Team C5

Four Kingdoms of Kralpridan

Test Report

Jordan Bell
Alexander Coleman
Priyan Mistry
Krishna Patel

Contents

Introduction

This test plan is for the game Four Kingdoms of Kralpridan (FFK). To ensure everything is tested this test plan will contain the methods chosen for testing and the procedures for these methods to follow when testing all aspects of the FFK game.

What will be covered in the testing?

The different components of the game that has been tested

- User interface
- Game Engine
- Networking
- Players (Stats)
- Artificial Intelligence (Auto Player)

These different components of the game will be tested to find any errors or issues within the game that need to be looked at. The high level, low level and the HCI components of the overall game is tested to find any defects with the interaction or game functionality.

Methods of Testing (Approach)

Three methods of testing have been to test this game. These three methods are Usability Testing, JUnit Testing and Integration Testing. Testing was done throughout the whole project to fix bugs whilst creating the game to ensure the game works efficiently and the game balance fits so that the game is competitive, meaning not too easy or not too hard.
Usability Testing was done to gather feedback on how usable our game. This was done near the end of the project where the game had almost full functionality so we could get productive feedback on the game as a whole for the usability. From this we could make any changes to the game based on the feedback we received.
JUnit Testing was created near the end of the project. These tests were created to test individual methods and classes within the project to seek out any errors within the code and ensure the code does what it was planned to do. Integration testing is done as a part of JUnit testing to check for any issues/errors when merging different components together.

Pass/Fail Criteria

Usability Testing

There was no pass/fail criteria for Usability testing as this involves giving the game to a user to test how easy the game as a whole is to use/play. So, from here we are receiving feedback and making changes based on the users perceptions.

JUnit Testing

The criteria of completing this phase of testing is that every test case that has been created for the all aspects of the game runs without any errors occurring. The reason for this is because a single error could have an effect on the functionality or the efficiency of the game.

Integration Testing

The criteria for Integration testing to be passed is when each individual unit test for each class or component that is being integrated together must have passed. The reason for this is because an error could have a negative effect on the integration between two components, meaning they are not integrated properly.

Usability Testing

The user acceptance testing was done when the game had almost fully functionality overall for both the user interface and the actual gameplay. As this was done towards the end of the development of the game, we were able to give the game to many users of varying experience and skill levels of games. From the responses we received we were able to gage how to users perceived the game and make any changes based on these perceptions.

User Reported Bugs

Reported Bug ID	**Bug found**	**Changes Made**
1.	Health drops below 0 into negative values.	Amended code to ensure player and A.I health does not drop below 0.
2.	Pets are placed on enemy team	Redesigned monster instantiation, fixing the bug occurrence. Pet monsters now stay on players team rather than switch to enemy team.
3.	On some builds game would crash when the battle timer reached 00:00	Discovered this only occurred, consistently, on outdated code. No reported occurrences on newer versions.
4.	Levelling pacing way off – sometimes levels several times at once	Tweaked data values to reduce the experience multiplier on monsters so the player doesn't gain multiple levels at once. This was again readjusted later to facilitate a faster game.
5.	Old enemies re-appearing in battles with negative health.	Enemies were not being copy constructed; fixed so that each battle produces a new version of an enemy.

User Critical Comments

Critical Comment ID	**Comment Made**	**Changes Made**
1.	• Too easy • Need to up the difficulty, and then the game will be 10/10.	Monster difficulty has been significantly increased in zones 3 and 4, this has included enhancing stats and the move set of the monsters.
2.	• Target selection not clear, clicking enemy rather than arrows above them. • Arrows keys to cycle through the enemy selection • Click on Sprite rather than arrow for target selection.	Implemented the ability to click the actual monsters rather than arrows above them when performing an attack.

3.	Unclear which item equips to a particular slot in the inventory window.	Added picture in the inventory menu to which items should be equipped in which slot.
4.	• Slow pacing • Combat to slow	Increased the speed of which attacks happen in the battle. Adjusted health values on monsters and players so kills happen quicker.
5.	• Only being able to tame one animal	As a result we have implemented a system which allows you to tame new monsters. The discarded monster will remain in the game world where the player can come back and retrieve it if they wish.
6.	Probably too much battle encounters, it felt like I was stuck in the tall grass of Pokémon.	The chance of monster chance was decreased to enable the players to explore the world more rather than constantly entering battles.
7.	Can you add buttons to open up the inventory/skills. Didn't know what else you could do besides battle Need a popup to help skill buying.	We have a guide at the beginning of the game which explains the basic functionalities which seemed to be skipped immediately by most players. We have improved certain UI features of the game to make there functionality more obvious. We did find more experienced gamers had minimal issues playing the game.

Suggested Improvements

<u>**Suggested Improvement ID**</u>	<u>**Suggested Improvement**</u>	<u>**Changes Made**</u>
1.	More varied monsters	We have a total of 32 monsters in the game. They are split between 4 zones based on difficulty, most players did not go past the first zone due to slow combat and chance of battle, and these have been improved upon allowing players to explore the world more.
2.	Many moves felt redundant.	We have many moves including skill trees to gain further moves as you level up. However the game difficulty was fairly easy which meant players could use the basic "strike" attack to kill many of the monsters. However the game difficulty has been increased from user feedback and as a result players should have to use a wider range of the move set to win battles

Test Cases

These test cases were created and completed to test the functionality in the GUI from menu navigation to overall gameplay (characters moving around). They were started a week prior to the prototype presentation. The test cases can all be found in the Appendix, in Appendix A.

JUnit Testing

JUnit tests were carried out once changes were made after the responses from the User Acceptance testing. Each and every unit test carried out can be found on the SVN within the 'Testing' package in the source folder of the project.
The test classes created for unit testing are as follows (All test classes are within the 'Testing' package) : -

TestAI

This test class tests each AI to ensure they are behaving the way they should be. In the way that they are either attacking the weakest target or healing their team mate. So for each AI they are given fighters to attack or heal and they must attack or heal the correct one depending on which AI the Monster beholds. The outcome for each test in this class was a pass.

TestServerSide

This test class was used to test some of the processing that is done on the server. Mainly it tests the servers AI generation. The server contains dummy data values that is uses for the average battle stats when it is run from a test class. The tests check that all the AI stats are within the set parameters. Although it does throw errors at the end of the class, these are nothing to do with the server tests but actually related to the server missing some information when it isn't run with a full game. The outcome for each test in this class was a pass.

TestClientSide

This test class was used to test the client side of networking by testing the chat features for both team and everyone, updating of player stats and updating the team. This was done by setting up a server with a player, sending chats items and checking that client received what it should. This tests the network because the when I client sends a message it isn't just displayed in the chat box, instead the client sends a message to the server and the server echoes it out to all the clients. This echo is what the client's chat box displays. The outcome for each test in this class was a pass.

TestFighters

This test class was used to test the battle system features which include a fighter taking damage, healing fighters, enhancing fighters attack, defence and speed stats and also deducting move cost after a move is used. This was tested by creating a player fighter and a monster fighter and giving them moves. Then take damage was tested by using a move on the monster for both non-multiplier and multiplier moves to ensure the HP of monster goes down the same amount the move inflicted. The same principle for take heal, was done but for the player fighter. For enhancing attack, defence and speed – stats were added and checked to ensure the correct amount was added and the new stat was correct. Lastly for deducting cost and move was carried out by the player fighter and the MP was check to see if had been changed correctly. The outcome for each test in this class was a pass.

TestGameObject
This test class was used to test the game object which is the visual and functional components of the game. This was done by setting up parent, child and two leaf game objects (which includes setting position and name) and testing each of them by getting the names of child of the ‘parent’, and testing the coordinates of each game object. The outcome for each test in this class was a pass.

TestItemGeneration
This test class was used to test the item generation from chests and monsters within the game. This was done by initialising GameStateManager library, Item and Monster lists and rarity chances. For the Monsters there are 4 tiers which are tested separately by getting the chance each tiered item can occur for type of Monster. Then testing for all tiered monster at the same time to ensure you get the correct item chance. For each type of chest Iron, Silver and Gold – the item occurrence chance is tested to ensure there is not a higher chance of getting a certain tiered item. The outcome for each test in this class was a pass.

TestLevelProgress
This test class was used to test the level progress of the player, when a player increases level the battle stats and skill point need to be altered also. This was tested by setting battle stats and new level the adding enough experience to progress to new level, then other stats are tested to see if they are changed when the player has levelled up. The outcome for each test in this class was a pass.

TestPlayer
This test class was used to test the World Player class for the Overworldplayers movement, battle stats and monster (companion). This was tested by initialising game libraries, world player, world object and player model. The outcome for each test in this class was a pass.

TestRect
This test class was used to test the Rect class for simple rectangles using coordinates. This was used to build the world. It was tested by creating a rectangle and points using the functions within the Rect class. When these were created the methods within the Rect class was tested such as Halve and Double – to ensure you get the correct outcome when manipulating the input. The outcome for each test in this class was a pass.

TestResourceLoading
This test class was used to test the efficiency and the function of loading the resources and ensuring they only load once. It was tested by initialising the OpenGL libraries and loading multiple images (which are the same) in and ensuring they only load in once each different image, the same was done for each font type. The outcome for each test in this class was a pass.

TestSkillTree
This test class was used to test the skill tree for when unlocking and moves. This was tested by adding two moves in a ‘lock and key’ and testing when you unlock a move the correct move unlocks. Also when one move is dependent on another it gets unlocked only if previous move is unlocked, and ensuring you cannot unlock a move if you have no skill points. The outcome for each test in this class was a pass.

TestStatOfMax
This test class was used to test the maximum stats of any stat using max and current stat. This was done by setting the max stat, zero stat and locked stat and ensuring you get the correct outcome when altering the value. The outcome for each test in this class was a pass.

Integration Testing

Overview
For integration testing, we needed to ensure that each component of the game and its systems workedwell together, eliciting no errors and producing clear, expected results. For this, we adopted a top-down integration technique, starting with the overall engine and working our way down to each game state's functions.

These are the main systems that were to be integrated:

1. Game State Management of:
    1. Main Menu
    2. World Navigation
    3. Battles
2. UI and graphical representations of the above
3. Networking and multiplayer communication

The modular design of the game meant that we were able to work on each system independent from the others. This made it easier to find and resolve bugs, as each module was decoupled appropriately. Additionally, prior to integration testing, each developer was able to perform internal tests before performing integration tests with other modules.

Integration Technique
Having decided on a top-down integration technique, we started with a breadth-first testing plan, starting at the root: Game State Management. The game state system was tested using Unit tests, and were developed early on as it was vital to operating the different systems. Having tested the game state transitions, we could rely on the functions built thereafter which assumed reliable functionality. Each Game State's responses were tested with a test instance for GameStateManager. The system includes functons: onEnter, onExit, update, render and onKeyEvent. Each of these were tested, and could therefore ensure Game State interaction on a low level.

Graphical User Interface Integration
Another aspect we developed and tested early on was User Interface, and its representation of data within each state. Integration for GUI was easy to test, as in most cases the GUI was self-evidence for the data it was portraying. However, it was a large job integrating the GUI to work fluidly and reliable both internally with itself, and externally with other systems.

Module Data Communication

For higher level Game State testing, we tested the exchange of data to see if the specific game state implementations interact correctly and function as expected. Values such as player information and battle conditions were seen to be passed correctly, without fault. All pairs of game state transitions were tested and eventually proven to be successfully integrated. Prior to integration, each state was tested as a singly functioning module. Only when each Game State was confirmed to function (on a basic level), did we integrate them together using the aforementioned, and previously tested, management system. However, as we had not yet integrated networking, some network-managed data transitions could not be tested until the networking was integrated and tested. Some states' data flow was managed locally, in which case we were able to tested their data flow immediately.

Network Integration

Networking integration was implemented bit by bit, integrating with each of the game state modules individually and independently from each other. In fact, in terms of networked chat, we were able to test data transitions (player communications) independent of any Game State, due to the nature of the Chat Box representation in game. Each of the individual network implementations were integrated one by one and tested accordingly. Consequently, Game State transitions that required networked data (ie Game Settings from the Host's Main Menu state) could then be tested in the context of the new functions. One error we found whilst testing the game was during single player mode in the battle game state when two players were in their respected battle the networked though they were in the same battle so moves carried out.

Appendices

Appendix A – Test Cases

Test Case ID	Input	Expected Output	Output	Pass?	Date
1	Click 'Exit' button	Quits the game and closes window.	Window closes	Y	12/02/2015
2	Click 'Play' button	Takes you to Host/Join menu	Shows Host/Join menu	Y	12/02/2015
3	Click 'Host' button	Take you to Host menu	Shows Host menu	Y	12/02/2015
4	Click 'Join' button	Take you to Join menu	Shows Join Menu	Y	12/02/2015
5	Click 'Start' button in host menu.	Takes you to game lobby	Shows game lobby	Y	12/02/2015
5	Joins a team game lobby	Shows you have joined game and can select character	You are in a team an can choose character	Y	12/02/2015
6	Click 'Play' button in game lobby	Takes you to overworld, displaying a tutorial.	In game, showing the tutorial menu.	Y	13/02/2015
7	Press the 'Up', 'Right', 'Down' and 'Left' buttons on the keyboard in that order.	The character will move ' 'Up', 'Right', 'Down' then 'Left'.	Character move 'Up', 'Right', 'Down' then Left	Y	13/02/2015
8	Move around using direction buttons.	Pet that has been tamed follows one block behind.	Pet follows you.	Y	04/03/2015
9	When standing next to a chest, press 'f' on the keyboard.	The chest contents appear.	Chest contents appear	Y	16/02/2015
10	Click 'Take All' on the chest contents.	All items in chest will move to inventory.	All items from chest in in the inventory.	Y	16/02/2015
11	In Overworld press 'i'	The inventory menu opens	Inventory menu opens	Y	16/02/2015
12	In inventory menu drag sword item to sword slot.	Sword is accepted and attack stat goes up.	Attack goes up and sword is attached.	Y	16/02/2015
13	In inventory menu drag shield item to shield slot.	Shield is accepted and attack stat goes up.	Defence goes up and shield is attached.	Y	16/02/2015
14	In inventory menu drag armour item to armour slot.	Armour is accepted and attack stat goes up.	Defence goes up and armour is attached.	Y	16/02/2015
15	In inventory menu drag helmet item to helmet slot.	Helmet is accepted and attack stat goes up.	Defence goes up and helmet is attached.	Y	16/02/2015
16	In Overworld press 'h'	The skill tree menu opens	Skill tree menu opens.	Y	16/02/2015
17	Go up to quest character and press 'f'	A quest is given to you to complete.	Quest is given to complete.	Y	04/03/2015
18	Complete the given quest.	Experience is added when the quest is completed.	Experience added.	Y	04/03/2015