Testing Document
PacMan
version: 1

Group F8
1155127434 HO Chun Lung Terrance
Department of Philosophy, The Chinese University of Hong Kong
1155143519 WOO Pok
Department of Physics, The Chinese University of Hong Kong
1155157839 NG Yu Chun Thomas
Department of Computer Science and Engineering, The Chinese University of Hong Kong
1155157719 LEUNG Kit Lun Jay
Department of Computer Science and Engineering, The Chinese University of Hong Kong
1155143569 MOK Owen
Department of Mathematics, The Chinese University of Hong Kong

May 5, 2023

Contents

1 TEST PLAN **2**

2 TEST CASES **3**

2.1 Shop System 3
2.2 Setting 4
2.3 Game Status 6
2.4 Character's Movement and controls 7

1 TEST PLAN

In order to test the software properly, we will apply a bigbang approach of software testing. We consider for each functionality of the software, there will be pure approaches (black- or white-box testing). To be specific, for complicated functionalities (mostly inter-module functionals and implementational functionals), we choose to apply black-box testing; for simple functionalities (mostly mono-module functionals and object reactions), we choose to apply white-box testing. Afterall, we will integrate the modules and test the whole system directly.

The testing will be under following environment:

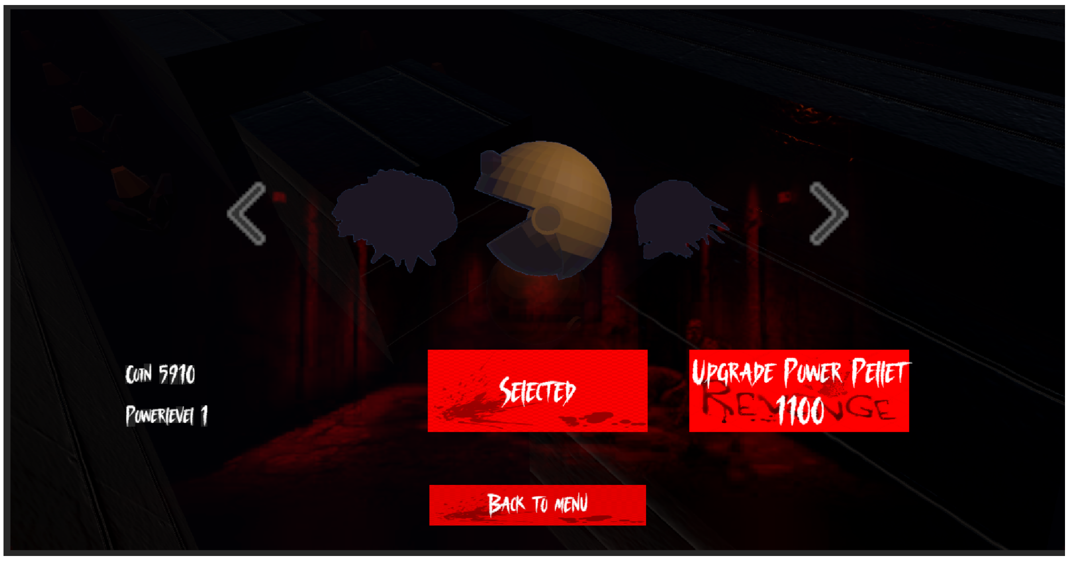
- Software: Unity (version 2021.3.17f1), Windows7
- Hardware:
  - CPU: i5-8300
  - NVidia-GTX960M
  - 8GB-2400MHZ

We list out the functionalities to be tested:

1. White box testing:
   (a) User Management
   (b) Player Data
2. Black box testing:
   (a) Menu IU
   (b) Shop system
   (c) Setting
   (d) Game Status
   (e) Player control
   (f) Ghost AI & Status

2 TEST CASES

2.1 Shop System



Under the assumption of black box testing, we focus on the functionalities of each button:

1. The 'selected' button should be changed to 'buy' or 'select' when the character focus is changed, and should be clickable to select the character.
2. The upgrade button should be changing the power level and coins and the value inside the button.
3. the 'back to menu' button should be able to direct player to the menu.
4. The left right button should beable to change the character shown.

Inputs and expected outputs

1. Test Case 1: upgrade power pellet
   (a) Input: Clicking the upgrade button
   (b) Expected Output: coin decreased, powerlevel and value on the button increased.
2. Test Case 2: Clicking the back to menu button
   (a) Input: click the button
   (b) Expected Output: Back to menu
3. Test Case 3: Change to other character, and buy it.

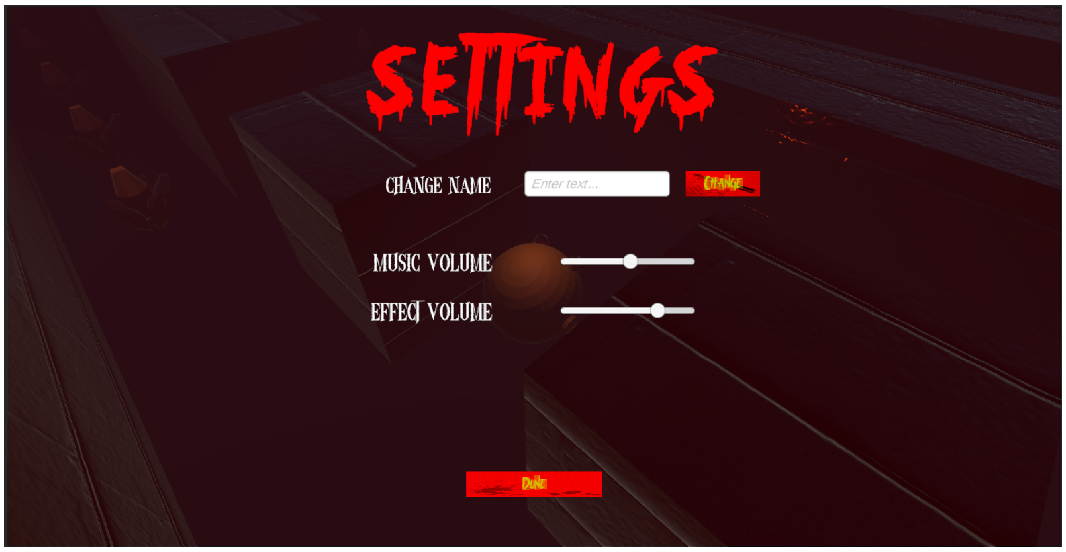
   (a) Input: click left right button and buy.
   (b) Expected Output: character changed, and it will be successfully bought if coins enough, coins decresed, and button becomes selected; unsuccesful purchase will remain everything unchanged.

4. Test Case 4: Clicking blank space
   (a) Input: Clicking on blank area.
   (b) Expected Output: No output.

Results

The above results to be successful under the assumption of black box testing.

2.2 Setting



Under the assumption of black box testing, we focus on the functionalities of each button:

1. Name can be changed in the change name field, by clicking the change button.
2. Music and Effect Volume can be adjusted.
3. Done button will direct user to previous page.

Inputs and expected outputs

1. Test Case 1: Directly clicking change button

    (a) Input: Clicking the change button

    (b) Expected Output: The name cannot be changed

2. Test Case 2: Input a new name and click the change button

    (a) Input: new name, change button

    (b) Expected Output: Name is changed.

3. Test Case 3: Adjust Music Volume to 0.

    (a) Input: adjust the slider of music volume

    (b) Expected Output: No music is played.

4. Test Case 4: Adjust Effect Volume to 0.

    (a) Input: adjust the slider of effect volume

    (b) Expected Output: No sound effect is played.

5. Test Case 5: Adjust Music Volume to nonzero.

    (a) Input: adjust the slider of music volume

    (b) Expected Output: music volume varies with slider.

6. Test Case 6: Adjust Effect Volume to nonzero.

    (a) Input: adjust the slider of effect volume

    (b) Expected Output: effect volume varies with slider.

7. Test Case 7: Clicking done button with different entrance

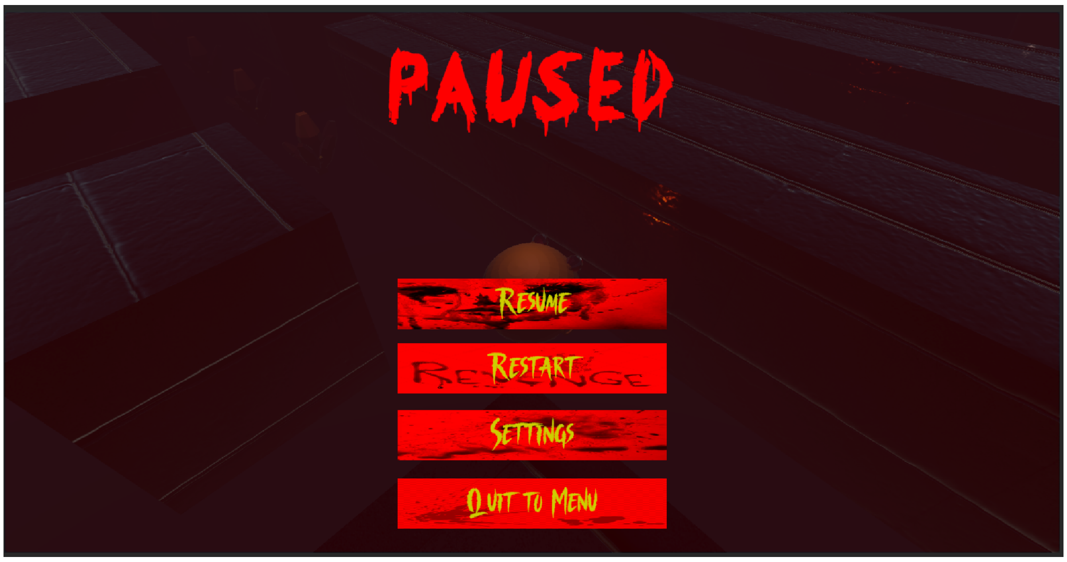
    (a) Input: Go to setting from different entrance, and click the done button

    (b) Expected Output: Returning the previous page.

Results

The above results to be successful under the assumption of black box testing.

2.3 Game Status



Under the assumption of black box testing, we focus on the functionalities of each button:

1. By clcking spacebar, we can enter the pause section
2. Resume button allows player resume the game
3. Restart button allows player restart the game
4. Settings button allows player enter setting page
5. Quit to menu allows player to halt the game

Inputs and expected outputs

1. Test Case 1: Never pressing spacebar during the game
   (a) Input: Usual game control
   (b) Expected Output: The pause section is never been called.
2. Test Case 2: Pressing spacebar anytime during the game
   (a) Input: press the spacebar
   (b) Expected Output: Pause section is called
3. Test Case 3: Clicking resume button
   (a) Input: resume button

   (b) Expected Output: Game continued.

4. Test Case 4: Clickiong restart button
   (a) Input: restart button
   (b) Expected Output: Game restart
5. Test Case 5: Clicking settings button
   (a) Input: settings button
   (b) Expected Output: entering settings
6. Test Case 6: Clicking Quit to menu button
   (a) Input: quit to menu
   (b) Expected Output: diresting player to menu

Results

The above results to be successful under the assumption of black box testing.

2.4 Character's Movement and controls

For character movement control (both the player and the enemies), the main class of interest is the movement class controlling the player's action. We will be focusing on white box testing on the function SetDirection().

```
public void SetDirection(Vector3 targetDir, bool forced = false) {
    Debug.Log("SetDirection() called with target direction: " +
        targetDir.ToString() + " and forced: " + forced.ToString());

    if (isBlocked(targetDir) && !forced) {
        nextDirection = targetDir;
        Debug.Log("Path is blocked. Setting nextDirection to: " +
            nextDirection.ToString());
    }
    else {
        nextDirection = targetDir;
        direction = targetDir;
        nextDirection = targetDir;
        Debug.Log("Path is not blocked. Setting direction and nextDirection
            to: " + direction.ToString());
        rotation = new Vector3(0, Mathf.Atan2(direction.z, -1 * direction.x) *
            Mathf.Rad2Deg - 90, 0);
        if (rotation.y < 0) rotation += new Vector3(0, 360, 0);
        if (rotation.y >= 360) rotation -= new Vector3(0, 360, 0);
        // Force rotation.y to become 0 or 90 or 180 or 270
        rotation = new Vector3(0, Mathf.Round(rotation.y / 90) * 90, 0);
        Debug.Log("Calculated rotation: " + rotation.ToString());
    }

    // Add this debug line to log the current direction, nextDirection, and
        rotation
    Debug.Log("Current direction: " + direction.ToString() + ", nextDirection:
        " + nextDirection.ToString() + ", rotation: " + rotation.ToString());
}
```

And here is the function update() in movement.cs who take care of the variable NextDirection.

```
private void Update() {
    if (nextDirection != Vector3.zero) {
        SetDirection(nextDirection);
    }

    // Add this debug line to log the current nextDirection value
    Debug.Log("Current nextDirection: " + nextDirection.ToString());
}
```

Here we added debug message for checking the validity of the output direction sequence.

Inputs and expected outputs

1. Test Case 1: Pacman moves in an open path
   (a) Input: targetDir = Vector3.forward (0, 0, 1)
   (b) Expected Output: direction and nextDirection should be set to Vector3.forward, and the appropriate rotation should be calculated.
2. Test Case 2: Pacman moves in a blocked path without forced movement
   (a) Input: targetDir = Vector3.left (1, 0, 0) and a wall in the left direction
   (b) Expected Output: nextDirection should be set to Vector3.left, but direction should not change.
3. Test Case 3: Pacman moves in a blocked path with forced movement

   (a) Input: targetDir = Vector3.left (1, 0, 0), a wall in the left direction, and forced = true
   (b) Expected Output: direction and nextDirection should be set to Vector3.left, and the appropriate rotation should be calculated.

4. Test Case 4: Pacman changes direction when a previous blocked path becomes unblocked
   (a) Input: Pacman is moving forward (0, 0, 1) and attempted to turn left (1, 0, 0) but it was blocked. The wall blocking the path is now removed.
   (b) Expected Output: direction and nextDirection should be set to Vector3.left, and the appropriate rotation should be calculated.

Outputs and analysis

Here are the output log messages:

1. Test Case 1: Pacman moves in an open path

```
SetDirection() called with target direction: (0, 0, 1) and forced:
    False
Path is not blocked. Setting direction and nextDirection to: (0, 0, 1)
Calculated rotation: (0, 0, 0)
Current direction: (0, 0, 1), nextDirection: (0, 0, 1), rotation: (0,
    0, 0)
```

2. Test Case 2: Pacman moves in a blocked path without forced movement

```
SetDirection() called with target direction: (1, 0, 0) and forced:
    False
Path is blocked. Setting nextDirection to: (1, 0, 0)
Current direction: (previous direction), nextDirection: (1, 0, 0),
    rotation: (previous rotation)
```

3. Test Case 3: Pacman moves in a blocked path with forced movement

```
SetDirection() called with target direction: (1, 0, 0) and forced:
    True
Path is not blocked. Setting direction and nextDirection to: (1, 0, 0)
Calculated rotation: (0, 270, 0)
Current direction: (1, 0, 0), nextDirection: (1, 0, 0), rotation: (0,
    270, 0)
```

4. Test Case 4: Pacman changes direction when a previous blocked path becomes unblocked

```
Current nextDirection: (1, 0, 0)
SetDirection() called with target direction: (1, 0, 0) and forced:
    False
Path is not blocked. Setting direction and nextDirection to: (1, 0, 0)
Calculated rotation: (0, 270, 0)
Current direction: (1, 0, 0), nextDirection: (1, 0, 0), rotation: (0,
    270, 0)
```

The test cases are successful. Furthermore, simple black box testing (on modules other than this one, or say the overall movement ability) was also done when testing this current module. We conclude that we did not discover significant faults on the character movement feature.