controller

world

<<Interface>>
WorldController

+ playGame(world: World)

WorldConsoleController

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

+ WorldConsoleController(in: Readable, out: Appendable, turn: Integer)
+ playGame(world: World)
- createPlayer(world: World, order: int)
- start(world: World)
- displayPlayerDetail(player: Player, world: World)
- pickUp(player: Player, cur: Space, weapon: String, world: World)
- movePlayer(player: Player, cur: Space, neighbor: String, world: World)
- displayPaylerSpaceDetail(player: Player, world: World)

<<Interface>>
World

+ getAllSpaces(): List<String>
+ getNeighbors(name: String): List<String>
+ getNeighbors(index: int): List<String>
+ getSpace(name: String): Space
+ getSpace(index: int): Space
+ moveTarget(): Space
+ getAllPlayers(): List<Player>
+ getPlayer(index: int): Player
+ addPlayer(player: Player): Boolean
+ movePlayer(player: Player, space: Space)
+ pickUp(player: Player, weapon: Weapon)
+ showGraphicalImage(directory: String)

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)

WorldImpl

- matrix: int[][]
- name: String
- target: Target
- spaces: List<Space>
- players: List<Player>

+ WorldImpl(in: Readable)
+ WorldImpl(row: int, col: int, name: String, target: Target, baseSpaces: List<BaseSpace>, baseWeapons: List<BaseWeapon>
+ getAllSpaces(): List<String>
+ getNeighbors(name: String): List<String>
+ getNeighbors(index: int): List<String>
+ getSpace(name: String): Space
+ getSpace(index: int): Space
+ moveTarget(): Space
+ getAllPlayers(): List<Player>
+ getPlayer(index: int): Player
+ addPlayer(player: Player): Boolean
+ movePlayer(player: Player, space: Space)
+ pickUp(player: Player, weapon: Weapon)
+ showGraphicalImage(directory: String)
- initMatrixAndValidateSpaces(m: int, n: int, baseSpaces: List<BaseSpace>)
- initNeighborMapAndSpaces(m: int, n: int, baseSpaces: List<BaseSpace>, baseWeapons: List<BaseWeapon>)
- getNeighborsFormMatrix(baseSpace: baseSpace, baseSpaces: List<BaseSpace>, m: int, n: int): List<BaseSpace>
- getNeighborNamesFromSpace(space: Space): List<String>

<<Interface>>
Player

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

Space

- neighbors: List<BaseSpace>
- weapons: List<Weapon>
- occupiers: List<Player>

+ Space(base: BaseSpace, neighbors: List<BaseSpace>, weapons: List<Weapon>)
+ getNeighbors(): List<BaseSpace>
+ setNeighbors(neighbors: List<BaseSpace>)
+ getWeapons(): List<Weapon>
+ setWeapons(weapons: List<Weapon>)
+ getOccupiers(): List<Player>
+ setOccupiers(List<Player>)

<<enumeration>>
PlayerType

HUMAN
COMPUTER

BasePlayer

- order: int
- name: String
- spaceIndex: int
- type: PlayerType
- weaponLimit: Integer
- weapons: List<Weapon>

+ 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>

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

BaseWeapon

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

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

Weapon

+ Weapon(base: BaseWeapon)

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 set neighbors function of Space, expect setting success.
2. Test get neighbors function of Space, expect returning the correct neighbors.
3. Test set weapons function of Space, expect setting success.
4. Test get weapons function of Space, expect returning the correct weapons.
5. Test set occupiers function of Space, expect setting success.
6. Test get occupiers function of Space, expect returning the correct occupiers.

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.

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.

Weapon class:

Success cases:

1. Test construct Weapon with valid BaseWeapon instance, expect success.

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.
2. Test get order function of BasePlayer, expect returning the correct order.
3. Test get name function of BasePlayer, expect returning the correct name.
4. Test set space index function of BasePlayer, expect success.
5. Test get space index function of BasePlayer, expect returning the correct value.
6. Test get weapon limit function of BasePlayer, expect returning the correct limit.
7. Test get type function of BasePlayer, expect returning the correct type.
8. Test add weapon function of BasePlayer within player holding weapons less than the limit amount, expect success (return true).
9. Test add weapon function of BasePlayer with player holding weapons the same as the limit amount, expect fail (return false).
10. Test get weapons function of BasePlayer, expect returning the correct value.

Exception cases:

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

WorldImpl class:

Success cases:

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

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

Exception cases:

1. Test construct WorldImpl with world specification file, there are invalid row, valid column, valid spaces in the file, expect IllegalArgumentException.
2. Test construct WorldImpl with world specification file, there are valid row, invalid column, valid spaces in the file, expect IllegalArgumentException
3. Test construct WorldImpl 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 WorldImpl 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 WorldImpl 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.
6. Test move player function of World with null space, expect IllegalArgumentException.
7. Test player pick up function of World with null weapon, expect IllegalArgumentException.

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 steam to create human-controlled player with repeated name and invoke a real model, expect creating failure.
7. Test getting input form input steam to create computer-controlled player with repeated name and invoke a real model, expect creating failure.
8. Test getting input form input steam to create human-controlled player with repeated name and invoke a mock model, expect mocked failure.
9. Test getting input form input steam 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 display the space info that the player occupies and invoke a real model, expect printing the correct result.
18. Test getting input from input stream to display the space info that the player occupies 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.

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.