Dungeon Crafter

Unit Testing and Inspection Report

Prepared by Group 1:
Michal Bochnak
Sean Martinelli
Alex Viznytsya
Artur Wojcik

University of Illinois Chicago
Spring 2018

Table of Contents

I Overview

1 Project Description

Dungeon Crafter is an online role playing mobile game for the Android operating system. The object of the game is complete adventures to level up your character. To begin an adventure, players enter the find group lobby in order to pair-up with three additional players to form a team. Each adventure consists of one hundred levels to be completed by a team. Each level contains multiple enemy characters that the team must defeat in order to move on. Players each take a turns moving their character and attacking an enemy if in range. The level of difficulty of each adventure increases as your character gains experiences and progresses.

2 Document Description

This document outlines the results of the inspection and unit testing performed on this application. The inspection of the methods in section II of this document was performed by three of the four team members that did not write that particular piece of code. The team members agreed upon a check list to confirm the validity of the code. Each method that was tested in the unit testing section was tested by a team member that did not write that particular piece of code. The team members identified the equivalence classes for the input of each method, and developed tests for each of the identified equivalence classes.

3 Description of Code Tested and Inspected

3a public int extractRace(JSONObject obj)

This method extracts the characters race from the JSONObject sent from the server. The race is returned as an integer representation of the race.

3b private void processResponse(JSONObject response)

This method extracts all of the necessary information sent in the form of a JSONObject from the server.

3c public class CharacterFactory

This class creates and returns Character objects that are used during gameplay. The Character objects returned can be either player characters or enemy characters.

3d synchronized private void updateGrid()

This method updates the view of the game grid for the user to show where characters and enemies are located.

3e private boolean validateNewPosition(Position newPosition)

This method checks the specified Position to see if it is valid. It is valid if it is within the bounds of the grid and no other player is already occupying the position. If the position is valid, true is returned. If it is not, false is returned.

3f gamePlay.php

This server side script is responsible for processing the JSONObjects sent by the clients during gameplay. Upon the processing of an object, the database is updated to reflect the new state of the game. The new game state is then pushed to the other clients participating in the gameplay.

3g private int rollDice()

This method rolls the dice by generating a random number between 1 and 6. The view corresponding to the number of moves remaining is updated to show the generated number and player is marked as having rolled the dice.

3h private int determineDamageToDeal(Character character)

This method calculates and returns the amount of damage that is dealt by a character when attacking. This number is calculated based on the strength of the character passed to the method.

II Inspection

4 public int extractRace(JSONObject obj)

Code to be inspected

```
public int extractRace(JSONObject obj) {
  int race = -1;
  try {
      race = obj.getInt(getString(R.string.race));
  }
  catch (Exception e) {
      e.printStackTrace();
  }
  return race;
}
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation

- ✗ Is the code clearly and adequately documented with good commenting style?
- ✗ Are all comments consistent with the code?

Variables

- ✓ Are all variables properly defined with meaningful, consistent, and clear names?
- ✓ Do all assigned variables have proper type consistency or casting?
- ❑ Are there any redundant or unused variables?

Loops and Branches

- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ❑ Are the most common cases tested first in IF - ELSE chains?
- ❑ Does every case statement have a default?
- ❑ Are loop termination conditions obvious and invariably achievable?
- ❑ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming

- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ❑ Are imported data and input arguments tested for validity and completeness?
- ❑ Are all output variables assigned?
- ✓ Are the correct data operated on in each statement?
- ✓ Are timeouts or error traps used for external device accesses?
- ❑ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

5 private void processResponse(JSONObject response)

Code to be inspected

```
private void processResponse(JSONObject response) {

  JSONArray arr1 = null;
  try {
      arr1 = response.getJSONArray("players");
  } catch (Exception e) {};

  // extract players
  JSONArray players = extractPlayers(response);
  setUpPlayers(players);

  // extract enemies
  JSONArray enemies = extractEnemies(response);
  setUpEnemies(enemies);

  runOnUiThread(new Runnable() {
      @Override
      public void run() {
          setPlayerNameLabels();
      }
  });
}
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation

- ✓ Is the code clearly and adequately documented with good commenting style?
- ✓ Are all comments consistent with the code?

Variables

- ✗ Are all variables properly defined with meaningful, consistent, and clear names?
- ✓ Do all assigned variables have proper type consistency or casting?
- ✓ Are there any redundant or unused variables?

Loops and Branches

- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ❑ Are the most common cases tested first in IF - ELSE chains?
- ❑ Does every case statement have a default?
- ❑ Are loop termination conditions obvious and invariably achievable?
- ❑ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming

- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ❑ Are imported data and input arguments tested for validity and completeness?
- ❑ Are all output variables assigned?
- ✓ Are the correct data operated on in each statement?
- ✓ Are timeouts or error traps used for external device accesses?
- ❑ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

6 public class CharacterFactory

Code to be inspected

```
public class CharacterFactory {

  private static final int HUMAN_RACE = 101, DWARF_RACE = 102, ELF_RACE = 103,
          GNOME_RACE = 104, OGRE_RACE = 105, TROLL_RACE = 106;

  public static Character getCharacterInstance(int race) {
      Character character = null;

      switch (race) {
          case HUMAN_RACE:
              character = new Human(16, 100, new Position(0, 0), 0, false);
              break;
```

```
            case DWARF_RACE:
                character = new Dwarf(16, 100, new Position(0, 0), 0, false);
                break;
            case ELF_RACE:
                character = new Elf(10, 100, new Position(0, 0), 0, false);
                break;
            case GNOME_RACE:
                character = new Gnome(10, 100, new Position(0, 0), 0, false);
                break;
            case OGRE_RACE:
                character = new Ogre(10, 100, new Position(0, 0), 0, false, 1);
                break;
            case TROLL_RACE:
                character = new Troll(10, 100, new Position(0, 0), 0, false, 1);
                break;
        }

        return character;
    }
}
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation

- ✗ Is the code clearly and adequately documented with good commenting style?
- ✗ Are all comments consistent with the code?

Variables

- ✓ Are all variables properly defined with meaningful, consistent, and clear names?
- ✓ Do all assigned variables have proper type consistency or casting?
- ❑ Are there any redundant or unused variables?

Loops and Branches

- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ❑ Are the most common cases tested first in IF - ELSE chains?
- ✗ Does every case statement have a default?
- ❑ Are loop termination conditions obvious and invariably achievable?
- ❑ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming

- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ❑ Are imported data and input arguments tested for validity and completeness?
- ❑ Are all output variables assigned?
- ✓ Are the correct data operated on in each statement?
- ✓ Are timeouts or error traps used for external device accesses?
- ❑ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

7 synchronized private void updateGrid()

Code to be inspected

```
synchronized private void updateGrid()
{
  int numChildren = gameGrid.getChildCount();

  // Put players and enemies in one ArrayList
  ArrayList<Character> allCharacters = new ArrayList<Character>();
  allCharacters.addAll(groupMembers);
  allCharacters.addAll(enemies);

  // Loop through all grid spaces
  for(int i = 0 ; i < numChildren ; i++)
  {
     ImageView child = (ImageView) gameGrid.getChildAt(i);

     child.setImageResource(R.drawable.empty_cell_background);

     Position currentIndexPosition = convertGridIndexToPosition(i);

     // Loop through all characters
     for(Character character : allCharacters)
        if (character != null && character.getPosition().equals(currentIndexPosition))
        {
           // enemy image
           if (isEnemySubclass(character))
              child.setImageResource(getEnemyImageResource((Enemy) character));
           else
              child.setImageResource(getPlayerImageResource((Player) character));
        }

     child.invalidate();
  }

  // Update UI elements
  updateHealthLabels();
  updateTurnLabels();
  gameGrid.invalidate();
}
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation

- ✓ Is the code clearly and adequately documented with good commenting style?
- ✓ Are all comments consistent with the code?

Variables

- ✓ Are all variables properly defined with meaningful, consistent, and clear names?

- ✓ Do all assigned variables have proper type consistency or casting?
- ❑ Are there any redundant or unused variables?

Loops and Branches

- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ✗ Are the most common cases tested first in IF - ELSE chains?
- ❑ Does every case statement have a default?
- ✓ Are loop termination conditions obvious and invariably achievable?
- ❑ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming

- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ❑ Are imported data and input arguments tested for validity and completeness?
- ❑ Are all output variables assigned?
- ✓ Are the correct data operated on in each statement?
- ✓ Are timeouts or error traps used for external device accesses?
- ❑ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

8 private boolean validateNewPosition(Position newPosition)

Code to be inspected

```
private boolean validateNewPosition(Position newPosition)
{
  int newRow = newPosition.getRow();
  int newCol = newPosition.getColumn();

  //Make sure column is valid
  if(newCol < 1 || newCol > NUM_COLS)
     return false;

  //Make sure row is valid
  if(newRow < 1 || newRow > NUM_ROWS)
     return false;

  ArrayList<Character> allCharacters = new ArrayList<>();
  allCharacters.addAll(groupMembers);
  allCharacters.addAll(enemies);

  //Make sure another character is not already at this position
  for(Character character : allCharacters)
  {
     Position characterPosition = character.getPosition();
     if(characterPosition.equals(newPosition))
        return false;
  }

  return true;
}
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation

- ✓ Is the code clearly and adequately documented with good commenting style?
- ✓ Are all comments consistent with the code?

Variables

- ✓ Are all variables properly defined with meaningful, consistent, and clear names?
- ✓ Do all assigned variables have proper type consistency or casting?
- ❑ Are there any redundant or unused variables?

Loops and Branches

- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ✓ Are the most common cases tested first in IF - ELSE chains?
- ❑ Does every case statement have a default?
- ✓ Are loop termination conditions obvious and invariably achievable?
- ✓ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming

- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ❑ Are imported data and input arguments tested for validity and completeness?
- ❑ Are all output variables assigned?
- ✓ Are the correct data operated on in each statement?
- ❑ Are timeouts or error traps used for external device accesses?
- ❑ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

9 gamePlay.php

Code to be inspected

```php
<?php

error_reporting( E_ALL );
header( "Content-Type: application/json; charset=UTF-8" );
try {
   $db = new PDO( 'mysql:host=localhost;dbname=DungeonCrafter;charset=utf8',
                  'avizny2', 'CS440@18' );
} catch (Exception $e) {
  die( 'Error : '.$e->getMessage() );
}
```

```
$singupRawData = json_decode( file_get_contents( 'php://input' ) );

$sessionID = $singupRawData->sessionid;
$groupID = ( int )$singupRawData->groupId;
$playerID = ( int )$singupRawData->playerId;

$response = array ("players" => array(), "enemies" => array());

$sql = "SELECT * FROM gameplay WHERE group_id = $groupID";
$getAllPlayers = $db->prepare($sql);
$getAllPlayers->execute();
$allPlayers  = $getAllPlayers->fetchAll( PDO::FETCH_ASSOC );
$playerNameSQL = "SELECT * FROM players WHERE id = :playerId";
$getPlayerName = $db->prepare($playerNameSQL);

foreach ( $allPlayers as $key => $value ) {
   if( ( int )$value['player_id'] < 0 ) {
   array_push( $response['enemies'], array( "playerId" => ( int
)$value['player_id'],
                                            "playerName" => "Enemy",
                                            "x" => ( int )$value['x_coord'],
                                            "y" => ( int )$value['y_coord'],
                                            "health" => ( int
)$value['health'],
                                            "turn" => ( int )$value['turn'],
                                            "race" => ( int )$value['race'] )
);
   } else {
       $getPlayerName->execute( array( ":playerId" => ( int
)$value['player_id'] ) );
       $playerName = $getPlayerName->fetch( PDO::FETCH_ASSOC )['name'];
       array_push($response['players'], array("playerId" =>
(int)$value['player_id'],
                                              "playerName" => $playerName,
                                              "x" => ( int )$value['x_coord'],
                                              "y" => ( int )$value['y_coord'],
                                              "health" => ( int
)$value['health'],
                                              "turn" => ( int )$value['turn'],
                                              "race" => ( int
)$value['race']));
   }
}
print( json_encode( $response ) );
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation

- ✗ Is the code clearly and adequately documented with good commenting style?
- ✗ Are all comments consistent with the code?

Variables

- ✓ Are all variables properly defined with meaningful, consistent, and clear names?

- ❑ Do all assigned variables have proper type consistency or casting?
- ✗ Are there any redundant or unused variables?

Loops and Branches
- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ✓ Are the most common cases tested first in IF - ELSE chains?
- ❑ Does every case statement have a default?
- ✓ Are loop termination conditions obvious and invariably achievable?
- ✓ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming
- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ✗ Are imported data and input arguments tested for validity and completeness?
- ✓ Are all output variables assigned?
- ❑ Are the correct data operated on in each statement?
- ✓ Are timeouts or error traps used for external device accesses?
- ✓ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

10 private int rollDice()

Code to be inspected

```
private int rollDice()
{
  Random rand = new Random();
  int remMoves = rand.nextInt(6) + 1;
  rollTxtView.setText(Integer.toString(remainingMoves));
  hasRolled = true;
  return remMoves;
}
```

Inspection Check List

Check List Legend: ✓ - fulfilled, ✗ - violated, ❑ - not applicable

Documentation
- ✗ Is the code clearly and adequately documented with good commenting style?
- ✗ Are all comments consistent with the code?

Variables
- ✓ Are all variables properly defined with meaningful, consistent, and clear names?
- ✓ Do all assigned variables have proper type consistency or casting?
- ❑ Are there any redundant or unused variables?

Loops and Branches

- ✓ Are all loops, branches, and logic constructs complete and properly nested?
- ✓ Are the most common cases tested first in IF - ELSE chains?
- ❑ Does every case statement have a default?
- ❑ Are loop termination conditions obvious and invariably achievable?
- ❑ Are indexes or subscripts properly initialized, just prior to the loop?
- ❑ Can any statements that are enclosed within loops be placed outside the loops?

Defensive Programming

- ❑ Are indexes, pointers, and subscripts tested against array, record, or file bounds?
- ❑ Are imported data and input arguments tested for validity and completeness?
- ✓ Are all output variables assigned?
- ✓ Are the correct data operated on in each statement?
- ❑ Are timeouts or error traps used for external device accesses?
- ❑ Are files checked for existence before attempting to access them?
- ❑ Are all files and devices are left in the correct state upon program termination?

III Unit Testing

11 private boolean validateNewPosition(Position newPosition)

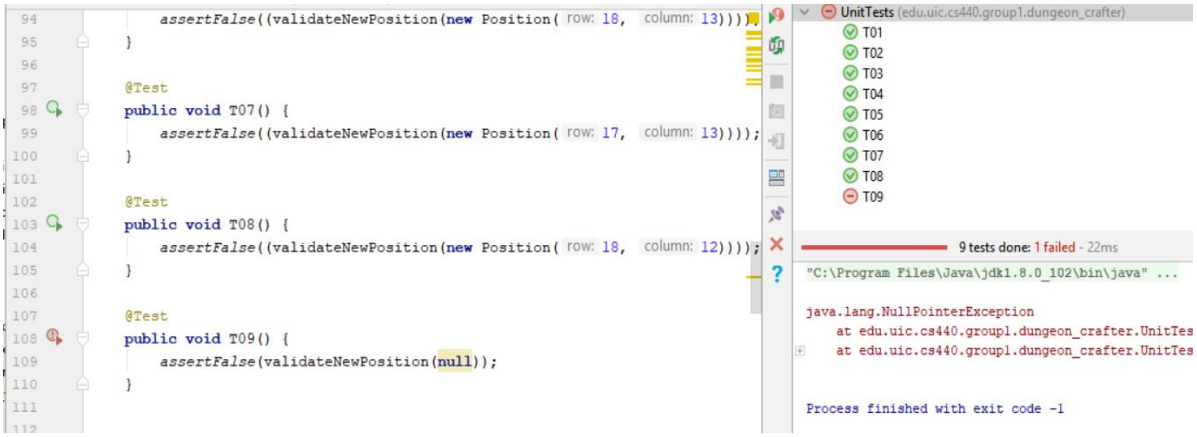
Code to be tested

```
private boolean validateNewPosition(Position newPosition)
{
  int newRow = newPosition.getRow();
  int newCol = newPosition.getColumn();

  //Make sure column is valid
  if(newCol < 1 || newCol > NUM_COLS)
     return false;

  //Make sure row is valid
  if(newRow < 1 || newRow > NUM_ROWS)
     return false;

  ArrayList<Character> allCharacters = new ArrayList<>();
  allCharacters.addAll(groupMembers);
  allCharacters.addAll(enemies);

  //Make sure another character is not already at this position
  for(Character character : allCharacters)
  {
     Position characterPosition = character.getPosition();
     if(characterPosition.equals(newPosition))
        return false;
  }

  return true;
}
```

Test cases

Test case #	X	Y	Position is vacant	Expected result
T01	X < 1	Y < 1	True	False
T02	X < 1	1 ≥ Y ≥ 12	True	False
T03	1 ≤ X ≤ 17	Y < 1	True	False
T04	1 ≤ X ≤ 17	1 ≤ Y ≤ 12	True	True
T05	1 ≤ X ≤ 17	1 ≤ Y ≤ 12	False	False
T06	X > 17	Y > 12	True	False
T07	1 ≤ X ≤ 17	Y > 12	True	False
T08	X > 17	1 ≤ Y ≤ 17	True	False
T09	NULL	NULL	N/A	False

Test results



Revised code

```
private boolean validateNewPosition(Position newPosition)
{
    // Make sure newPosition is not null
    if(newPosition == null)
        return false;

    int newRow = newPosition.getRow();
    int newCol = newPosition.getColumn();

    //Make sure column is valid
    if(newCol < 1 || newCol > NUM_COLS)
        return false;

    //Make sure row is valid
    if(newRow < 1 || newRow > NUM_ROWS)
        return false;

    ArrayList<Character> allCharacters = new ArrayList<>();
    allCharacters.addAll(groupMembers);
    allCharacters.addAll(enemies);

    //Make sure another character is not already at this position
    for(Character character : allCharacters)
    {
        Position characterPosition = character.getPosition();
        if(characterPosition.equals(newPosition))
            return false;
    }

    return true;
}
```

Test results for revised code

```
60      @Test
61      public void T01() {
62          assertFalse((validateNewPosition(new Position( row: 0,  column: 0))));
63      }
64
65      @Test
66      public void T02() {
67          assertFalse((validateNewPosition(new Position( row: 0,  column: 1))));
68      }
69
70      @Test
71      public void T03() {
72          assertFalse((validateNewPosition(new Position( row: 1,  column: 0))));
73      }
74
75      @Test
76      public void T04() {
77          assertTrue((validateNewPosition(new Position( row: 1,  column: 1))));
78      }
79
80      @Test
81      public void T05() {
82          assertFalse((validateNewPosition(new Position( row: 4,  column: 4))));
83      }
84
85      @Test
86      public void T06() {
87          assertFalse((validateNewPosition(new Position( row: 18,  column: 13))));
88      }
89
90      @Test
91      public void T07() {
92          assertFalse((validateNewPosition(new Position( row: 17,  column: 13))));
93      }
94
95      @Test
96      public void T08() {
97          assertFalse((validateNewPosition(new Position( row: 18,  column: 12))));
98      }
```

```
UnitTests (edu.uic.cs440.group1.dungeon_crafter)
    T01
    T02
    T03
    T04
    T05
    T06
    T07
    T08

All 8 tests passed - 0ms
```

12 public int extractRace(JSONObject obj)

Code to be tested

```
public int extractRace(JSONObject obj) {
    int race = -1;
    try {
        race = obj.getInt(getString(R.string.race));
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return race;
}
```

Test cases

Test case #	Json with RaceID	Expected result
T01	No race in ID field	-1
T02	Race field not integer	-1
T03	Null object	-1
T04	Race field integer	Race field given

Test results

```
115     @Test
116     public void T01() {
117         JSONObject json = new JSONObject();
118         try {
119             json.put( name: "type",  value: 7);
120         } catch (Exception e) {}
121
122         assertTrue( condition: extractRace(json) == -1);
123     }
124
125     @Test
126     public void T02() {
127         JSONObject json = new JSONObject();
128         try {
129             json.put( name: "race",  value: "test");
130         } catch (Exception e) {}
131
132         assertTrue( condition: extractRace(json) == -1);
133     }
134
135     @Test
136     public void T03() {
137         assertTrue( condition: extractRace( obj: null) == -1);
138     }
139
140     @Test
141     public void T04() {
142         JSONObject json = new JSONObject();
143         try {
144             json.put( name: "race",  value: 101);
145         } catch (Exception e) {}
146
147         assertTrue( condition: extractRace(json) == 101);
148     }
149
150 }
```

```
UnitTests (edu.uic.cs440.group1.dungeon_crafte
    T01
    T02
    T03
    T04

All 4 tests passed - 31ms
```

13 public class CharacterFactory

Code to be tested

```
public class CharacterFactory {

   private static final int HUMAN_RACE = 101, DWARF_RACE = 102, ELF_RACE = 103,
           GNOME_RACE = 104, OGRE_RACE = 105, TROLL_RACE = 106;

   public static Character getCharacterInstance(int race) {
       Character character = null;

       switch (race) {
           case HUMAN_RACE:
               character = new Human(16, 100, new Position(0, 0), 0, false);
               break;
           case DWARF_RACE:
               character = new Dwarf(16, 100, new Position(0, 0), 0, false);
               break;
           case ELF_RACE:
               character = new Elf(10, 100, new Position(0, 0), 0, false);
               break;
           case GNOME_RACE:
               character = new Gnome(10, 100, new Position(0, 0), 0, false);
               break;
           case OGRE_RACE:
               character = new Ogre(10, 100, new Position(0, 0), 0, false, 1);
               break;
           case TROLL_RACE:
               character = new Troll(10, 100, new Position(0, 0), 0, false, 1);
               break;
       }

       return character;
   }
}
```

Test cases

Test case #	Character creation	Expected result
T01	101	Human Object
T02	102	Dwarf Object
T03	103	Elf Object
T04	104	Gnome Object

T05	105	Ogre Object
T06	106	Troll Object
T07	< 101	Null
T08	> 106	Null

Test results

```
@Test
public void T01() {
    assertTrue(CharacterFactory.getCharacterInstance( race: 101) instanceof Human);
}
@Test
public void T02() {
    assertTrue(CharacterFactory.getCharacterInstance( race: 102) instanceof Dwarf);
}
@Test
public void T03() {
    assertTrue(CharacterFactory.getCharacterInstance( race: 103) instanceof Elf);
}
@Test
public void T04() {
    assertTrue(CharacterFactory.getCharacterInstance( race: 104) instanceof Gnome);
}
@Test
public void T05() {
    assertTrue(CharacterFactory.getCharacterInstance( race: 105) instanceof Ogre);
}
@Test
public void T06() {
    assertTrue(CharacterFactory.getCharacterInstance( race: 106) instanceof Troll);
}
@Test
public void T07() {
    assertTrue( condition: CharacterFactory.getCharacterInstance( race: 107) == null);
}
@Test
public void T08() {
    assertTrue( condition: CharacterFactory.getCharacterInstance( race: 100) == null);
}
}
```

UnitTests (edu.uic.cs440.group1.dungeon_crafter)
- T01
- T02
- T03
- T04
- T05
- T06
- T07
- T08

All 8 tests passed - 0ms

14 private int determineDamageToDeal(Character character)

Code to be tested:

```
private int determineDamageToDeal(Character character)
{
    // Set upper and lower limit based on character's strength
    int upperLimit = character.getStrength();
    int lowerLimit = character.getStrength() - 5;

    //make sure the lower limit is not negative
    if(lowerLimit < 0)
        lowerLimit = 0;
```

```
    //Generate a random number between the upper and lower limits
    Random rand = new Random();
    return rand.nextInt(upperLimit - lowerLimit) + lowerLimit;
}
```

Test cases

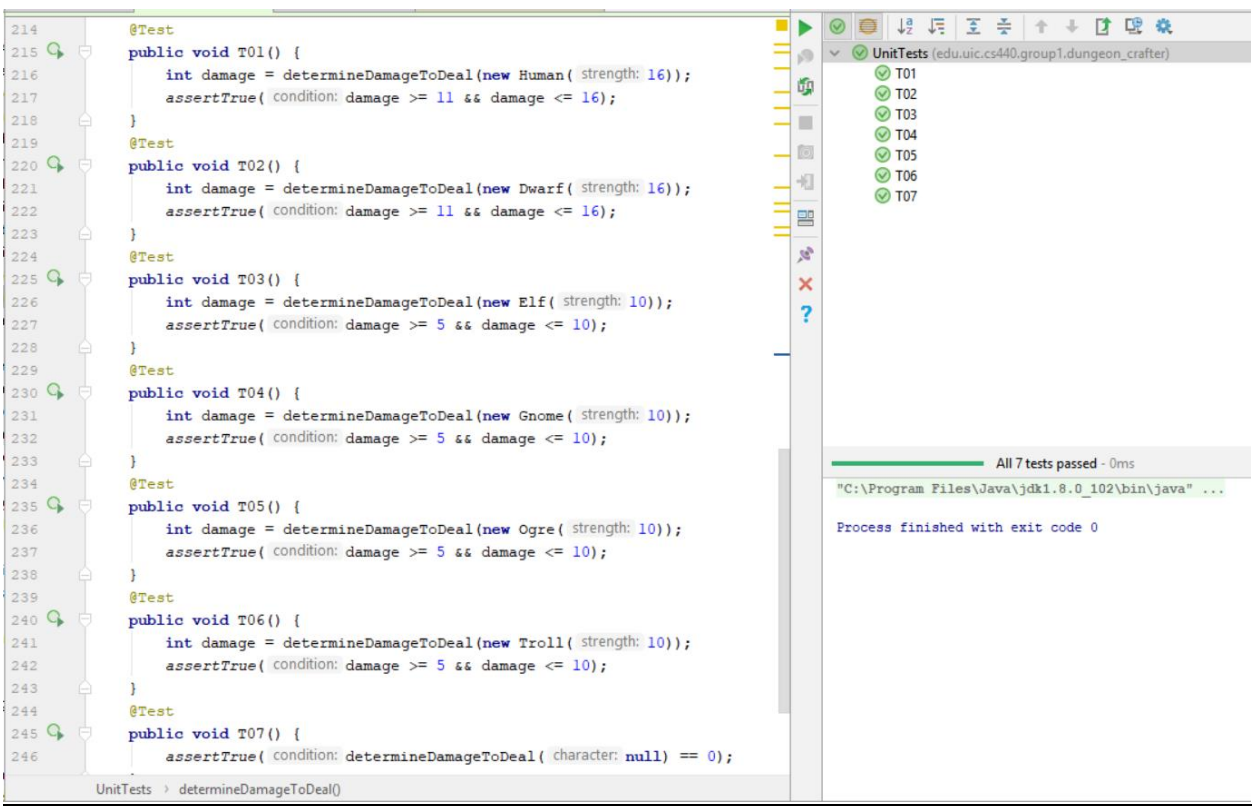
Test case #	Character instance	Expected result
T01	Human Object	11 – 16
T02	Dwarf Object	11 – 16
T03	Elf Object	5 – 10
T04	Gnome Object	5 – 10
T05	Ogre Object	5 – 10
T06	Troll Object	5 – 10
T07	Null	0

Test results

Revised Code

```java
private int determineDamageToDeal(Character character)
{
    // Make sure character is not null
    if(character == null)
        return 0;

    // Set upper and lower limit based on character's strength
    int upperLimit = character.getStrength();
    int lowerLimit = character.getStrength() - 5;

    //make sure the lower limit is not negative
    if(lowerLimit < 0)
        lowerLimit = 0;

    //Generate a random number between the upper and lower limits
    Random rand = new Random();
    return rand.nextInt(upperLimit - lowerLimit) + lowerLimit;
}
```

Test results



15 gamePlay.php

Code to be tested

```php
<?php

error_reporting( E_ALL );
header( "Content-Type: application/json; charset=UTF-8" );
try {
   $db = new PDO( 'mysql:host=localhost;dbname=DungeonCrafter;charset=utf8',
                  'avizny2', 'CS440@18' );
} catch (Exception $e) {
  die( 'Error : '.$e->getMessage() );
}

$singupRawData = json_decode( file_get_contents( 'php://input' ) );

$sessionID = $singupRawData->sessionid;
$groupID = ( int )$singupRawData->groupId;
$playerID = ( int )$singupRawData->playerId;

$response = array ("players" => array(), "enemies" => array());

$sql = "SELECT * FROM gameplay WHERE group_id = $groupID";
$getAllPlayers = $db->prepare($sql);
$getAllPlayers->execute();
$allPlayers  = $getAllPlayers->fetchAll( PDO::FETCH_ASSOC );
$playerNameSQL = "SELECT * FROM players WHERE id = :playerId";
$getPlayerName = $db->prepare($playerNameSQL);

foreach ( $allPlayers as $key => $value ) {
   if( ( int )$value['player_id'] < 0 ) {
   array_push( $response['enemies'], array( "playerId" => ( int
)$value['player_id'],
                                            "playerName" => "Enemy",
                                            "x" => ( int )$value['x_coord'],
                                            "y" => ( int )$value['y_coord'],
                                            "health" => ( int
)$value['health'],
                                            "turn" => ( int )$value['turn'],
                                            "race" => ( int )$value['race'] )
);
   } else {
       $getPlayerName->execute( array( ":playerId" => ( int
)$value['player_id'] ) );
       $playerName = $getPlayerName->fetch( PDO::FETCH_ASSOC )['name'];
       array_push($response['players'], array("playerId" =>
(int)$value['player_id'],
                                              "playerName" => $playerName,
                                              "x" => ( int )$value['x_coord'],
                                              "y" => ( int )$value['y_coord'],
                                              "health" => ( int
)$value['health'],
                                              "turn" => ( int )$value['turn'],
                                              "race" => ( int
)$value['race']));
   }
}
print( json_encode( $response ) );
```

Test cases

Test case #	Group ID	Player ID	Expected result
T01	ID < 1	any ID	Empty JSON
T02	ID > 1	ID < 0 and not in group	Empty JSON
T03	ID > 1	ID > 1 and not in group	Empty JSON
T04	ID > 1	ID > 1 and exist in group	Player/Enemy JSON Info
T05	ID > 1	ID < 1 and exist in group	Player/Enemy JSON Info

Unit testing platform for server-side scripts

```php
<?php

error_reporting( E_ALL );

$data = array(
    'sessionid'     => '06fbaf9a2d24240583096500846397bf',
        'playerId'     => '39',
    'groupid'     => 2);

$options = array(
      'http'     => array(
        'method'      => 'POST',
        'Content'    => json_encode($data),
        'Header'    => 'Content-Type: application/json\r\n' .
                       'Accept: application/json\r\n'));

$url = "http://alexviznytsya.me/dungeoncrafter/gameplay.php";

$context = stream_context_create( $options );
$result = file_get_contents( $url, false, $context );
var_dump($result);
```

Changing the `$data` array should result in different output from the database.

Database snippet before tests

group_id	player_id	x_coord	y_coord	race	health	class	turn
7	-1	10	12	105	100	-1	1
7	-2	17	10	106	100	-1	0
7	39	13	2	101	100	201	1
7	47	3	9	103	100	202	0

Test Results

#T01:

```
$data = array(
    'sessionid'     => '06fbaf9a2d24240583096500846397bf',
        'playerId'     => 39,
    'groupId'     => -1);
```

Script output from running `tester.php`:

```
JSON:
{"players":[],"enemies":[]}
```

#T02:

```
$data = array(
    'sessionid'     => '06fbaf9a2d24240583096500846397bf',
        'playerId'     => -3,
    'groupId'     => 7);
```

Script output from running `tester.php`:

```
JSON:
{"players":[],"enemies":[]}
```

#T03:

```
$data = array(
    'sessionid'     => '06fbaf9a2d24240583096500846397bf',
        'playerId'     => 40,
    'groupId'     => 7);
```

Script output from running `tester.php`:

```
JSON:
{"players":[],"enemies":[]}
```

#T04:

```
$data = array(
    'sessionid'     => '06fbaf9a2d24240583096500846397bf',
        'playerId'     => 39,
    'groupId'     => 7);
```

Script output from running `tester.php`:

```
JSON:
{"players":[{"playerId":39,"playerName":"sean","x":13,"y":2,"health":100,"turn"
:1,"race":101},{"playerId":47,"playerName":"Elf","x":3,"y":9,"health":100,"turn
":0,"race":103}],"enemies":[{"playerId":-
1,"playerName":"Enemy","x":10,"y":12,"health":100,"turn":1,"race":105},{"player
Id":-2,"playerName":"Enemy","x":17,"y":10,"health":100,"turn":0,"race":106}]}
```

#T05:

```
$data = array(
    'sessionid'     => '06fbaf9a2d2424058309650084 6397bf',
        'playerId'      => -1,
    'groupId'     => 7);
```

Script output from running `tester.php`:

```
JSON:
{"players":[{"playerId":39,"playerName":"sean","x":13,"y":2,"health":100,"turn"
:1,"race":101},{"playerId":47,"playerName":"Elf","x":3,"y":9,"health":100,"turn
":0,"race":103}],"enemies":[{"playerId":-
1,"playerName":"Enemy","x":10,"y":12,"health":100,"turn":1,"race":105},{"player
Id":-2,"playerName":"Enemy","x":17,"y":10,"health":100,"turn":0,"race":106}]}
```

16 private int rollDice()

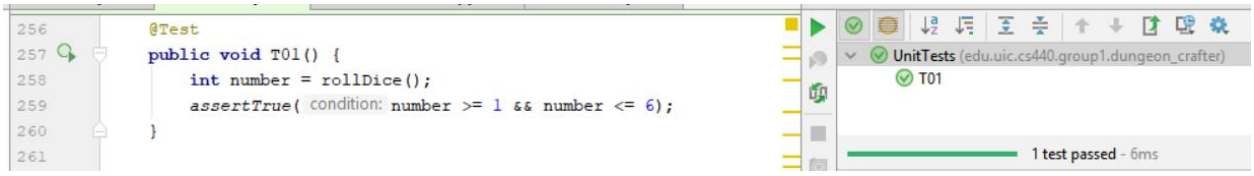
Code to be tested

```
private int rollDice()
{
  Random rand = new Random();
  int remMoves = rand.nextInt(6) + 1;
  rollTxtView.setText(Integer.toString(remainingMoves));
  hasRolled = true;
  return remMoves;
}
```

Test cases

Test case #	Input	Expected result
T01	N/A	0 < x < 7

Test Results



IV Conclusion

In summary, the results of the inspection revealed a number of areas with potential to increase efficiency. In addition, a number of styling inconsistencies throughout the program were apparent. The correction of the inconsistencies allowed for greater readability of the code. With regards to the unit testing, we were successful in uncovering two bugs. These bugs resulted from not addressing null being passed to the methods.

Bibliography

Dave, Jay, et al. *Dungeon-Crafter*. 2014, pp. 1–72, *Dungeon-Crafter*.

Wiegers, Karl E. “Checklist for Code Reviews.” 2001.