A8 Test Plan

Team 10 – Zombie Apocalypse
Chad Nelson, Cole Anagnost, Tas Fox, Tim Flanigan

Test Plan

Zombie Apocalypse

Chad Nelson
Cole Anagnost
Tasewell Fox
Tim Flanigan

1. TEST PLAN

1.1 SystemController

Author: *Chad Nelson*

Complexity:
The SystemController is not the most complex part of the system, but it must work perfectly because it is critical to the base architecture of the system.

Frequently Used Pieces:
The SystemController is perhaps the most frequently used component of the client system because it holds reference to all Screens and Services. Any screen that uses a Service must call SystemController.getInstance() in order to get the singleton instance of the SystemController (alternatively, they could call new SystemController() which would return the singleton instance). Another commonly used function of the SystemController is to animate the transition between Screen that occurs when a Screen calls animateScreenChange(...).

Test Plan:
The test plan we originally used involved only manual tests. These tests were organized into a spreadsheet. After a revision, we would rerun all the tests and add a column in the spreadsheet for that results of that test pass on the revision. In this way, we could see what tests we fixed and broke from revision to revision.

In an ideal world, all tests would be automated. There are two types of functions: functions that take inputs and return a value based on those inputs, and functions that access other parts of the system to manipulate data. Writing unit tests for input/output functions is a simple task, similar to Junit testing. To accurately do unit tests on functions access complex parts of the system, we would use ASmock, and open source project on SourceForge (link). ASmock would allow us to create mock objects, replacing components of the system, and allowing our unit tests to ensure that the function they're testing continue to call the correct functions across the system. Mocking objects also let's us separate unit tests from integration tests.

1.2 DataServer

Author: *Chad Nelson*

Complexity:
The DataServer is a multi-threaded process that must manage race conditions and appropriately handle cases when the user logs off.

Frequently Used Pieces:
Some of the most frequently used components of the server are function handlers for logging in and updating user variables (like cash, weapons, score, ammo, etc).

Test Plan:
The test plan we used was a set of very high level tests that manually confirmed the end

result of having a server in the background.

A better way to test the Server would be to use Junit tests. We could also mock out the socket connection(s) for the unit tests. To test the integration between the server and client, we could use tests that confirmed the interface between the client and server matched. These tests would give us a greater amount of information if and when something broke.

1.3 Sound Service

Author: *Tas Fox*

Complexity: This is complex because each user must have a dedicated sound channel so they can all play sounds. This means that at any moment several sounds can be played. These sounds include weapon fires, reloading, and weapon empty.

Frequently Used Pieces: Some of the most frequently used operations are local call and server call.

Test Plan: The test plan for the sound service was to do manual tests with logged output to make sure all sounds played as expected. Automated tests are not practical for this test because you need someone to listen to make sure the proper sounds are being played at the proper time.

1.4 Weapon Fire

Author: *Tas Fox*

Complexity: Every time a weapon fires its trajectory must be calculated, it must be animated, and it must be tested to see if it hits any zombies

Frequently Used Pieces: Firing functions, draw bullets, hit test

Test Plan: The test plan includes a mix of automated and manual tests. The automated tests will set strict conditions inside the game that guarantee a target will be hit with each test. The test will be repeated multiple times and the output will be logged using the system debugger.

1.5 BuyScreen

Author: *Tim Flanigan*

Complexity:
The buy screen is used to manage weapons and money that the player has access too. This is a large feature in the game and must work correctly to store player information and change it accordingly.

Frequently Used Pieces:
There are two main functions that are most frequently used in the BuyScreen class. The first is showWeaponStats, which is used to show weapon information when selected. The

second is buyWeapon, which purchases the selected weapon or buys ammunition.

Test Plan:
Our test plan for this screen was mainly manual testing. We would select all the weapons to make sure that all the stats and pricing on the weapon was correct. We would then try to purchase the weapon and make sure that it was possible, and that money was deducted correctly. We also made sure that when the weapon was bought, it was actually stored in the users account information for later use.

Although we did manual testing, automated testing would not have been hard to set up. We would need to set up a test to either select the weapons or buy them, and then check variables to make sure that they were adjusted correctly.

1.6 HighScoresScreen

Author: *Tim Flanigan*

Complexity:
The HighScoreScreen itself isn't too complicated, however, it makes a call to the server to retrieve the highscores for all users from the database, and then splits that result and puts it into a table format for easy viewing for the user. The complexity lay in the integration between the screen and the database on the server.

Frequently Used Pieces:
The HighScoresScreen has two main functions that are used. The first one is the init function, which makes the call to the database to retrieve the high score information pertaining to the user. The information is serialized and returned to the class as a string containing all the information. The second function used in this class is the loadScores function. This function Splits the serialized data recieved from the database that contains all the high score information, and puts it into a table widget.

Test Plan:
The testing for the HighScoresScreen was done manually. We had users play the game enough to get statistics for an account, and then viewed the highs core screen to make sure that the highscores were correctly displayed on to the screen.

To test the init function and make sure that the information received from the database was correct, it could just be printed to the screen to be manually check to make sure that all the correct information was included.

1.7 LevelService

Author: *Cole Anagnost*

Complexity:
The LevelService tracks all the variables about a game level including zombies, players, pickups, bullets, and bloodspatters.

Frequently Used Pieces:
The main functions used in the LevelService are the functions for remote updates (such as zombie and pickup positions) as well as the functions for collision detections (either with a boundary or with a pickup on the level).

Test Plan:
The testing for LevelService was done manually by setting specific variables and testing in single player or multi-player mode (depending on the function being tested).

Ideally these tests would have been automated to ensure that no bugs slipped through the manual testing phase.

1.8 ClientService

Author: *Cole Anagnost*

Complexity:
The ClientService holds the connection to the server. It is responsible for communicating with the server to send and receive player data (such as money, weapons, and ammo) and game synchronization data (such as player and zombie positions during multi-player games).

Frequently Used Pieces:
The main functions used in the ClientService are the methods that send and receive socket data. The socketData function receives a string from the server, and splits it into an opcode and a list of arguments delimited by a pipe character ("|", i.e. "LOGIN:username|password"). These are passed to the receiveData function, which calls the appropriate function based on the opcode and passes in the provided argument list. The sockSend function loops back messages with opcodes relating to gameplay updates (in both single and multi-player modes) and sends them to the server (only in multi-player).

Test Plan:
The testing for the ClientService was done manually by playing the game in both single and multi-player mode and examining the messages sent and received by the client.

Ideally the tests would be automated using ASmock to ensure that the correct functions are being called, and that the incoming data is parsed correctly.

2. EXAMPLE OF ORIGINAL TEST PLAN

The test plan that we used during the term was a simple document used to keep track of the results of manual tests. A partial example of the document is shown below:

	Date	10/22/2009	10/26/2009	11/1/2009
	Version	v0.102	v0.119	v0.132
TestID	**Description**			
1.1	**Prerequisites**			
1.1.1	Make sure you are using a Windows PC	Pass	Pass	Pass
1.1.2	Make sure Adobe Flash Player is installed	Pass	Pass	Pass
1.1.3	Make sure the server is running at chad.game-host.org	Pass	Pass	Pass
1.2	**Logging In**			
1.2.1	Create a new user:	Pass	Pass	Pass
	1. Start the program			
	2. Ensure the Login Screen is displayed			
	3. Enter a new unique username, password, and confirmation of the password in the create new user prompts			
	4. Click the create user button			
	5. You should be taken to the Main Menu and be logged in			
1.2.2	Receive an error when creating a new user with an old username:	Pass	Pass	Pass
	1. Start the program			
	2. Ensure the Login Screen is displayed			
	3. Enter an old username, password, and confirmation of the password in the create new user prompts			
	4. Ensure the screen displays an indication that the username is already taken			
1.2.3	Receive an error when creating a new user with unmatched passwords:	Pass	Pass	Pass
	1. Start the program			
	2. Ensure the Login Screen is displayed			
	3. Enter a new unique username, password, and confirmation of the password in the create new user prompts. Make sure the password and confirmation do not match			
	4. Ensure the screen displays an indication that the passwords do not match			
1.2.4	Login as an existing user:	Pass	Pass	Pass
	1. Start the program			
	2. Ensure the Login Screen is displayed			
	3. Enter your username and password			
	4. Click the login button			
	5. You should be taken to the Main Menu and be logged in			
1.2.5	Receive an error when logging in as an existing user with an unknown username	Pass	Pass	Pass
1.2.6	Receive an error when logging in as an existing user with the wrong password	Pass	Pass	Pass
1.2.7	Ensure that passwords are not displayed as clear text, and instead are show as *	Fail	Pass	Pass
1.3	**Main Menu**			
1.3.1	Ensure clicking the Single Player button takes you to the Level Selection Screen:	Pass	Pass	Pass
	1. Start the program and login			
	2. Ensure you are at the Main Menu			
	3. Click the button that says "Single Player"			
	4. Ensure you are taken to the Level Selection Screen			
1.3.2	Ensure clicking the Multiplayer button takes you to the Lobby Screen	Pass	Pass	Pass
1.3.3	Ensure clicking the High Scores button takes you to the High Scores Screen	Pass	Pass	Pass
1.3.4	Ensure clicking the Tutorial or Help button take you to the Tutorial Screen	Pass	Pass	Pass
1.4	**Level Select Screen**			
1.4.1	If you are a new user, ensure only the first level is available to you.	Pass	Pass	Pass
1.4.2	If you are an existing user, ensure that only the first n levels are available to play (where n is the number of completed levels + 1)	Fail	Fail	Pass
1.4.3	Ensure you are able to go back to the Main Menu by clicking a button	Pass	Pass	Pass
1.4.4	Ensure that you can go to the Buy Screen	Fail	Pass	Pass
1.4.5	Ensure that you can select a level and begin playing it in Single Player mode.	Pass	Pass	Pass
1.4.6	Ensure the weapon selection panel is visible	Fail	Pass	Pass
1.5	**Weapon Selection Panel**			
1.5.1	Ensure the weapon selection panel is visible in the Buy Screen and Level Select Screen	Fail	Pass	Pass
1.5.2	Ensure that all weapons you have purchased and own are displayed on the panel.	Fail	Pass	Pass
1.5.3	Ensure that the ammo amounts listed are accurate	Fail	Pass	Pass
1.5.4	Ensure that you may select one primary and one secondary weapon	Fail	Pass	Pass
1.5.5	Ensure that you start the level with the weapons that you have selected	Fail	Pass	Pass
1.6	**Buy Screen**			
1.6.1	Ensure the "Main Menu" button takes you back to the Main Menu	Pass	Pass	Pass
1.6.2	Ensure there is a continue button that's text changes	Pass	Pass	Pass
1.6.3	The continue button will read "Retry" if you just failed a level and will let you retry the level	Pass	Pass	Pass
1.6.4	The continue button will read "Next Level" if you just completed a level	Pass	Pass	Pass
1.6.5	The continue button will read "Go back" if you just came from the Level Select Screen	Pass	Pass	Pass
1.6.6	Ensure that you are able to click on each weapon in the store, and that relevant stats are displayed	Fail	Pass	Pass
1.6.7	Ensure you may only buy weapons for which you have enough money	Fail	Pass	Pass
1.6.7	If you own a gun, then the Buy button shall read "Buy Ammo" and the price will be less than the gun	Fail	Pass	Pass
1.7	**High Scores Screen**			
1.7.1	Ensure that a ranked and ordered list of users with the highest score is displayed	Fail	Fail	Pass
1.7.2	Ensure a back button exists to go back to the main menu	Fail	Pass	Pass
1.8	**Tutorial Screen**			
1.8.1	Ensure accurate directions for how to play the game are displayed	Skip	Skip	Pass
1.8.2	Ensure a back button exists to go back to the main menu	Skip	Skip	Pass