Advanced Application Development

Michael Little – 1103677

**King of Tokyo**

**Test Specification**

**Contents**

1. Introduction
   1. Purpose
   2. Additional Information
   3. References
2. Application Overview
3. Testing
   1. Functionality Testing
      1. Dice roll Functionality
      2. Process Results Functionality
      3. Attack functionality
   2. Unit Testing
      1. Player Class
      2. Dice Class
      3. Gameplay Class
4. Functionality Testing Results
   1. Dice Roll Functionality
   2. Process Results Functionality
   3. Attack Functionality
5. Unit Testing Results
   1. Player Class
      1. Testing Stage 1
   2. Dice Class
      1. Testing Stage 1
   3. Gameplay Class
      1. Testing Stage 1
6. **Introduction**
   1. **Purpose**

The purpose of this document is to detail the various tests that will be used to ensure the functionality of the associated application King of Tokyo.

* 1. **Additional Information**

This Project has been managed and run by Michael Little, and as such all testing responsibilities fall on him.

* 1. **References**

King of Tokyo. 2010. [Board] Iello, Richard Garfield.

Iello 2010. King of Tokyo Instruction Leaflet

1. **Application Overview**

The application that is being tested is a digital version of the Board game King of Tokyo. The game is a turn based competitive game where up to 4 players compete to rule the city of Tokyo. Each player spends their turn rolling dice and using the results to determine their actions for the turn, depending on the results players can:

* Earn victory points to help them win the game.
* Attack the other players
* Heal damage
* Gain energy to spend on cards

The game is over when either there is only one monster left standing, or one player gains 20 Victory Points.

1. **Testing**

The testing stage for this application will ensure the various features of the application are working accordingly. This will cover the mouse controls, player and dice classes and the general gameplay flow.

Mouse Control testing will determine:

* The functionality of the various buttons in the games UI.

Player Class testing will determine

* The player stats functionality.

Dice Class testing will determine

* The dice roll functionality.
* The dice hold functionality.
* The process result functionality.

And Gameplay testing will determine

* The attack functionality.
* The Player rotation Functionality
* Win/lose functionality.

Overall testing of the application will hopefully eliminate bugs, and glitches as well as improve the general functionality of the application. For the purpose of this document, the Functionality testing section will cover the methodology of the mouse controls as well as the player and dice classes. The Unit Testing will cover the individual sections of the application.

* 1. **Functionality Testing**
     1. **Dice Roll Functionality**

**Unit:** DiceRoll();

**Purpose:** Checks if the method can be called within another method in another class.

**Test Inputs:** Click the roll button.

**Test Procedure:** Place Breakpoint within the DiceRoll() function located in the Dice Class.

**Process:**

* Run Application
* Click the Dice Roll Button

**Pass/Fail Conditions:**

If all works out according to plan Visual Studios will pause the application at the point in the code where the breakpoint was inserted, if not then the method wasn’t called and the test fails.

**Notes:**

This doesn’t test the functionality of the method as it was yet to be implemented, it merely test that the function can be called successfully within a method in another class.

**Results for this test are shown in section 4**

* + 1. **Process Result Functionality**

**Unit:** ProcessResult();

**Purpose:** The method takes in 6 numbers, processes them and then takes the appropriate results.

**Test Inputs:** The Roll dice and Hold dice buttons are pressed until an appropriate results are reached, then the roll button is clicked once more to process the result.

**Test Procedure:** A breakpoint is placed within the method so that a step by step process can begin which checks that each variable is processed and the appropriate responses are taken.

**Process:**

* Run the Application
* Click the roll button and the hold dice buttons until appropriate results for the test are gained.
* Click the roll button once more to process the results.

**Pass/Fail Conditions:** If done correctly the players statistics should change according to the results processed. If not the test is failed.

**Notes:** The test had to be carried a few times due to the dice results being random.

**Results for this test are shown in section 4**

* + 1. **Attack Functionality**

**Unit:** Attack();

**Purpose:** Check whether or not the attack function works.

**Test Inputs:** The Attack result was rolled and processed.

**Test Procedure:** The current player’s variable “bool inTokyo” was set to true, and a break point was placed within the code.

**Process:**

* Run the Application
* Click the roll button and the hold dice buttons until appropriate results for the test are gained.
* Click the roll button once more to process the results.

**Pass/Fail Conditions:** If the test was successful the other players health should be reduced by the amount of Attack results were rolled. If not test is failed.

**Notes:** The test had to be carried a few times due to the dice results being random.

**Results for this test are shown in section 4**

* 1. **Unit Testing**
     1. **Player Class**
        1. **Test Stage 1 Unit:** Player class

**Purpose:** To determine if different instances of the player class can have its variables edited by the other players.

**Test Inputs:** breakpoints within certain functions.

**Test Procedure:** with different players, roll the specific results with the dice, then check that the appropriate variables change, e.g. rolling energy adds 1 energy.

**Process:**

* Run Application
* Roll the specific results for testing and process them.
* Check that the specified variable for the player has changed.
* Repeat with different player.

**Notes:** The test will have to be carried a few times due to the dice results being random.

**Results for this are displayed in section 5.**

* + 1. **Dice Testing**
       1. **Test Stage 1 Unit:** Dice Class

**Purpose:** To determine if the dice produce different results each time.

**Test Inputs:** Dice Roll button will be clicked.

**Test Procedure:** Run the Application multiple times, then roll all 6 dice several times and see if the results differ enough.

**Process:**

* Run the application
* Roll the dice 3 times, and note the results.
* Close and repeat previous steps.

**Results for this are displayed in section 5**

* + 1. **Gameplay Testing**
       1. **Test Stage 1 Unit:** Gameplay Class

**Purpose:** To determine if defeated players are removed from the turn rotation.

**Test Inputs:** the application will be run several times and the dice roll button will be clicked several times.

**Test Procedure:** Dice will be rolled several times focusing on attacking the players, after one of the players dies play will continue until it comes to their turn, if it skips their turn the test was a success.

**Process:**

* Run the application
* Doll the dice until a player is defeated
* See if they get a turn

**Results for this test are shown in section 5**

1. **Functionality Testing Results**
   1. **Dice Roll Functionality**

**Procedure:**

Place Breakpoint within the DiceRoll() function located in the Dice Class.

**Outcome:**

Once the Application was running, the dice button was clicked using the mouse, the application then paused itself and visual studios opened up the dice class at the exact point the breakpoint was placed.

**Pass/Fail:**

The test was a success as the DiceRoll() function was successfully called within another method.

* 1. **Process Results Functionality**

**Procedure:**

A breakpoint is placed within the method so that a step by step process can begin which checks that each variable is processed and the appropriate responses are taken.

**Outcome:**

After the application was run several times where different results were processed, the different variables of the player class did change accordingly. With the exception of the attack result as the method had yet to be written.

**Pass/Fail:**

The test was a success as the player’s health, energy and victory points did change according to the results of the 6 dice.

* 1. **Attack Functionality**

**Procedure:**

The current player’s variable “bool inTokyo” was set to true, and a break point was placed within the code.

**Outcome:**

After the application was run and the attack result was rolled, the results were processed, the other players health then reduced by the number of attack results processed.

**Pass/Fail:**

The test was a success as the other players took he appropriate amount of damage, and development then continued into the TakeTokyo() method.

1. **Unit Testing Results**
   1. **Player Class**
      1. **Testing Stage 1**

**Test Procedure:**

With different players, roll the specific results with the dice, then check that the appropriate variables change, e.g. rolling energy adds 1 energy.

**Outcome:**

After running the application, the dice were rolled and processed with different players, the results were then checked so that they were correct.

**Pass/Fail:**

The test was successful as all of the players stats were appropriately affected by the dice rolls.

* 1. **Dice Class**
     1. **Testing Stage 1**

**Test Procedure:**

Run the Application multiple times, then roll all 6 dice several times and see if the results differ enough.

**Outcome:**

The 6 dice were rolled 3 times and the results were noted down, the application was then re-run and the dice were then rolled again.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attempt 1 | Dice 1 | Dice 2 | Dice 3 | Dice 4 | Dice 5 | Dice 6 |
| Roll 1 | 1 | 1 | 4 | 2 | 6 | 1 |
| Roll 2 | 6 | 4 | 1 | 4 | 3 | 2 |
| Roll 3 | 2 | 6 | 4 | 1 | 4 | 3 |
| Attempt 2 | **Dice 1** | **Dice 2** | **Dice 3** | **Dice 4** | **Dice 5** | **Dice 6** |
| Roll 1 | 6 | 4 | 1 | 5 | 4 | 4 |
| Roll 2 | 6 | 6 | 5 | 6 | 4 | 4 |
| Roll 3 | 2 | 3 | 5 | 1 | 3 | 4 |
| Attempt 3 | **Dice 1** | **Dice 2** | **Dice 3** | **Dice 4** | **Dice 5** | **Dice 6** |
| Roll 1 | 4 | 1 | 4 | 2 | 2 | 3 |
| Roll 2 | 1 | 4 | 1 | 3 | 1 | 4 |
| Roll 3 | 4 | 6 | 6 | 3 | 3 | 4 |
| Attempt 3 | **Dice 1** | **Dice 2** | **Dice 3** | **Dice 4** | **Dice 5** | **Dice 6** |
| Roll 1 | 6 | 5 | 4 | 4 | 3 | 3 |
| Roll 2 | 6 | 3 | 4 | 6 | 3 | 1 |
| Roll 3 | 3 | 1 | 3 | 4 | 4 | 5 |

**Pass/Fail:**

The dice results appear to be random, so the test is a success.

* 1. **Gameplay Class**
     1. **Testing Stage 1**

**Test Procedure:**

Dice will be rolled several times focusing on attacking the players, after one of the players dies play will continue until it comes to their turn, if it skips their turn the test was a success.

**Outcome:**

The attack function was called several times until a player was successfully killed (player 2), the turn rotation then continued as normal until it came to just before the dead players turn. The pass button was presses and it skipped past the dead player on to the next living player (player 3).

**Pass/Fail:**

The dead player was removed from the test rotation, so the test was a success.