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

Additional Tests (MileStone 3)-

- verifying starting position of the players
- verifying starting position of the pet
- verifying starting position of the target character
- verifying space description
- verify move to neighbor
- verify not moving
- verify pick up item that isn't there
- verify pickup beyond maximum
- verify look around with no items
- verify looking around with items
- verify looking around - target character
- verifying looking around - players
- verify player description with no items
- verify player description with items
- taking turns
- verify one action
- verifying if pet moves according to the expected algorithm (DFS)
- verify if target player moves expectedly
- verify player order
- verify end of game when maximum turns reached
- verify display info command
- verify getting graphical representation
- verifying add human player command
- verifying add computer player command
- verify move command
- verify pick-up command
- verify look around command
- verify player description command
- verify if health decreases when weapon used
- verify if an attack is *seen* by another player (human or computer), it is automatically stopped and no damage is done
- verify unseen successful attacks
- verify if a player successfully kills the target character in which case they win the game.
- Verify if the maximum number of turns is reached in which case the target character escapes and runs away to live another day and nobody wins.
- Verify poking attack
- Verify poking attack is done by player with no weapons
- Verify if turn includes moving the pet.
- Verify if the space that is occupied by the pet *cannot be seen* by its neighbors making it virtually invisible to the user.

- Verify if the pet enters the game in the same space as the target character
- Verify if the description of the space includes the pet
- Verify if the target character is in the description

```
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!");
  }
```