**WorldConsoleControllerTest.java**

After every turn, controller will show the current player info (health, current position, name, etc.). A player will have the option to see the surrounding spaces, weapons in those spaces, and the player information (current position and weapons they have)
// Input where the q comes
// Input where non-integer garbage comes instead of the expected value
// Input where the move is integers (space index), but outside the bounds of the board
// Input where the move is integers(space index), but invalid because the player has reached the maximum turns.
// Multiple invalid moves.
// Input including valid moves interspersed with invalid moves, game is played to completion
// What happens when the input ends "abruptly" -- no more input, but not quit, and game not over
// Test if the health returned is correct
// Test if the weapon shown in a particular space is valid
// Test if the available spaces to move is valid
// Test if player moves if not allowed
// Test if player moves if not allowed
// Test if game ends with a winner
// Test if the game ends with the correct winner
// Test if the computer-player moves in the correct spaces.
// Test if the weapons present in a particular space
// Test if the correct weapons are present in a particular space
// Test if the player has reached the max limit of having weapons
// Test valid moves
// Test invalid moves
// Test if the player dies if the health reduces below zero
// Test valid turns
// Test invalid turns

**WorldModel.java**

//Test if returns valid player info
//Test if returns valid space info
//Test if return valid weapon info
//Test if returns all the players
//Test if returns all the spaces

```
//Test if returns weapons info
//Test if returns valid world description
//Test if space has valid weapons
//Test if space has valid players
//test if space has valid neighbor

public void testIfFileParsing()
/**
 * Testing illegal world description.
 */
@Test(expected = IllegalArgumentException.class)
public void testInvalidWorldDescription()

/**
 * Testing file parsing.
 */
@Test
public void testParsing()

/**
 * Testing world details parsing errors.
 */
@Test(expected = IllegalArgumentException.class)
public void testParsingErrors()

/**
 * Testing invalid room and weapons while parsing.
 */
@Test(expected = IllegalArgumentException.class)
public void testInvalidRoomAndWeapon()

/**
 * Testing invalid world description.
 */
@Test
public void testInValidWorldDescription2()
/**
 * Testing invalid room description.
 */
@Test
```

```
public void testInValidRoomDescription()

/**
 * Testing invalid weapon description.
 */
@Test
public void testInValidWeaponDescription()

/**
 * Testing room having zero neighbors.
 */
@Test(expected = NullPointerException.class)
public void zeroNeighborTest()

/**
 * Testing room having one neighbor.
 */
@Test
public void oneNeighborTest()

/**
 * Testing description of room having one neighbor.
 */
@Test
public void oneNeighborDescTest()

/**
 * Testing multiple neighbors.
 */
@Test
public void multipleNeighborTest()

/**
 * testing one item description.
 */
@Test
public void oneItemDescTest()

/**
 * Testing no item description.
```

```
 */
@Test
public void noItemDescTest()

/**
 * Target Character Starting Position Test.
 */
@Test
public void targetCharacterStartingPositionTest()

/**
 * target Character Move Test.
 *
 */
@Test
public void targetCharacterMoveTest() throws GameOverException

/**
 * target Character Move Zero To One Test.
 *
 */
@Test
public void targetCharacterMoveZeroToOneTest()

/**
 * target Character Any Move Test.
 *
 */
@Test
public void targetCharacterAnyMoveTest() throws GameOverException

/**
 * target Character Move End Start.
 *
 */
@Test(expected = GameOverException.class)
public void targetCharacterMoveEndStart() throws GameOverException

/**
 * verifying correctness of the world image.
```

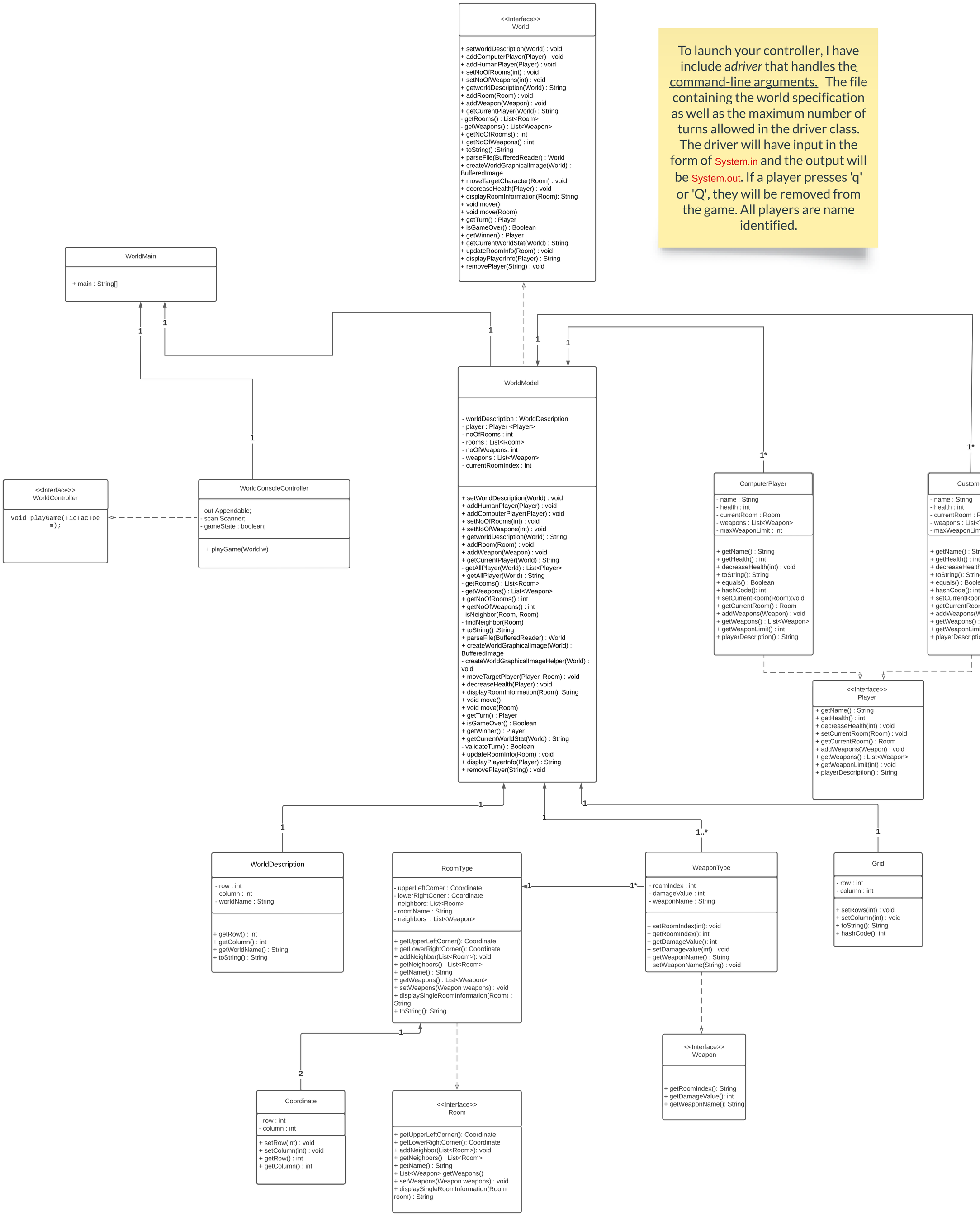
```
 *
 */
@Test
public void verifyImageWorld() throws IOException
```

**FailingAppendable.java**

```
 @Override
 public Appendable append(CharSequence csq) throws IOException {
  throw new IOException("Fail!");
 }

 @Override
 public Appendable append(CharSequence csq, int start, int end) throws IOException {
  throw new IOException("Fail!");
 }

 @Override
 public Appendable append(char c) throws IOException {
  throw new IOException("Fail!");
 }
```


To launch your controller, I have include a driver that handles the command-line arguments. The file containing the world specification as well as the maximum number of turns allowed in the driver class. The driver will have input in the form of System.in and the output will be System.out. If a player presses 'q' or 'Q', they will be removed from the game. All players are name identified.
<<Interface>>
World
+ setWorldDescription(World) : void
+ addComputerPlayer(Player) : void
+ addHumanPlayer(Player) : void
+ setNoOfRooms(int) : void
+ setNoOfWeapons(int) : void
+ getworldDescription(World) : String
+ addRoom(Room) : void
+ addWeapon(Weapon) : void
+ getCurrentPlayer(World) : String
- getRooms() : List<Room>
- getWeapons() : List<Weapon>
+ getNoOfRooms() : int
+ getNoOfWeapons() : int
+ toString() :String
+ parseFile(BufferedReader) : World
+ createWorldGraphicalImage(World) : BufferedImage
+ moveTargetCharacter(Room) : void
+ decreaseHealth(Player) : void
+ displayRoomInformation(Room): String
+ void move()
+ void move(Room)
+ getTurn() : Player
+ isGameOver() : Boolean
+ getWinner() : Player
+ getCurrentWorldStat(World) : String
+ updateRoomInfo(Room) : void
+ displayPlayerInfo(Player) : String
+ removePlayer(String) : void
WorldMain
+ main : String[]
WorldModel
- worldDescription : WorldDescription
- player : Player <Player>
- noOfRooms : int
- rooms : List<Room>
- noOfWeapons: int
- weapons : List<Weapon>
- currentRoomIndex : int
+ setWorldDescription(World) : void
+ addHumanPlayer(Player) : void
+ addComputerPlayer(Player) : void
+ setNoOfRooms(int) : void
+ setNoOfWeapons(int) : void
+ getworldDescription(World) : String
+ addRoom(Room) : void
+ addWeapon(Weapon) : void
+ getCurrentPlayer(World) : String
- getAllPlayer(World) : List<Player>
+ getAllPlayer(World) : String
- getRooms() : List<Room>
- getWeapons() : List<Weapon>
+ getNoOfRooms() : int
+ getNoOfWeapons() : int
- isNeighbor(Room, Room)
- findNeighbor(Room)
+ toString() :String
+ parseFile(BufferedReader) : World
+ createWorldGraphicalImage(World) : BufferedImage
- createWorldGraphicalImageHelper(World) : void
+ moveTargetPlayer(Player, Room) : void
+ decreaseHealth(Player) : void
+ displayRoomInformation(Room): String
+ void move()
+ void move(Room)
+ getTurn() : Player
+ isGameOver() : Boolean
+ getWinner() : Player
+ getCurrentWorldStat(World) : String
- validateTurn() : Boolean
+ updateRoomInfo(Room) : void
+ displayPlayerInfo(Player) : String
+ removePlayer(String) : void
<<Interface>>
WorldController
void playGame(TicTacToe m);
WorldConsoleController
- out Appendable;
- scan Scanner;
- gameState : boolean;
+ playGame(World w)
ComputerPlayer
- name : String
- health : int
- currentRoom : Room
- weapons : List<Weapon>
- maxWeaponLimit : int
+ getName() : String
+ getHealth() : int
+ decreaseHealth(int) : void
+ toString(): String
+ equals() : Boolean
+ hashCode(): int
+ setCurrentRoom(Room):void
+ getCurrentRoom() : Room
+ addWeapons(Weapon) : void
+ getWeapons() : List<Weapon>
+ getWeaponLimit() : int
+ playerDescription() : String
CustomPlayer
- name : String
- health : int
- currentRoom : Room
- weapons : List<Weapon>
- maxWeaponLimit : int
+ getName() : String
+ getHealth() : int
+ decreaseHealth(int) : void
+ toString(): String
+ equals() : Boolean
+ hashCode(): int
+ setCurrentRoom(Room):void
+ getCurrentRoom() : Room
+ addWeapons(Weapon) : void
+ getWeapons() : List<Weapon>
+ getWeaponLimit() : int
+ playerDescription() : String
<<Interface>>
Player
+ getName() : String
+ getHealth() : int
+ decreaseHealth(int) : void
+ setCurrentRoom(Room) : void
+ getCurrentRoom() : Room
+ addWeapons(Weapon) : void
+ getWeapons() : List<Weapon>
+ getWeaponLimit(int) : void
+ playerDescription() : String
WorldDescription
- row : int
- column : int
- worldName : String
+ getRow() : int
+ getColumn() : int
+ getWorldName() : String
+ toString() : String
RoomType
- upperLeftCorner : Coordinate
- lowerRightConer : Coordinate
- neighbors: List<Room>
- roomName : String
- neighbors : List<Weapon>
+ getUpperLeftCorner(): Coordinate
+ getLowerRightCorner(): Coordinate
+ addNeighbor(List<Room>): void
+ getNeighbors() : List<Room>
+ getName() : String
+ getWeapons() : List<Weapon>
+ setWeapons(Weapon weapons) : void
+ displaySingleRoomInformation(Room) : String
+ toString(): String
WeaponType
- roomIndex : int
- damageValue : int
- weaponName : String
+ setRoomIndex(int): void
+ getRoomIndex(): int
+ getDamageValue(): int
+ setDamagevalue(int) : void
+ getWeaponName() : String
+ setWeaponName(String) : void
Grid
- row : int
- column : int
+ setRows(int) : void
+ setColumn(int) : void
+ toString(): String
+ hashCode(): int
Coordinate
- row : int
- column : int
+ setRow(int) : void
+ setColumn(int) : void
+ getRow() : int
+ getColumn() : int
<<Interface>>
Room
+ getUpperLeftCorner(): Coordinate
+ getLowerRightCorner(): Coordinate
+ addNeighbor(List<Room>): void
+ getNeighbors() : List<Room>
+ getName() : String
+ List<Weapon> getWeapons()
+ setWeapons(Weapon weapons) : void
+ displaySingleRoomInformation(Room room) : String
<<Interface>>
Weapon
+ getRoomIndex(): String
+ getDamageValue(): int
+ getWeaponName(): String
1
1*
1..*
2