**SYSC 3010 Project Test Plan**

**R Tanks (Group W7)**

**Group Members:** Jake Bridge, 101072342

Thomas Sloan, 101023007

Mahmoud Al Sous, 101071213

Marc Donovan, 101073218

**Project Overview**

The goal of the rTanks project is to create a laser-tag experience where users will sync their phones to a miniature remote-control tank and be able to battle other users. One tank and one phone are required per player. In the game, there are remote-control tanks that will utilize IR emitters to serve as the tanks “cannon” and IR receivers to tell if a tank was “hit” by another tanks shot. The tanks will be controlled using a smart-phone app connected via Bluetooth. Specifically, the connection is made using UART Bluetooth from a smartphone to an Adafruit Bluefruit Arduino that controls the tank’s sensors and motors. The architecture diagram of the project is given below.



Figure 1: Architecture Diagram of rTanks System

**Project Testing**

1. **General Utility Class**

**Context:** The tank is controlled using an Arduino with a Bluetooth connection from a smartphone application

**Test information:**

* The Bluetooth connectivity class will be tested.
* The test will include procedures for the 3 types of data we expect to receive from the smartphone application: The press and release of a button, a button pressed and held for some amount of time and finally reading the position of the joystick on the application.
* The test will ensure that the smartphone is able to properly communicate with the Arduino that controls the tank.

**Test Procedure:**

Table 1: General utility class test procedure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Test Name | Description | Test Steps | Inputs | Expected Outputs |
| 1 | sendAndRecord | Press a button that sends a signal to the server and records the results | 1. Set a value for the button. 2. Send value when the button is pressed. 3. Server receives value and records it. 4. Check if the value is equal to the button’s value. | “2” | “True” |
| 2 | buttonHold | Press and hold a button keeps sending signals to the server until the user lets go of the button. Each signal shall be recorded. | 1. Set a value for the button. 2. Send values when the button is held. 3. Server receives values and records them. 4. Check if the value is equal to the button’s value. | “3 3 3 3 3 3 3 3 “ | “True” |
| 3 | JoyStickMovements | Send x and y coordinates of the position of the joystick to the server and check if the correct coordinates are received. | 1. Move Joystick in a direction 2. Send x and y coordinates to the server 3. Check server if the same coordinates are received. | “X = 75  Y = 187” | “True” |

1. **Hardware Unit Test**

**Context:** The tank uses two types of motors. A stepper motor is used to drive the tank and servo motors are used to for the turret and to steer the tank. The stepper motor is attached to the rear wheels and can make the tank: move forwards, backwards or stop.

**Test Information:**

* For the hardware unit test we will be testing the drive motor of the tank. This is implemented in the system using a stepper motor.
* This test ensures that the tank will be able to move forwards, backwards and stop.

**Test Procedure:**

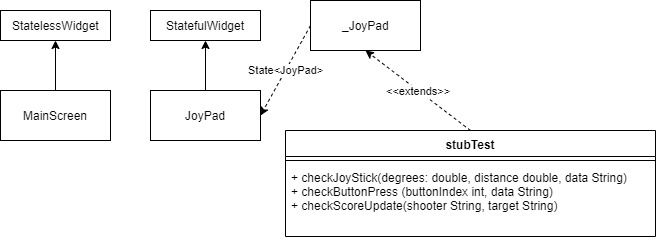
Table 2: Hardware unit test procedure

|  |  |  |
| --- | --- | --- |
| Test Description | Test Steps | Expected Result |
| No input (Initial Condition)   * This test will simulate no input from the user | 1. Do nothing | The motor should keep a constant position. |
| Move: Forwards   * This test will simulate a user that wants their tank to move forwards | 1. Set the car to brake 2. Wait 200 ms to ensure braking is achieved 3. Set the car to forwards at 20% speed 4. Delay 1 second | The motor should spin clockwise, which in turn moves the tank forwards. |
| Move: Backwards   * This test will simulate a user that wants their tank to move backwards | 1. Set the car to brake 2. Wait 200 ms to ensure braking is achieved 3. Set the car to backwards at 20% speed 4. Delay 1 second | The motor should spin counter-clockwise, which in turn moves the tank forwards. |
| Max speed   * This test will test the motor at its maximum speed | 1. Set the car to brake 2. Wait 200 ms to ensure braking is achieved 3. Set the car to forwards at 100% speed 4. Delay 1 second | The motor should rotate at its maximum defined speed. |
| Half speed   * This test will test the motor at half of its maximum speed | 1. Set the car to brake 2. Wait 200 ms to ensure braking is achieved 3. Set the car to backwards at 50% speed 4. Delay 1 second | The motor should rotate at half of its maximum defined speed. |

**Stub Test**

The stub class will only be used if the variable stubTest is set to true. In this case, the arduino will not establish a bluetooth connection with the Arduino but simply compare messages that will be sent from the smartphone application from different interactions with the smartphone. Below is the class diagram of the system. The MainScreen object is the object that determines the basic layout of the application. The Joypad object simply initiates an instance of \_JoyPadState to commence application. stubTest will be called within \_JoyPad to verify that the information being generated is valid and correct by doing one of the following

1. Calculating the result given certain values and verifying the correct string has been generated
2. Verify that the index of the button corresponds with the message being sent.

****

checkJoyStick will use degrees and distance to calculate the correct string to be generated and compare it to the string that was passed through

checkButtonPress will verify that the buttonindex corresponds with the correct message being sent

checkScoreUpdate will verify the correct names are being passed to the database by generating String within the checkScoreUpdate method.

1. **Distributed System Test**

The objective of testing the smartphone application as a mock object is to make sure it can communicate with the database as well as the tanks because the smartphone application controls the tanks and is the main communicating component in the system. The following table shows a list of test cases for the smartphone application:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Title | Description | Context | Trigger | Response |
| 1 | testButtons | The database is used to keep track of the values sent by pressing the buttons in the controlling screen. | Connection between application and database. | Button is pressed. | Database receives values and views them. |
| 2 | testJoystick | The database is used to keep track of the values sent by movements of the joystick in the controlling screen. This tests if database receives values for each of the joystick movements. | Connection between application and database. | Joystick is moved. | Database receives values and views them. |
| 3 | testTankConnection | The user connects to the tank using the smartphone application. This is to test if the smartphone is capable of connecting to the tank. | Connection between application and tank. | A button is pressed. | Server outputs a message confirming whether it is connected or not. |
| 4 | testScoreUpdate | The score is stored in the database. The application should update the score throughout the match whenever a player scores. | Connection between application and database. | A player scores. | Database overwrites the score with the updated score values. |
| 5 | testTankOccupation | This tests the tank selection between players. If the tank is already connected to a different device. Do not connect to it. | Connection between application and tank. | A button is pressed. | (Error case)  Server sends a message confirming that the tank is connected to the “new” user. |
| 6 | testGoogleSignIn | The smartphone application uses google servers to authorize the user’s credentials when logging in. This test verifies if the information the user entered is valid or not. | Connection between application and server. | The login button is pressed. | (Error Case)  Server does not accept login information. Application provides an error message notifying the user that the login has failed. |

**Acceptance Testing**

**Functional Requirements:**

1. Users are able to sign in to the application with their Google log-in credentials and connect via bluetooth to a tank
2. Users are able to move their tank using the applications joystick
3. Users are able to angle the direction of the cannon on their tank using the application
4. Users are able to “fire their cannon” (emit IR light) using the application
5. Tanks are able to tell when they have been hit with IR light and report it to the application
6. Information can be added to the database
7. Scoreboard can be updated

**Acceptance Testing:**

Table 4: Testing scenarios and acceptable requirement functionality

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Case ID** | **Description** | **Req.1** | **Req.2** | **Req.3** | **Req.4** | **Req.5** | **Req.6** | **Req.7** |
| 1 | User opens the application on their phone | User should be prompted with Google sign in followed by a prompt to select a tank | N/A | N/A | N/A | N/A | Users account is stored in database | N/A |
| 2 | User moves their joystick around | N/A | The stepper (driving) motor and servo (steering) motor will mimic the position of the joystick to move like a generic remote control car | N/A | N/A | N/A | N/A | N/A |
| 3 | User presses their arrows used to angle position of the cannon | N/A | N/A | 1.If moved left the cannon will angle left  2.If moved right the cannon will angle right | N/A | N/A | N/A | N/A |
| 4 | User fires cannon and hits the other tank | N/A | N/A | N/A | IR emitter emits light from tank A | The IR receiver on tank B senses the IR light and reports it to the application | Database is updated to reflect tank A getting a hit and tank B being hit | +1 on scoreboard for tank A |
| 5 | User fires cannon and misses the other tank | N/A | N/A | N/A | IR emitter emits light from tank A | The IR receiver on tank B does not sense the IR light | N/A | No change |