context: FlowContext)
- getValidators(flow: String)
- executeValidator(validators: List<Validator>, context: FlowContext)
- getActions(flow: String)
- executeAction(actions: List<Action>, context: FlowContext)
+ startProcess(context: FlowContext)

BaseProcessCallBack<T extends BaseRequest, R extends BaseResult>

+ enrichContext(request: T, context: FlowContext)
+ process(request: T, result: R, context: FlowContext)
+ fillResult(request: T, result: R, context: FlowContext)
+ createDefaultResult(): R

<<Interface>>
ServiceTemplate<T extends BaseRequest, R extends BaseResult>

+ execute(request: T, flowName: String, callback: BaseProcessCallBack): R

<<Interface>>
Action

+ execute(context: FlowContext)

MovePlayerAction

+ execute(context: FlowContext)

PickUpWeaponAction

+ execute(context: FlowContext)

LookAroundAction

+ execute(context: FlowContext)

MovePetAction

+ execute(context: FlowContext)

AttackRequestAssembleAction

+ execute(context: FlowContext)

DropWeaponAction

+ execute(context: FlowContext)

AttackAction

+ execute(context: FlowContext)

PetTraverseAction

+ execute(context: FlowContext)

<<Interface>>
Validator

+ validate(context: FlowContext)

NeighborValidator

+ validate(context: FlowContext)

WeaponValidator

+ execute(context: FlowContext)

SpaceValidator

+ validate(context: FlowContext)

AttackValidator

+ validate(context: FlowContext)

<<Interface>>
Enricher

+ enrich(context: FlowContext)

InputEnricher

+ enrich(context: FlowContext)

view

<<Interface>>
WorldView

+ displayWeapons(weapons: List<BaseWeapon>): String
+ displayInfo(message: String)
+ displayError(message: String)
+ restartGame(spaceNames: List<String>)
+ addPlayerScreen(spaceNames : List<String>)
+ setupScreen()
+ refreshBoard(context: Context, currentPlayer: Player, numberOfTurns: int, message: String, listener: MouseClickListener)
+ welcomeScreen()
+ quit()
+ addActionListener(buttonListener: ButtonListener)
+ addActionListener(keyboardListener: KeyboardListener)
+ confirmScreen(message: String)
+ getPlayerInput(): String[]
+ setPlayerColor(spaceNames: List<String>)
+ getSetupInput(): String[]
+ gameEndScreen(result: String)

WorldGuiView

- welcomePanel: JPanel
- sidePanel: SidePanel
- menu: GameMenu
- addPlayerPanel: JPanel
- gamePlayPanel: GamePlayPanel
- newGamePanel: JPanel
- button: JButton
- sp: JSplitPane
- playersColor: Map<String, Color>
- name: JTextField
- location: JComboBox<String>
- capacity: JTextField
- computer: JRadioButton
- human: JRadioButton
- chosenColor: Color
- playGame: JButton
- filePath: String
- maxPlayers: JTextField
- turns: JTextField

+ displayWeapons(weapons: List<BaseWeapon>): String
+ displayInfo(message: String)
+ displayError(message: String)
+ restartGame(spaceNames: List<String>)
+ addPlayerScreen(spaceNames : List<String>)
+ setupScreen()
+ refreshBoard(context: Context, currentPlayer: Player, numberOfTurns: int, message: String, listener: MouseClickListener)
+ welcomeScreen()
+ quit()
+ addActionListener(buttonListener: ButtonListener)
+ addActionListener(keyboardListener: KeyboardListener)
+ confirmScreen(message: String)
+ getPlayerInput(): String[]
+ setPlayerColor(spaceNames: List<String>)
+ getSetupInput(): String[]
+ gameEndScreen(result: String)

SidePanel

- playersPanel: JPanel
- targetPanel: JPanel
- logPanel: JPanel
- turnPanel: JPanel
- scrollPlayers: JScrollPane
- scrollTarget: JScrollPane
- scrollLog: JScrollPane
- scrollTurn: JScrollPane
- constraint: GridBagConstraints

+ SidePanel()
+ displayListOfPlayers(currentPlayer: Player, playersColor: Map<String, Color>, currentPlayerLocation: String)
+ displayTargetDetails(targetDetails: String)
+ displayTurnsLeft(numberOfTurns: int)
+ displayLog(message: String)

GameMenu

- menu: JMenu
- menuBar: JMenuBar
- newGame: JMenuItem
- restartGame: JMenuItem
- help: JMenu
- quitGame: JMenuItem
- addPlayer: JMenuItem
- aboutGame: JMenuItem
- howtoPlay: JMenuItem

+ GameMenu()
+ addActionListener(buttonListener: ButtonListener)

Icon

- url: URL

+ setIconColor(color: Color): ImageIcon

GamePlayPanel

- roomPanels: Map<String, JPanel>
- scale: int
- padding: int
- offset: int

+ GamePlayPanel()
+ displayRooms(context: Context, listener: MouseClickListen)
+ displayPlayers(players: List<Player>)
+ displayTarget(target: Target, pet: Pet, listener: MouseClickListener)

KeyboardListener

- map: Map<Integer, Runnable>

+ setMap(map: Map<Integer, Runnable>
+ keyPressed(e: KeyEvent)

ButtonListener

- map: Map<String, Runnable>

+ setMap(map: Map<String, Runnable>
+ actionPerformed(e: ActionEvent)

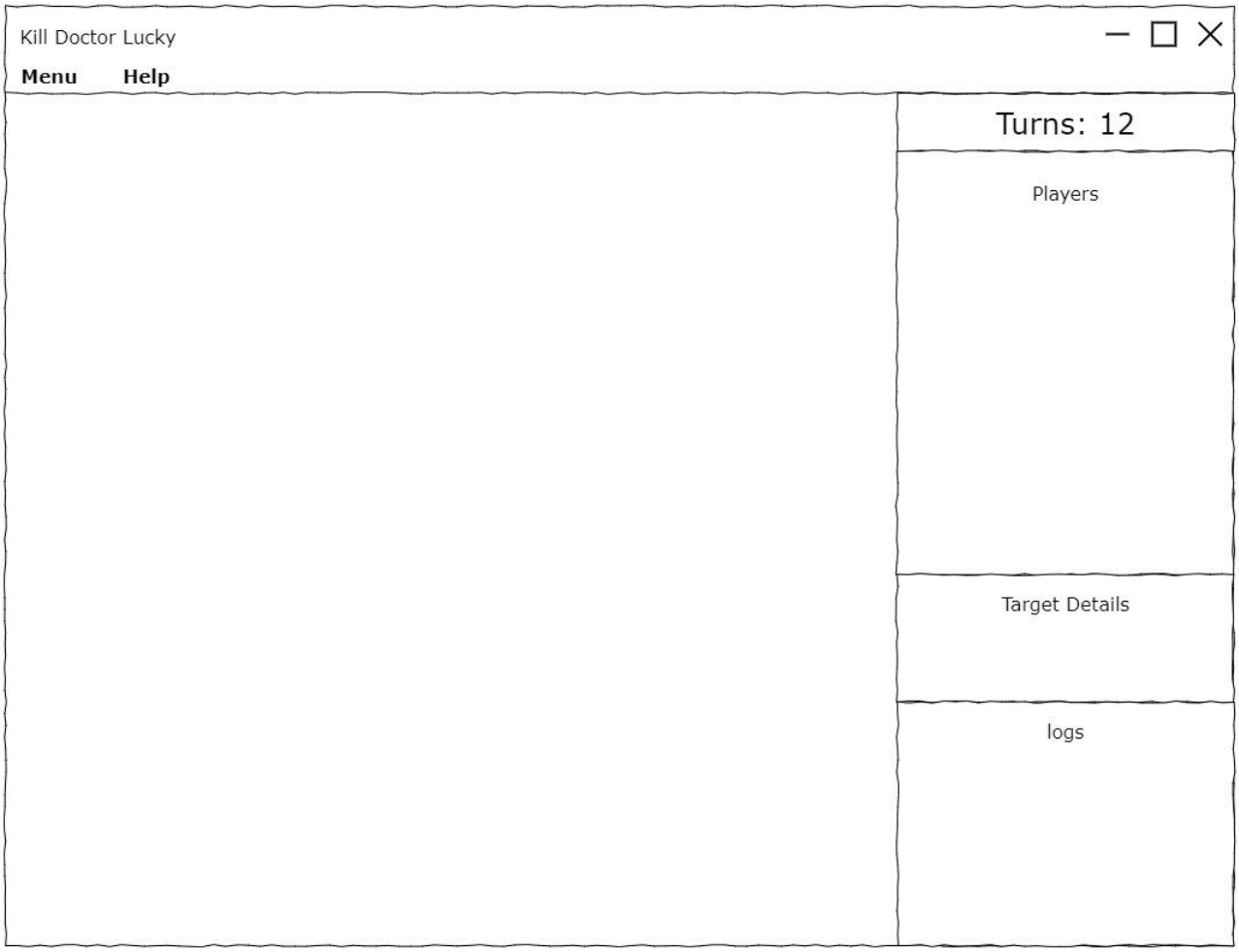
MouseClickListener

- map: Map<Integer, Consumer<String>>

+ setMap(map: Map<Integer, Consumer<String>
+ mouseClicked(e: MouseEvent)

The model of world is split into data and functions/methods. In this way, the players, spaces, weapons, target, pet are abstracted into data, and the data of this game can be abstracted into a **Context**, decoupled from the **World** model. All the functions/methods of the world can be static since the model has no data inside itself, then all functions/methods just take a context as the input and then generate the corresponding result or effect. With this kind of design, it gives me a chance to focus on the business orchestration. Since the methods and functions of world are all static, and the data of the game is a context instance, there's no so-called model layer, so there's no need to consider the access control and isolation.

UI Design



Test Plan for Project

BaseSpace class:

Success cases:

1. Test construct BaseSpace with valid coordinates and valid order, expect success.
2. Test get name function of BaseSpace, expect returning the correct name.
3. Test get start coordinates function of BaseSpace, expect returning the correct start coordinates.
4. Test get end coordinates function of BaseSpace, expect returning the correct end coordinates.
5. Test get order function of BaseSpace, expect returning the correct order index.

Exception cases:

1. Test construct BaseSpace with invalid Coordinates and valid order, expect IllegalArgumentException.
2. Test construct BaseSpace with valid coordinates and invalid order, expect IllegalArgumentException.

Space class:

Success cases:

1. Test get name function of Space, expect returning the correct name.
2. Test get order function of Space, expect returning the correct value.

3. Test get neighbors function of Space, expect returning the correct neighbors.
4. Test get weapons function of Space, expect returning the correct weapons.
5. Test get occupiers function of Space, expect returning the correct occupiers.
6. Test is exposed function of Space, expect returning true when it's exposed.
7. Test is exposed function of Space, expect returning false when it's not exposed.

BaseWeapon class:

Success cases:

1. Test construct BaseWeapon with valid space index and valid damage, expect success.
2. Test get space index function of BaseWeapon, expect returning the correct space index.
3. Test get damage function of BaseWeapon, expect returning the correct damage value.
4. Test get name function of BaseWeapon, expect returning the correct name.
5. Test set holder function of BaseWeapon, expect setting success.
6. Test get holder function of BaseWeapon, expect returning the correct value.

Exception cases:

1. Test construct BaseWeapon with invalid space index and valid damage, expect IllegalArgumentException.
2. Test construct BaseWeapon with valid space index and invalid damage, expect IllegalArgumentException.

Target class:

Success cases:

1. Test construct Target with valid health, expect success.
2. Test get health function of Target, expect returning the correct health value.
3. Test decrease health function of Target, expect returning the correct health value after decrease.
4. Test decrease health function with damage more than current health, expect health to be 0 after decrease.
5. Test get name function of Target, expect returning the correct name.
6. Test get position function of Target, expect returning the correct space index.
7. Test set position function of Target, expect setting success.

Exception cases:

3. Test construct Target with invalid health, expect IllegalArgumentException.

BasePlayer class:

Success cases:

1. Test construct BasePlayer with valid order value, expect success.

Exception cases:

1. Test construct BasePlayer with invalid order value, expect IllegalArgumentException.

Player class:

Success cases:

1. Test get order function of Player, expect returning the correct order.
2. Test get name function of Player, expect returning the correct name.
3. Test set space index function of Player, expect success.
4. Test get space index function of Player, expect returning the correct value.
5. Test get weapon limit function of Player, expect returning the correct limit.
6. Test get type function of Player, expect returning the correct type.

7. Test add weapon function of Player with player holding weapons less than the limit amount, expect success(return true).
8. Test add weapon function of Player with player holding weapons the same as the limit amount, expect fail(return false).
9. Test get weapons function of Player, expect returning the correct value.

ContextBuilder class:

Success cases:

1. Test construct Context with valid world specification file, expect success.

Exception cases:

1. Test construct Context with world specification file, there are invalid row, valid column, valid spaces in the file, expect IllegalArgumentException.
2. Test construct Context with world specification file, there are valid row, invalid column, valid spaces in the file, expect IllegalArgumentException
3. Test construct Context with world specification file, there are valid row, valid column and invalid spaces in the file, expect IllegalArgumentException. The space in the space list contains start coordinates which is more than the row or column.
4. Test construct Context with world specification file, there are valid row, valid column and invalid spaces in the file, expect IllegalArgumentException. The space in the space list contains end coordinates which is more than the row or column.
5. Test construct Context with world specification file, there are valid row, valid column and invalid spaces in the file, expect IllegalArgumentException. The space in the space list overlaps with another one in the list.

World class:

Success cases:

1. Test get all spaces function of World, expect returning the correct space name list.
2. Test get space's neighbors function with valid space name of World, expect returning the correct neighbors' names.
3. Test get space's neighbors function with invalid space name of World, expect returning an empty list.
4. Test get space's neighbors function with valid space index of World, expect returning the correct neighbors' names.
5. Test get space's neighbors function with invalid space index of World, expect returning an empty list.
6. Test get space function with valid space name of World, expect returning the correct weapons, neighbors and players.
7. Test get space function with invalid space name of World, expect returning null.
8. Test get space function with valid space index of World, expect returning the correct weapons, neighbors and players.
9. Test get space function with invalid space index of World, expect returning null.
10. Test get target position function of World, expect returning the correct space info.
11. Test move target function of World, expect returning the correct space info after moving.
12. Test move target function of World, make moves at the last space, expect returning the 0^{th} space.
13. Test graphical image rendering of World, expect the correct image.
14. Test get all spaces function of World, expect returning the correct space name list.
15. Test get all players function, expect returning the correct player list.

16. Test get player function of World with valid index, expect returning the correct player.
17. Test get player function of World with invalid index, expect returning null.
18. Test add player function of World with no repeated name, expect success.
19. Test add player function of World with repeated name, expect adding failure.
20. Test move player function of World with valid space, expect success.
21. Test player pick up function of World with player carrying less than limit amount weapons, expect success.
22. Test player pick up function of World with player carrying full amount weapons, expect picking up failure.

Exception cases:

1. Test move player function of World with null space, expect IllegalArgumentException.
2. Test player pick up function of World with null weapon, expect IllegalArgumentException.

AttackAction class:

Success cases:

1. Test player attack target with a weapon with no other player and no pet in the space, and the attack is not seen, expect success and the weapon becomes evidence and removed from the play.
2. Test player attack target with no weapon with no other player and no pet in the space, and the attack is not seen, expect success.
3. Test player attack target with a weapon with no other player and no pet in the space, and the attack is seen, expect attack fail and the weapon becomes evidence and removed from the play.
4. Test player attack target with no weapon with no other player and no pet in the space, and the attack is seen, expect attack fail.
5. Test player attack target with a weapon with no other player in the space, but the pet is in the space, expect success and the weapon becomes evidence and removed from the play.
6. Test player attack target with no weapon with no other player in the space, but the pet is in the space, expect success.
7. Test player attack target with other players in the space but no pet in the space, expect attack fail and the weapon becomes evidence and removed from the play.
8. Test player attack target with no weapon with other players in the space but no pet in the space, expect attack fail.
9. Test player attack target with a weapon with other players and the pet in the space, expect attack fail and the weapon becomes evidence and removed from the play.
10. Test player attack target with no weapon with other players and the pet in the space, expect attack fail.

Exception cases:

1. Test player attack when the target is not in the same space, expect IllegalArgumentException.

LookAroundAction class:

Success cases:

1. Test player look around and there is a pet in the neighbor spaces, expect the space that has the pet returning empty players, weapons list, but other neighbors return the corresponding result.
2. Test player look around and there is no pet in the neighbor spaces, expect all neighbors returning the corresponding result.

MovePetAction class:

Success cases:

1. Test player move the pet to a valid space, expect success.

Exception cases:

1. Test player move the pet to an invalid space, expect IllegalArgumentException.
2. Test player move the pet to the current space, expect IllegalArgumentException.

MovePlayerAction class:

Success cases:

1. Test player move to a valid neighbor space, expect success.

Exception cases:

1. Test player move to an invalid neighbor space, expect IllegalArgumentException.
2. Test player move to a space that is not a neighbor of current space, expect IllegalArgumentException.

PickUpWeaponAction class:

Success cases:

1. Test player pick up a valid weapon, and the weapon doesn't reach limit, expect success.

Exception cases:

1. Test player pick up a valid weapon and the weapon reaches limit, expect BusinessException.
2. Test player pick up a null weapon, expect IllegalArgumentException.

PetTraverseAction class:

Success cases:

1. Test the pet visiting all the space, expect visiting all the spaces.

WorldConsoleController.class:

Success cases:

1. Test construct WorldConsoleController with valid in and out object, expect success.
2. Test getting input from input stream to create human-controlled player and invoke a real model, expect creating the correct player success.
3. Test getting input form input stream to create computer-controller player and invoke a real model, expect creating the correct player success.
4. Test getting input from input stream to create human-controlled player and invoke a mock model, expect getting mocked result.
5. Test getting input from input stream to create computer-controlled player and invoke a mock model, expect getting mocked result.
6. Test getting input form input stream to create human-controlled player with repeated name and invoke a real model, expect creating failure.
7. Test getting input form input stream to create computer-controlled player with repeated name and invoke a real model, expect creating failure.
8. Test getting input form input stream to create human-controlled player with repeated name and invoke a mock model, expect mocked failure.
9. Test getting input form input stream to create computer-controlled player with repeated name and invoke a mock model, expect mocked failure.
10. Test getting input from input stream to display all spaces and invoke a real model, expecting printing the correct spaces.

11. Test getting input from input stream to display all spaces and invoke a mock model, expect printing the mocked result.
12. Test getting input from input stream to display a space info and invoke a real model, expect printing the correct result.
13. Test getting input from input stream to display a space info and invoke a mock model, expect printing the mocked result.
14. Test getting input from input stream to set turn limit, expect setting success.
15. Test getting input from input stream to display player detail and invoke a real model, expect printing the correct result.
16. Test getting input from input stream to display player detail and invoke a mock model, expect printing the mocked result.
17. Test getting input from input stream to look around the player's space and neighbors and invoke a real model, expect printing the correct result.
18. Test getting input from input stream to look around the player's space and neighbors and invoke a mock model, expect printing the mocked result.
19. Test getting input from input stream to move the player and invoke a real model, expect printing the correct result.
20. Test getting input from input stream to move the player and invoke a mock model, expect printing the mocked result.
21. Test getting input from input stream to make the player pick up a weapon and invoke a real model, expect printing the correct result.
22. Test getting input from input stream to make the player pick up a weapon and invoke a mock model, expect printing the mocked result.
23. Test printing input from input stream to generate the image and invoke a real model, expect printing the correct result.
24. Test getting input from input stream to generate the image and invoke a mock model, expect printing the mocked result.
25. Test getting input from input stream to move the pet and invoke a real model, expect printing the correct result.
26. Test getting input from input stream to move the pet and invoke a mock model, expect printing the mocked result.
27. Test getting input from input stream to make player attack the target and invoke a real model, expect printing the correct result.
28. Test getting input from input stream to make player attack the target and invoke a mock model, expect printing the mocked result.
29. Test playing the game with a real model and ending with reaching the maximum turn, the target is still alive, expect the target's health is bigger than 0.
30. Test playing the game with a mock model and ending with reaching the maximum turn, the target is still alive, expect the target's health is bigger than 0.
31. Test playing the game with a real model and ending with the target is dead, and the human-controlled player is winner, expect target's health is 0 and the human-controlled player is winner.
32. Test playing the game with a mock model and ending with the target is dead, and the human-controlled player is winner, expect target's health is 0 and the human-controlled player is winner.
33. Test playing the game with a real model and ending with the target is dead, and the computer-controlled

player is winner, expect target's health is 0 and the computer-controlled player is winner.

34. Test playing the game with a mock model and ending with the target is dead, and the computer-controlled player is winner, expect target's health is 0 and the computer-controlled player is winner.

Exception cases:

1. Test construct WorldConsoleController with invalid in object and valid out object, expect throwing IllegalArgumentException.
2. Test construct WorldConsoleController with valid in object and invalid out object, expect throwing IllegalArgumentException.

WorldGuiController.class:

Success cases:

1. Add human-controlled player with valid name and space index, expect success.
2. Add computer-controlled player with valid name and space index, expect success.
3. Test start game, expect the game start to run successfully.
4. Restart game, expect the game restart successfully.
5. Quit game, expect the game end successfully.