A4 SRS-2

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

Team Work Distribution Form

Assignment A4 - Group #10

Name	% of Effort	Extra Work	Description of Work
Chad Nelson	27	Yes	Section 3.1.4, Minutes, Put Together
Cole Anagnost	24		
Tas Fox	25	Yes	Sections 3.3 and 3.4
Tim Flannigan	24		

Meeting Minutes

September 28

Name	Present	Late > 5 Min	Informed of Absence?	Scribe?
Chad Nelson	Yes	No	-	Yes
Cole Anagnost	Yes	Yes	*	-
Tas Fox	Yes	Yes	*	-
Tim Flannigan	No	No	Yes - sick	-

* Lack of communication that we were indeed having our weekly meeting.

Name	Old Action Item	Status	New Action Item	Due Date
Chad Nelson	A3 SRS-1	Done	Implement Server	10/5
Cole Anagnost	A3 SRS-1	Done	Collect Weapon Stats	10/12
Tas Fox	A3 SRS-1	Done	Implement firing/hit function	10/5
Tim Flannigan	A3 SRS-1	Done	*(sick)*	N/A

Meeting Minutes

October 5

Name	Present	Late > 5 Min	Informed of Absence?	Scribe?
Chad Nelson	Yes	No	-	Yes
Cole Anagnost	Yes	No	-	-
Tas Fox	Yes	No	-	-
Tim Flannigan	Yes	No	-	-

Name	Old Action Item	Status	New Action Item	Due Date
Chad Nelson	Implement Server	Done	Create Lobby/Login Screen	10/12
Cole Anagnost	Collect Weapon Stats	50%	Previous + Implement	10/12
Tas Fox	Implement firing function	Done	Zombie health/death/ammo	10/12
Tim Flannigan	*(sick)*	N/A	Level/Buy/Score/Help .swc/.fla	10/12

Meeting Minutes

October 12

Name	Present	Late > 5 Min	Informed of Absence?	Scribe?
Chad Nelson	Yes	No	-	Yes
Cole Anagnost	Yes	No	-	-
Tas Fox	Yes	No	-	-
Tim Flannigan	Yes	No	-	-

Name	Old Action Item	Status	New Action Item	Due Date
Chad Nelson	Lobby/Login Screen	Done	A4 SRS-2	10/19
Cole Anagnost	Weapons	Done	A4 SRS-2	10/19
Tas Fox	Zombie health/ammo	Done	A4 SRS-2	10/19
Tim Flannigan	Level/Buy/Score/Help .swc	50%	A4 SRS-2	10/19

Meeting Minutes

October 19

Name	Present	Late > 5 Min	Informed of Absence?	Scribe?
Chad Nelson	Yes	No	-	Yes
Cole Anagnost	Yes	No	-	-
Tas Fox	Yes	No	-	-
Tim Flannigan	Yes	No	-	-

Name	Old Action Item	Status	New Action Item	Due Date
Chad Nelson	A4 SRS-2	Done	Other items on code goals doc	10/26
Cole Anagnost	A4 SRS-2	Done	Pickup Images &Room/Userlist	10/26
Tas Fox	A4 SRS-2	Done	Collisions	10/26
Tim Flannigan	A4 SRS-2	Done	Game Room	10/26

Agenda

Discussed progress on coding goals

Assigned new coding goals

Added sound to game

Worked on A4 SRS-2

Scheduled a paraprogramming session for Sunday at Chad’s place.

Software Requirements Specification

Zombie Apocalypse

Chad Nelson
Cole Anagnost
Tasewell Fox
Tim Flanigan

TABLE OF CONTENTS

REVISION HISTORY

Version	Date	Author	Change
0.1	9/19/2009	Group 10	Initial Document
0.2	9/20/2009	Group 10	A3 SRS-1
0.3	10/20/2009	Group 10	A4 SRS-2

1. INTRODUCTION

1.1 PURPOSE

The purpose of this Software Requirements Specification document is to describe the architecture and design of the Zombie Apocalypse game. The intended audience includes the project's developers and the Professor, TAs, and students of the ComS 309 class for Fall 2009.

1.2 SCOPE

The scope of this document is the low-level architecture and intended design of the Zombie Apocalypse game. This document should give a detailed outline of the goals and expected functionality of the Zombie Apocalypse game.

The scope of this initial document does not include technical specifications intended for development.

1.3 DEFINITIONS, ACRONYMS, ABBREVIATIONS

Term	Description
Alpha	Transparency
AS3	(ActionScript 3)The programming language used with Adobe Flash
Chat Room	A collection of client connections to the server that are grouped together for the purpose of sharing text messages; the lobby and all game rooms are chat rooms
Client	(Actor) An instance of the game run in a web browser; there are many clients
DisplayObject	An interface that any visible item implements
Display Tree	A tree containing a root node and it's children; any visible object must be attached to the display tree before it is visible on the stage
Game Room	A chat room where all players have the intent of playing a multiplayer game together; limited number of players; there can be any number of game rooms
Lobby	The default chat room where players are allowed to create/join game rooms; unlimited number of players
MovieClip	An AS3 object that implements the DisplayObject interface; contains a number of frames that are animated
PC	(Player Character) The player's sprite
Player	(Actor) A person that plays the game by running an instance of the client
Server	(Actor) A program that manages connections and stores information. Manages room creation and multiplayer communication, and stores data about player accounts, high scores, and game status.
Sprite	An AS3 object that implements the DisplayObject interface; contains a single static image
Stage	A static AS3 object that represents the view of all visible objects

*.SWC	(Shockwave Flash Component extension) Contains code and graphics that can be instantiated by another Flash program
*.SWF	(Shockwave Flash extension) A Flash movie file; Contains compiled byte code and graphics that will be interpreted by a Flash Player and displayed on a webpage

1.4 DESIGN GOALS

1. Usability - The game must provide an intuitive player interface and control scheme, in addition to help or tips for an inexperienced player. The game should also remain responsive during intensive processing for many onscreen opponents. This design goal is met by implementing a widely used control scheme (WASD keyboard controls for player movement with QER for other actions, mouse for aiming and firing) with all necessary information shown in a GUI overlay.
2. Multiplayer - The game must allow multiple players to play on the same map in real time. This design goal will be met using a client-server architecture, where central servers store the player's data and relay information between client machines.

1.5 REFERENCES

[None]

1.6 OVERVIEW

[Omit]

2. OVERALL DESCRIPTION

Zombie Apocalypse is a browser-based, multiplayer, shooter game. It has many single player levels, and gives the players the ability to shoot zombies cooperatively with multiplayer. This feature includes being able to chat with other players. Players earn cash by shooting zombies, and can purchase weapons and other items with the money.

2.1 PRODUCT PERSPECTIVE

Zombie are a fairly common subject in mainstream video game culture. Zombie Apocalypse will differentiate itself by offering flash-based multi-player and persistent stat tracking. This will allow casual gamers and hardcore gamers alike to quickly pick-up our game and play it.

2.1.1 Concept of Operations

Zombie Apocalypse is developed in two different environments. The client is web based, written in ActionScript 3, and displayed in the browser using a Flash Player. The server is written in Python.

When a player loads an instance of the client in his web browser, it automatically creates a socket connection to the server. After logging in, the client fetches data about the player from the server and then displays the main menu interface. The user can choose either single player of multiplayer. In single player mode, the player begins a level and shoot zombies until the level is complete (or he dies). After a level is beaten, statistics are transferred to the server for storage. In multiplayer mode, the player is able to take advantage of the socket connection by chatting with other players connected to the server, and using to server the enter the same level with multiple players at the same time.

2.2.2 Major User Interfaces

Please see ***Appendix A*** for screenflow and interface mock-ups.

2.1.2.1 Example Screenshots and Descriptions

Please see ***Appendix A-2*** for screen shots and descriptions.

2.1.3 Hardware Interfaces

The player will interface with the client using the keyboard and mouse. Specifically:

Keyboard Controls:

- W - up
- A - left
- S - down
- D - right
- R - reload
- E - switch weapon
- Q - use powerup (if applicable)

- H - display help menu
- P - pause (single player only)

Mouse Controls:
- Click - Shoot

2.1.4 Software Interfaces

[Omit]

2.1.5 Communication Interfaces

[Omit]

2.1.6 Memory Constraints

[Omit]

2.1.7 Operations

[Omit]

2.1.8 Site Adaptation Requirements

[Omit]

2.2 PRODUCT FUNCTIONS

2.2.1.a Use Case 1 (Text)

Name: The player wants to log on to the game.
Author: Chad Nelson
Description: The player interacts with the client to provide credentials, the client then contacts the server to validate these credentials.
Actors: Player, Client, Server
Primary Flow:
Happy Path
1. The player enters his playername into the client
2. The player enters his password into the client
3. The player indicates he wants to log on with these credentials
4. The client confirms the credentials with the server; they are valid
5. The client downloads player data from the server and starts the game

Alternative Flows:
Invalid playername
1. The player enters an incorrect playername into the client
2. The player enters his password into the client
3. The player indicates he wants to log on with these credentials
4. The client confirms the credentials with the server; they are invalid
5. The client displays an error message and the player is allowed to try again

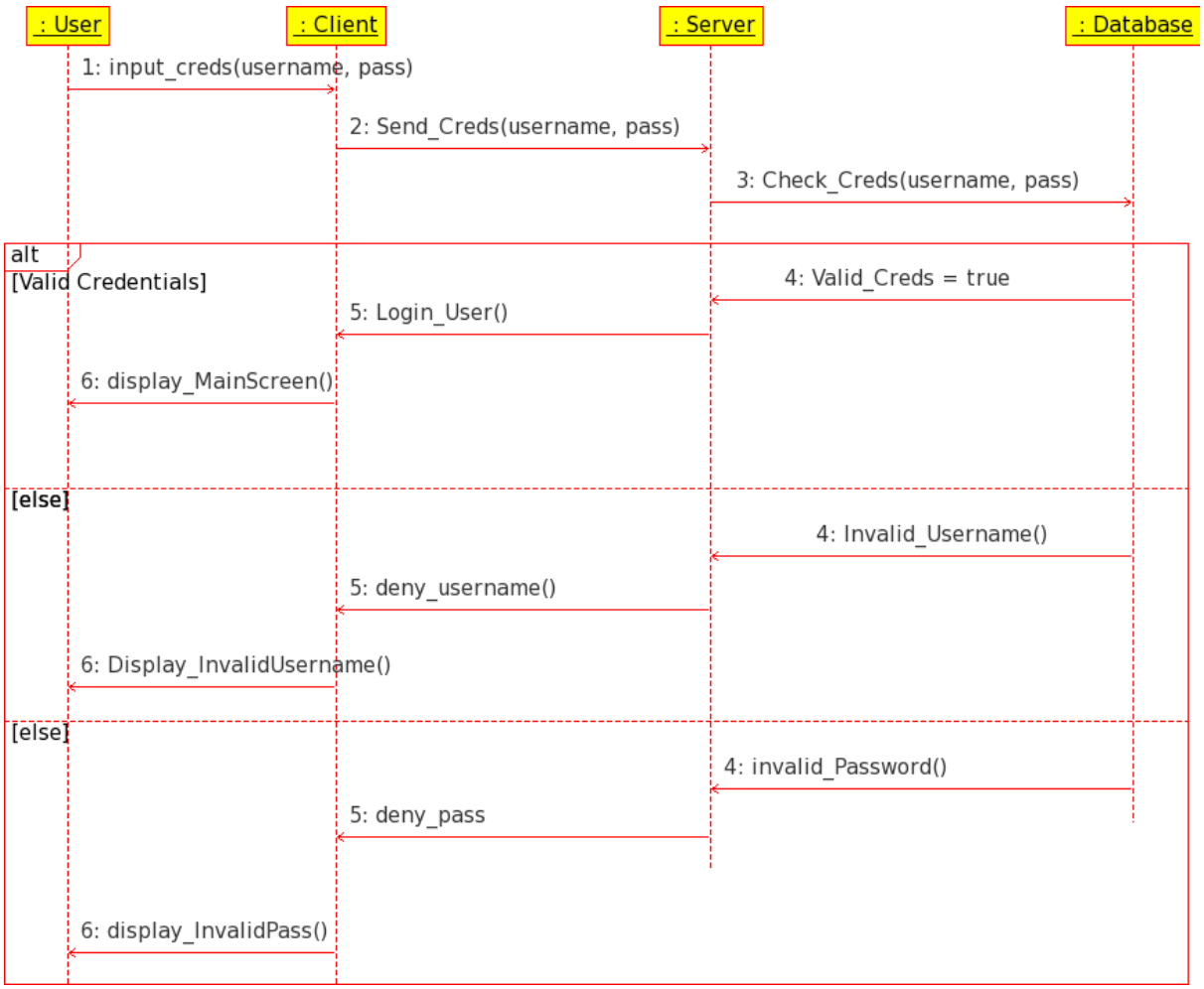
Invalid Password
1. The player enters his playername into the client
2. The player enters an incorrect password into the client
3. The player indicates he wants to log on with these credentials
4. The client confirms the credentials with the server; they are invalid

5. The client displays an error message and the player is allowed to try again

Preconditions: A player has instantiated an instance of the client.
Postconditions: The player is now logged in, and is presented with the main menu.

2.2.1.b Use Case 1 (Diagram)

UML Sequence Diagram for Use Case 2.2.1



2.2.2.a Use Case 2 (Text)

Name: The player wants to play the game with a friend.
Author: Chad Nelson
Description: The player wants to initialize a multiplayer game with another player or friend. The goal of this use case is to allow the player to have a straight forward process for communication and game creation.
Actors: Players, Client, Server
Primary Flow:
Happy Path:

1. The player selects "Multi-player Game" from the main menu.
2. The player is then taken to the game lobby and chat system.
3. The game lobby and chat system presents the player with a list of available games and other players currently logged into the chat system.
4. The player finds the person he/she wishes to play with and initiates communication using the chat system.
5. The player creates a multiplayer game.
6. The other players join the game

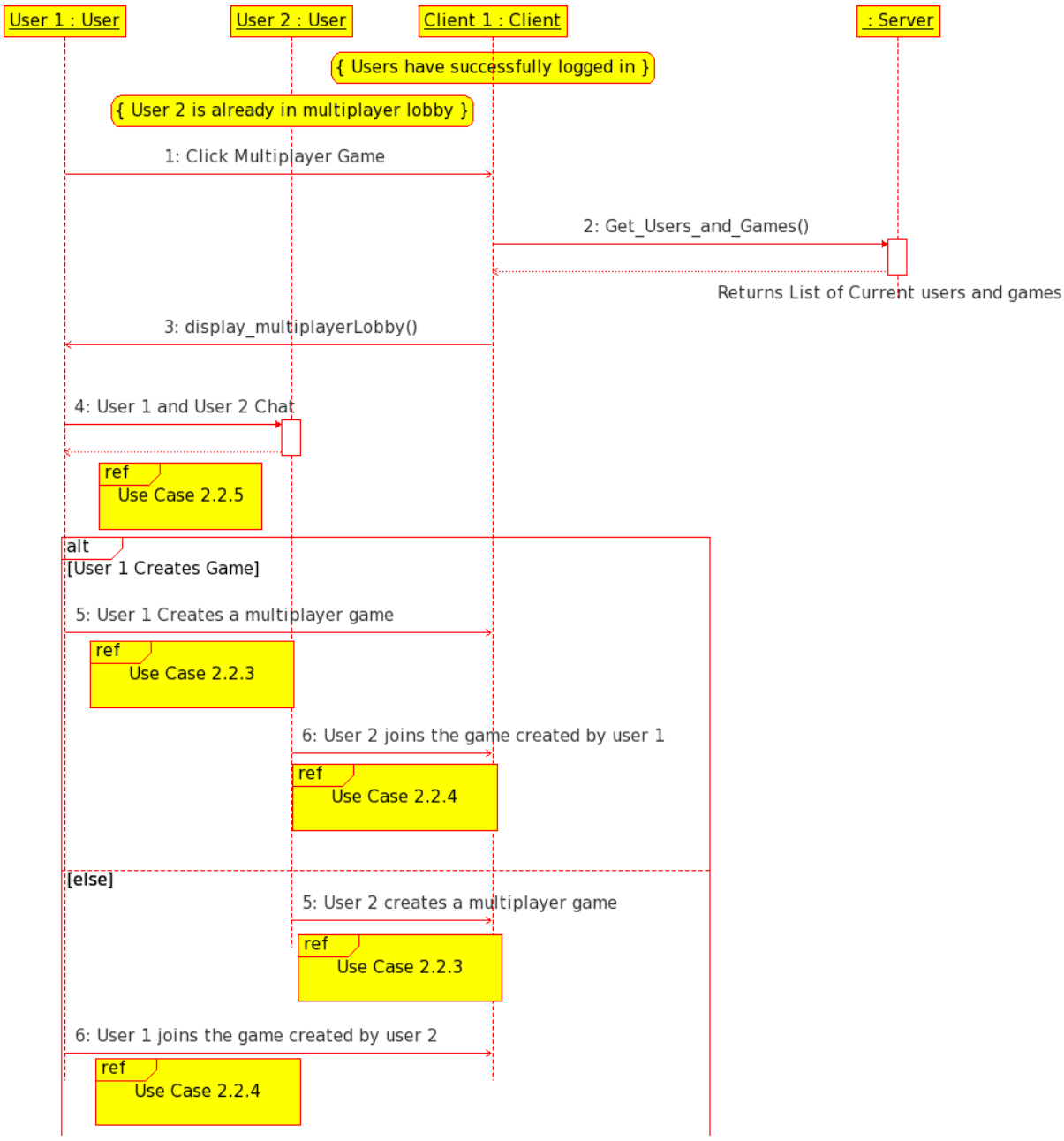
Alternative Flows:
Join Game:

1. Steps 1-4 from *Happy Path*
2. The player joins another players multiplayer game.

Preconditions: The player has successfully logged into the game system
Postconditions: The player has communicated to another player and is now in a game.

2.2.2.b Use Case 2 (Diagram)

UML Sequence Diagram for Use Case 2.2.2



2.2.3.a Use Case 3 (Text)

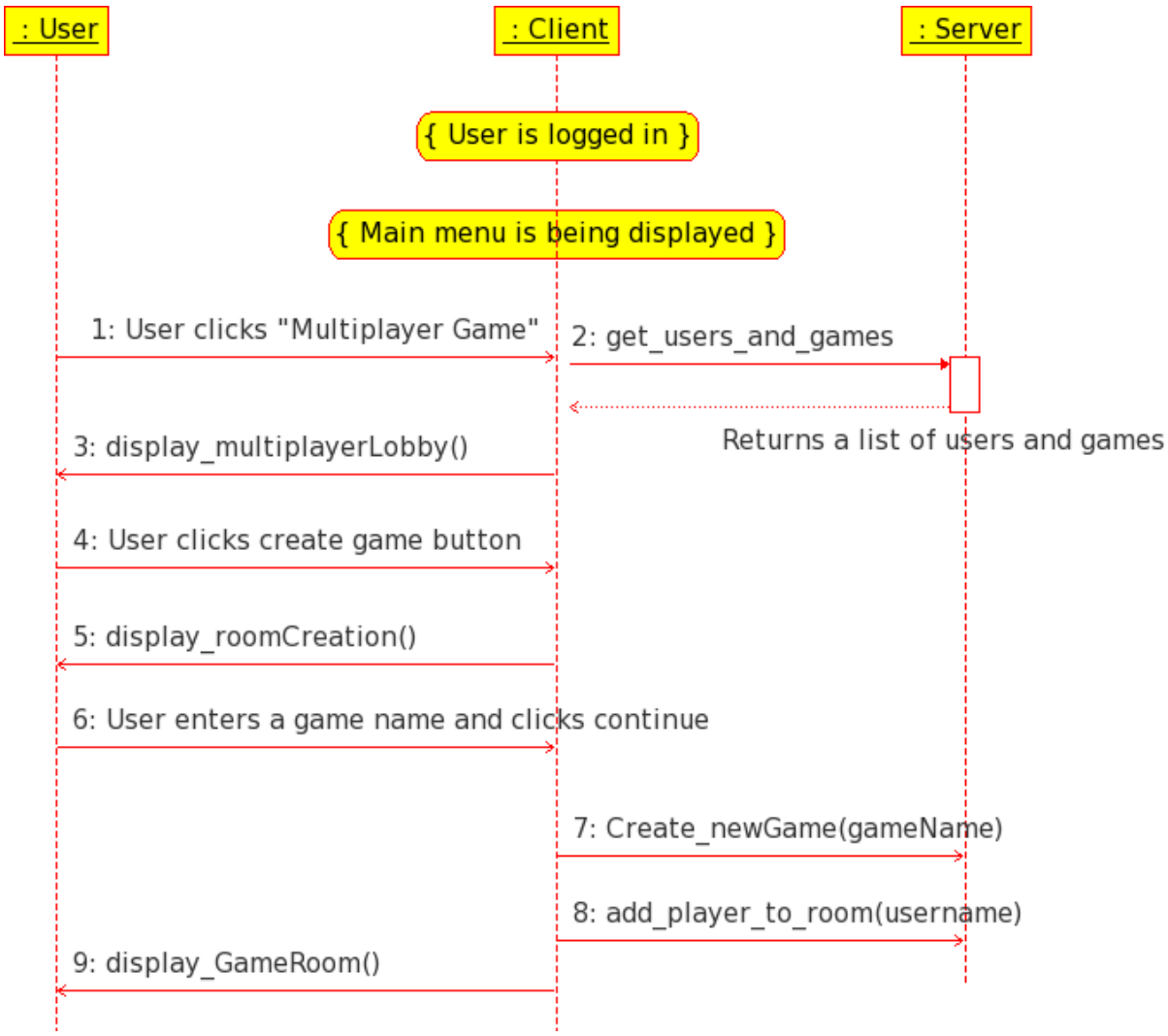
Name: The player wants to create a multiplayer game.
Author: Tim Flanigan
Description: The player interacts with the client to start a game, and the client communicates with the server to display the game in other clients.
Actors: Player, Client, Server
Primary Flow:
Happy Path:

1. The player selects "Multi-player Game" from the main menu
2. The player selects "Create Game" from the multi-player lobby menu
3. The player enters a name for the game room
4. The client contacts the server and adds the new game room to the list of open rooms
5. The client takes the player to the game room as the host, where they can change the game settings

Alternative Flows:
Preconditions: The player has logged into the client. Main menu is currently being displayed.
Postconditions: The player is in the game room as the host, waiting for other players to join.

2.2.3.b Use Case 3 (Diagram)

UML Sequence Diagram for Use Case 2.2.3



2.2.4.a Use Case 4 (Text)

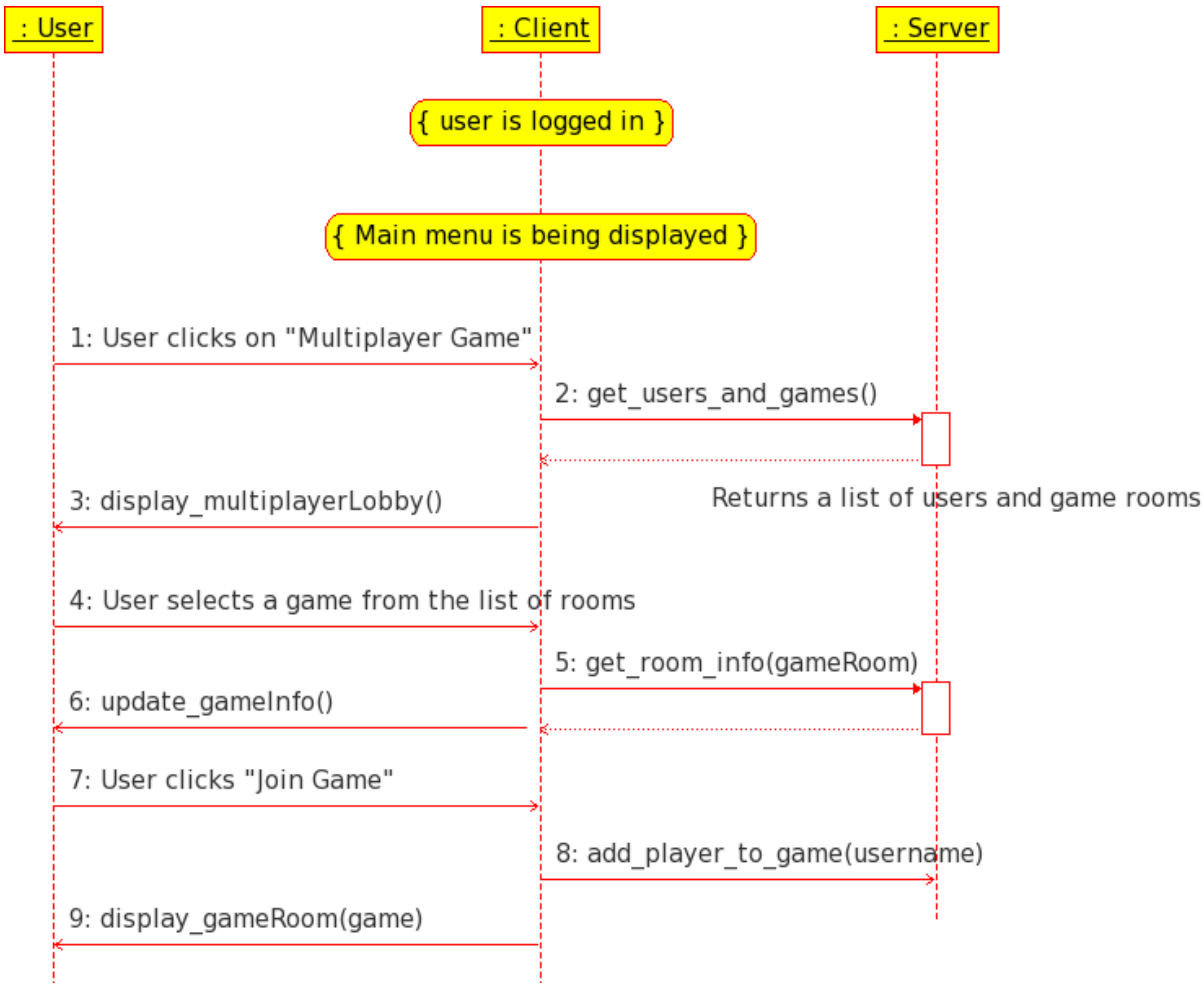
Name: The player wants to join a multiplayer game.
Author: Tim Flanigan
Description: The player interacts with the client to select a game room to join, and the client retrieves game information from the server.
Actors: Player, Client, Server
Primary Flow:
Happy Path

1. The player selects "Multi-player Game" from the main menu
2. The client gets a list of currently open game rooms from the server and displays them to the player
3. The player selects a game from the list
4. The client gets the game information from the server and takes the player to the game room

Alternative Flows:
Preconditions: The player has logged into the client. Main menu is being displayed.
Postconditions: The player is in the game room.

2.2.4.b Use Case 4 (Diagram)

UML Sequence Diagram for Use Case 2.2.4



2.2.5.a Use Case 5 (Text)

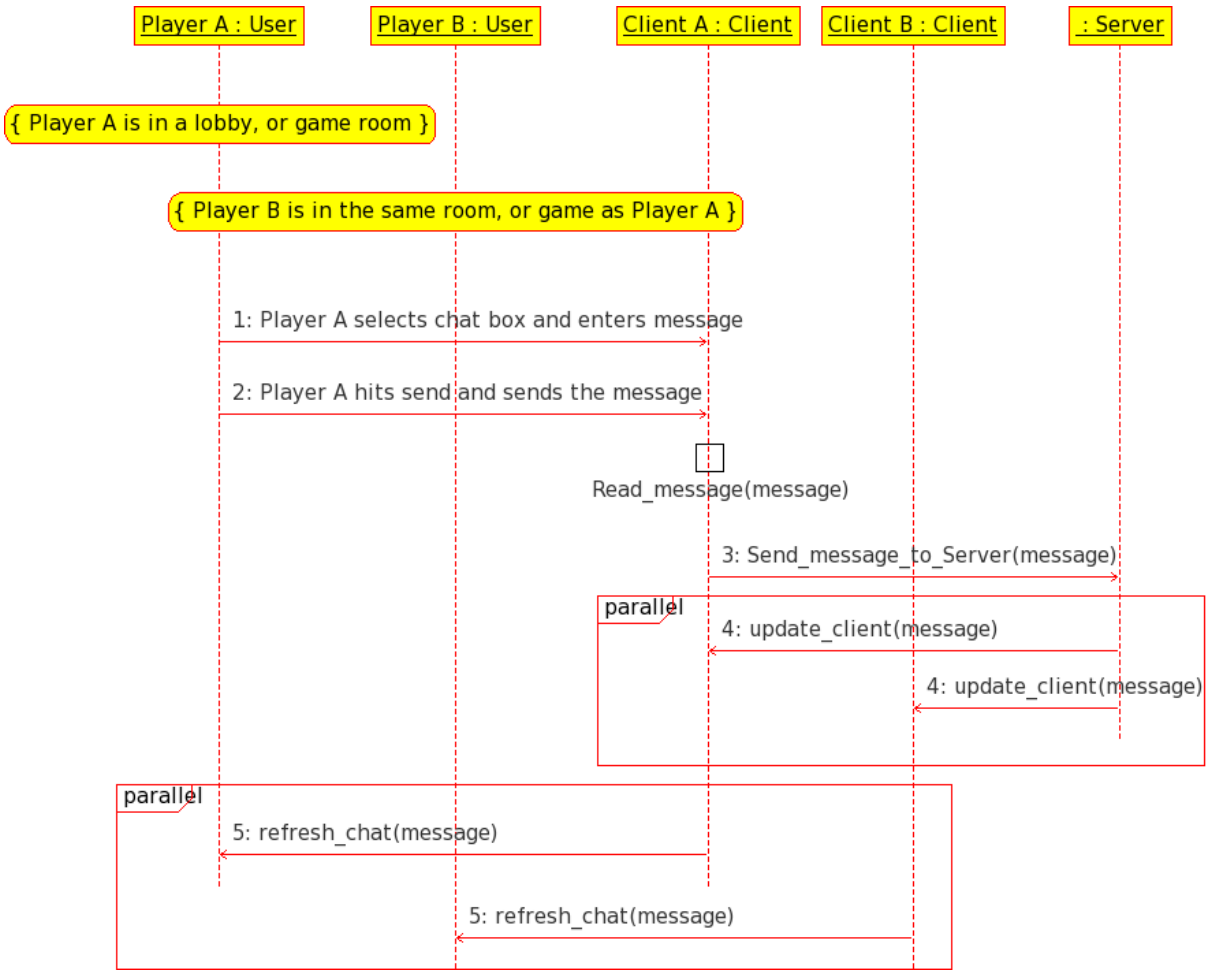
Name: The player wants to chat with other players.
Author: Tasewell Fox
Description: The player wants to communicate with other players while in the lobby, a game room, or while playing a multiplayer game.
Actors: PlayerA, PlayerB, Client, Server
Primary Flow:
Happy Path

1. PlayerA selects the chat box on the screen of the client
2. PlayerA types a message
3. PlayerA indicates he wants to submit his message
4. The client reads the message and sends it to the server
5. The server forwards the message to the other clients in PlayerA's chat room

Alternative Flows:
Preconditions: PlayerA is already in either the lobby, a game room, or a multiplayer game. PlayerB is in the same chat area (lobby or game room) as PlayerA.
Postconditions: PlayerB receives PlayerA's chat and it is displayed on his screen.

2.2.5.b Use Case 5 (Diagram)

UML Sequence Diagram for Use Case 2.2.5



2.2.6.a Use Case 6 (Text)

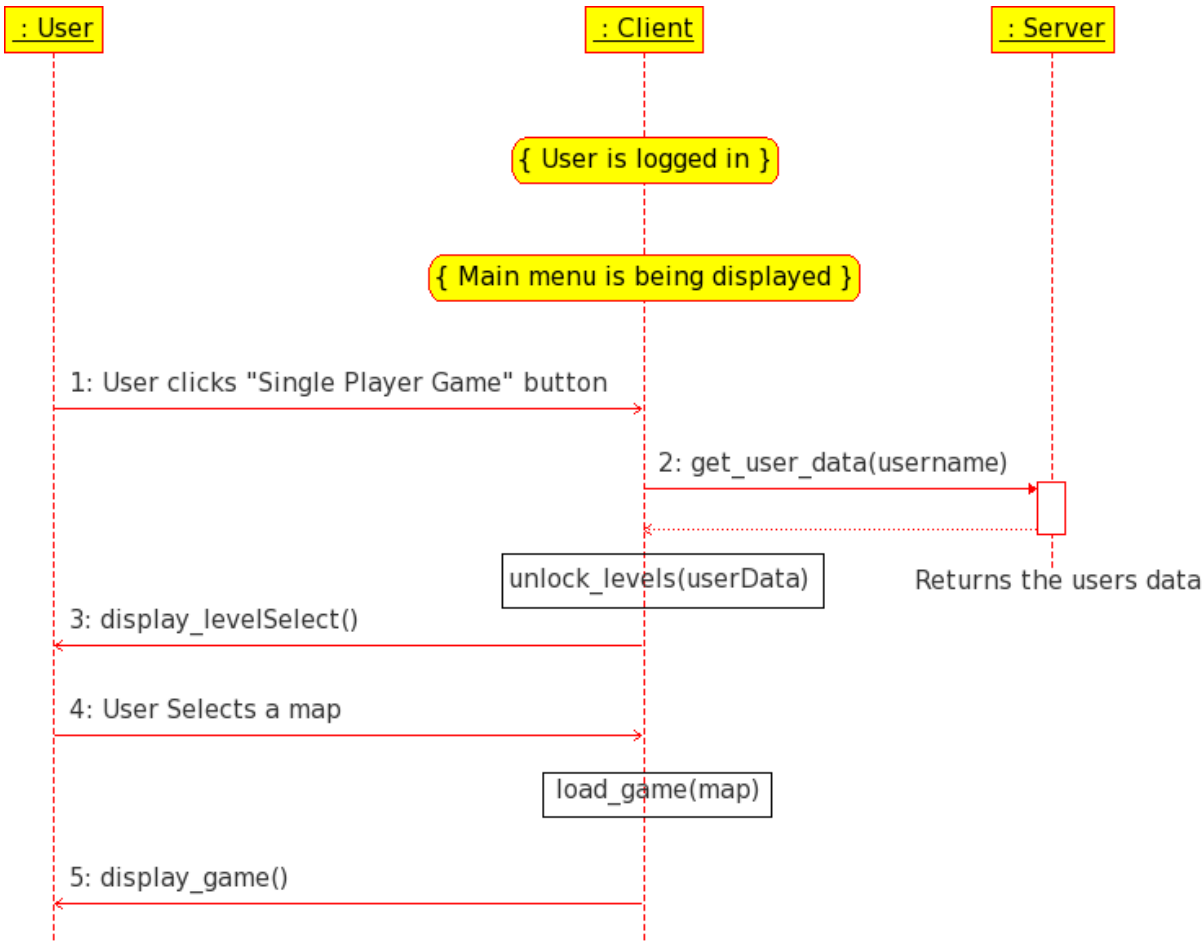
Name: The player wants to play the game alone.
Author: Tasewell Fox
Description: The players interacts with the client to start a single player game..
Actors: Player, Client, Server
Primary Flow:
Happy Path

1. The player selects "Single Player Game" from the main menu
2. The client checks the player's data, and presents the player with the maps he can play
3. The player selects a map
4. The client loads the map data and starts the game

Alternative Flows:
Preconditions: The player has logged in to the client and his player data has been downloaded.
Postconditions: The player is playing the game.

2.2.6.b Use Case 6 (Diagram)

UML Sequence Diagram for Use Case 2.2.6



2.2.7.a Use Case 7 (Text)

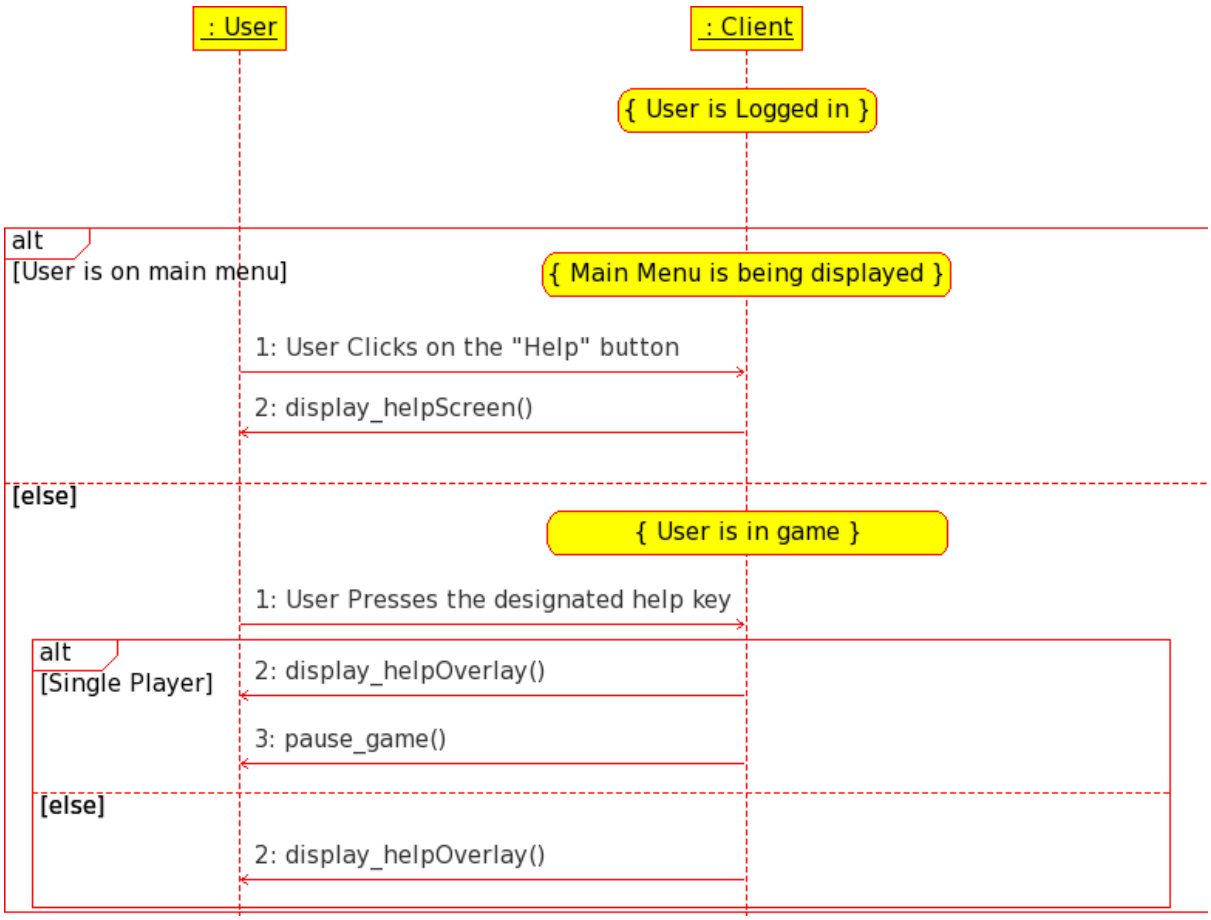
Name: The player wants to know how to play the game.
Author: Cole Anagnost
Description: The player interacts with the client to show the help screen.
Actors: Player, Client
Primary Flow:
Happy Path
1. The player selects "Help" from the main menu
2. The client displays the help screen

Alternative Flows:
During gameplay
1. The player presses a key ('H' for example)
2. The client displays the help screen (and pauses gameplay if single player)

Preconditions: The player has logged in to the client.
Postconditions: The player is at the help screen.

2.2.7.b Use Case 7 (Diagram)

UML Sequence Diagram for Use Case 2.2.7



2.2.8.a Use Case 8 (Text)

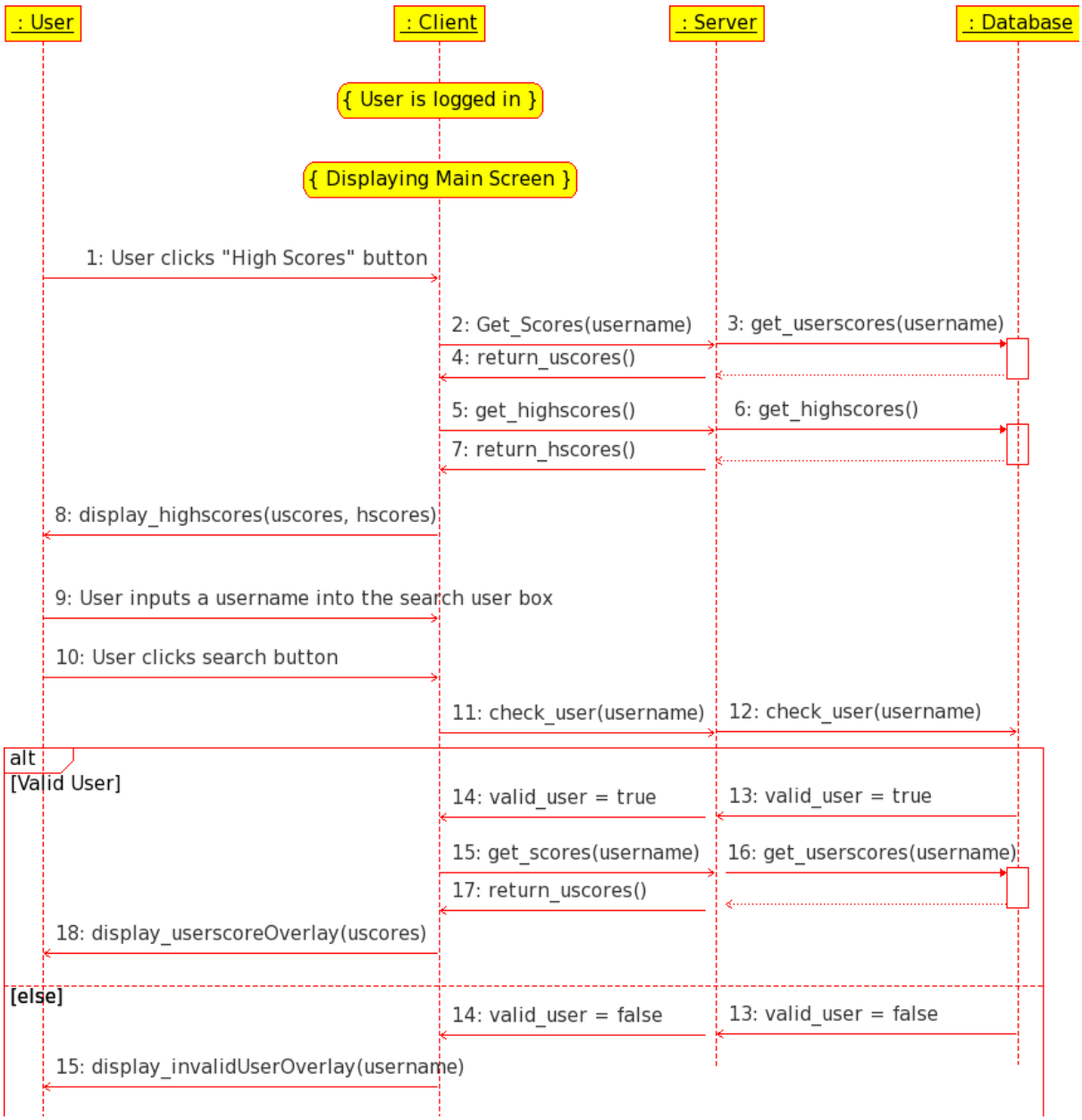
Name: The player wants to know statistics about other players / themselves.
Author: Cole Anagnost
Description: The player wants to know about their current statistics or the statistics of other players. The goal of this is to allow the player to track their stats over time to see improvement in their scores, and vie for the opportunity to reach a high score list. It also allows them to gauge the performance of other players to help with matchmaking decisions.
Actors: Player, Client, Server, Database
Primary Flow:
Happy Path:

1. The player clicks on the "High Scores" button on the main menu
2. Server sends a request to the database for the current list of high scores
3. The server displays the list of high scores to the client
4. The player is presented with a window containing the high scores and their current high scores.
5. The player can input another players name into the "Search player" box and click "Search"
6. The server sends a request to the database for that players high score
7. The player can then view that players score

Alternative Flows:
Invalid player:
If the player enters an invalid name at step 5 of the *Happy Path* then the player will be presented with an error message stating that the player is invalid.
Preconditions: The player is logged onto the game and on the main screen
Postconditions: The player is presented with a list of high scores

2.2.8.b Use Case 8 (Diagram)

UML Sequence Diagram for Use Case 2.2.8



2.3 USER CHARACTERISTICS

Our typical user will be a casual gamer who wants to play a game that is easy to pick-up play for a short time and then put down. There will also be the more hardcore users who will want to play for several rounds and keep track of their statistics and the statistics of their friends.

2.4 CONSTRAINTS

The size of the executable will be less than 25 MB.

2.5 ASSUMPTIONS AND DEPENDENCIES

The minimum requirements for a machine to run as a client will be:

- 1.6 GHz CPU
- 1 GB of memory
- At least 50 MB of free hard disk space
- A mouse and keyboard
- A sound card and speakers
- An internet connection of greater than 128 KB/s.

3. SPECIFIC REQUIREMENTS

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 User Interfaces

The requirements for the user interface are outlined and detailed in the screenshot mock-ups and screen flow diagram. Please see appendix A for these documents.

3.1.2 Hardware Interfaces

[None]

3.1.3 Software Interfaces

[None]

3.1.4 Communications Interfaces

The main communications interface is the boundary between client and server. Although the level of detail covered below is more fit for a design document, we list it here for transparency.

Our server will always be running and listening for incoming socket connections on the following DNS and port:

Host = chad.game-host.org
Port = 25432

Once a socket connection has been opened, UTF string messages will be sent between the client and server based on the criteria outlined in this section.

Messages are formatted by having an operation string followed by a colon, with any parameters following the opcode. Multiple parameters will be delimited by a single "|" character.

The format to describe our communications interface in the below scenarios is
>> Opcode: - text description of the opcode (param1, param2, ...)
where ">>" indicates a message the client sends to the server and "<<" indicates a message a message sent from the server to the client. "><" indicates a message that is sent both ways.

3.1.4.1 Login Screen

3.1.4.1.1 Logging In

>> LOGIN: - login (username, password)
<< LOGIN: - login succeeded (userdata)
OR
<< LOGINFAILED: - login failed (reason)

The client will send a string to the server (Ex. "LOGIN:zombieKiller|myPassword"). The server will then query its database to see if the username/password combo exists. If it does, it will send a message to the client containing the player's userdata and associates the active socket connection with that user. Otherwise, it will send an error message.

3.1.4.1.2 Check New Username Availability

>> CHECKUSER: - Check username (username)
<< USERAVAILABLE: - Pass, username is available (username)
<< No response for failure

This function is used on the login screen to check if a username is available to be used as a new user. If another player has already registered a user name, the server will not respond.

3.1.4.1.3 Create New User

>> CREATEUSER: - Create User (username, password)
<< LOGIN: - login succeeded (userdata)
<< CREATEUSERFAILED: - Failed to create user (reason)

Creates a new user and logs in with the newly created user.

3.1.4.2 Lobby Screen

3.1.4.2.1 Room creation

>> JOINROOM - Joins/creates the room with the name (roomname)
<< USERLIST - Memberlist (user1|user2|user3...|)
<< ROOMLIST - Roomlist (room1|room2|room3...|) *If the room being joined is "Lobby"
<< USERJOIN - A user has joined the room (username) *Success, sent to the room being joined
<< USERLEFT - A user has left the room (username) *To other users in the room the client left

Joins a user to a room. Users join rooms so they may receive CHAT messages and multiplayer commands from other users in the room.

When a user joins a room, they receive a list of all users in the room, and a USERJOIN message with their username. If they are joining the Lobby, they also receive a room list.

3.1.4.2.2 User Presence Notification

<< USERJOIN - A user has joined the room (username)
<< USERLEFT - A user has left the room (username)

These commands are sent to users in a room to alert them of users leaving or joining their room.

3.1.4.2.2 Room Presence Notification

<< CREATEROOM - Confirmation of creation *Sent to users in the Lobby
<< DELETEROOM - Last user has left a room (roomname) *Sent to users in the Lobby

These commands are sent to users of the Lobby to indicate when rooms are created or destroyed so their room lists may be updated.

3.1.4.2.3 Chat Messages

>< CHAT - A message from another user or an echo from current user (message)

Used to send user created messages to other players in a room.

3.1.4.3 High Scores Screen

>> GETSCORES - Get data for high scores screen
<< HIGHSCORES - List of data for high scores screen (row1, row2 ...) **

** Format is: HIGHSCORES:1,bobdylan,$25000,6|2,chadwick,$24000,5|...

This command is used to retrieve a list of high scores from the server.

3.1.4.4 Update User Variables

>> UPDATEUSERVAR - Update a user variable in the database (variable name, value)

This function is called after a user completes or fails a level. It would be used to update ammo counts and player statistics so that their user data is persistent over multiple sessions.

3.1.4.5 Multiplayer Communication

Multiplayer communications occur between users in a room. Each message sent to the server is used to update a shared object, and it is broadcast to all members of the room.

3.1.4.5.1 Setup Commands

>< SM: sets the map (level index)
>< SW: sets the weapons to be used (level index)
>< SD: sets the difficulty of the zombies (difficulty level)
<< SPL: player location (username, x coordinate, y coordinate)
<< SZL: zombie location (zombie index, x coordinate, y coordinate)

Setup commands are sent to each client so they may build the initial multiplayer game. Setup commands that are sent by a client can only be sent by a room's creator.

3.1.4.5.1 Action Commands

>< MKD: a user has pressed a key down (username, keypressed)
>< MKU: a user has pressed a key up (username, keyreleased)
<< MZD: deletes a zombie from the map (zombie index)
<< MZM: adds a movement command for a zombie (zombie index, params ...)

These messages are sent during a multiplayer game to update the state of the game.

3.2 FEATURES

Features are divided into eight subsections:

- 3.2.1 Dynamically Update GUI During Gameplay
- 3.2.2 Stat Tracking
- 3.2.3 Single Player Gameplay
- 3.2.4 Multiplayer Gameplay
- 3.2.5 Match Making
- 3.2.6 Persistent User
- 3.2.7 Online Chat
- 3.2.8 Unlockable Equipment and Levels

3.2.1 Dynamically Update GUI During Gameplay

Author: Chad Nelson

The client will dynamically update user variables during gameplay and display it in a widget in the upper right hand corner of the screen. The following requirements pertain only to Single Player and Multiplayer game modes, and not any navigation screen.

3.2.1.1 Money

3.2.1.1.1 The amount of money a player has shall be displayed in the upper right hand corner. (Mandatory)
3.2.1.1.2 Money shall increase when a player kills a zombie and retrieves a cash pickup. (Mandatory)
3.2.1.1.3 Money shall decrease when a player spends money in the buy screen. (Mandatory)
3.2.1.1.4 The amount of money a player has will be synchronized with the server. (Mandatory)

3.2.1.2 Health

3.2.1.2.1 A player's health shall be represented as a red bar located below the cash display in the upper right hand corner. (Mandatory)
3.2.1.2.2 The health bar shall act like a progress bar, with gray progress spreading from the left of the red bar to the right as the player looses health. (Mandatory)

3.2.1.3 Weapons

3.2.1.3.1 The currently selected weapon will appear in both graphical and textual form in the upper right hand corner. (Mandatory)
3.2.1.3.2 The user will be able to select at most two weapons (minimum one) to take into a level; one primary weapon and one secondary weapon.

3.2.1.4 Ammo

3.2.1.4.1 The amount of ammo left before the user must reload with the currently selected gun shall be displayed in the upper right hand corner of the screen. (Mandatory)
3.2.1.4.2 The amount of ammo the user is carrying for the currently selected gun shall be displayed in the upper right hand corner of the screen. (Mandatory)

3.2.2 Stat Tracking

Author: Tas Fox

The system has to be able to track the stats of the user. The feature groups of the stat tracking subsystem are:

3.2.2.1 Track number of zombies killed (Mandatory)

3.2.2.1.1 The system shall keep a running count of all zombies killed by the player.
3.2.2.1.2 The system shall record the kill count to the players user account data and store it on the database.
3.2.2.1.3 The client shall synchronize the kill count with the server after every level.
3.2.2.1.4 The client shall synchronize the kill count with the server if the kill count has risen by ten.

3.2.2.2 Track accuracy (Mandatory)

3.2.2.2.1 The system will calculate the overall accuracy of the player.
3.2.2.2.2 The accuracy calculation shall be calculated as follows:

Shots fired: Every time the user presses the fire key a counter is incremented.
hits: When a bullet hits a zombie it will be flagged as a hit.
Accuracy: (hits / shots fired) * 100

3.2.2.2.3 The system shall record this calculated value to the players user account data and store it on the database.
3.2.2.2.4 The client shall synchronize the accuracy with the server after every level.
3.2.2.2.5 The client shall synchronize the accuracy with the server after every 50 shots fired.

3.2.2.3 Track weapon usage (Optional)

3.2.2.3.1 The system shall keep track of the frequency of guns used by the player.
3.2.2.3.2 The system shall keep a count of each shot fired by each gun and record those to the players user account data and store it on the database.
3.2.2.3.3 The percentage of each weapon fired shall be calculated as follows:

Pistol shots fired: The number of shots fired by a specific weapon, in this example a pistol.
Total shots fired: The number of shots fired by all weapons.
Usage: (Pistol shots fired / Total shots fired) * 100
3.2.2.3.4 The client shall synchronize the data with the server whenever the accuracy is syncronized.

3.2.2.4 Track wins and losses (Mandatory)

3.2.2.4.1 The system will track the number of wins and losses of the player and record it to the user account data and store it on the database.

wins: Wins are when a player successfully completes a level
losses: Losses are when a player fails to complete a level or quits the game before completing the level.

3.2.2.5 Track user cash earned (Mandatory)

3.2.2.5.1 The system will track the amount of cash the player earns as they play the game. This number will just be a running count and will be recorded to the user account data and stored in the database.
3.2.2.5.2 The client shall synchronize the amount cash with the server after every level.
3.2.2.5.3 The client shall synchronize the amount cash with the server after each purchase in the buy screen.

3.2.2.6 Calculate users score (Mandatory)

3.2.2.6.1 The system will create a composite score and record it to the user account data and store it in the database.

3.2.2.6.2 The score will be a composite of several factors including:

- Zombies killed
- Accuracy
- Win Percentage
- Cash Earned

3.2.3 Single Player Gameplay

Author: Chad Nelson

3.2.3.1 Player Movement

3.2.3.1.1 The player shall move across the level determined by keyboard input. (Mandatory)
3.2.3.1.2 In the default mode, the W key shall move the player up, the A key shall move the player left, the S key shall move the player down, and the D key shall move the player right. (Mandatory)
3.2.3.1.3 In the default mode, the player will be able to move in eight different directions depending on varying key combinations of WASD. (Mandatory)
3.2.3.1.4 In the default mode, the player will rotate and face the direction of the mouse cursor. (Mandatory)
3.2.3.1.5 An alternative mode will allow the player to strife left and right with A and D, and move toward and away from the mouse cursor with W and S. (Optional)

3.2.3.2 Zombie Movement

3.2.3.2.1 The zombies shall move randomly around a level. (Mandatory)
3.2.3.2.2 When a zombie encounters a player, they will chase the player. (Mandatory)
3.2.3.2.3 Zombies shall not clump up on top of each other. (Preferred)

3.2.3.3 Fire Weapon

3.2.3.3.1 The player shall be able to fire a weapon by holding either the spacebar or F key. (Mandatory)
3.2.3.3.2 The client shall toggle between a primary and secondary weapon when the player press the tab key. (Mandatory)
3.2.3.3.3 When a player fires, a graphic representation of a bullet will appear on the screen. (Mandatory)
3.2.3.3.4 A player will not be able to fire while reloading. (Mandatory)
3.2.3.3.5 A zombie shall loose health when hit by a bullet. (Mandatory)
3.2.3.3.6 A zombie shall die when it's health reaches 0. (Mandatory)
3.2.3.3.7 A new player shall start single player mode with a fully loaded pistol. (Mandatory)

3.2.3.4 Collision Detection

3.2.3.4.1 Each level will contain a unique set of boundaries where players are not allowed to move (i.e. water, off the level, fences). (Mandatory)
3.2.3.4.2 A player shall not be able to enter a boundary area or cross a boundary. (Mandatory)
3.2.3.4.3 Zombies shall not be created in boundary areas. (Mandatory)
3.2.3.4.4 Zombies shall not move into boundary areas. (Mandatory)
3.2.3.4.5 Boundaries can be selected to allow or disallow bullet penetration. (Preferred)

3.2.3.5 Health

3.2.3.5.1 The player shall start each level with 100 health points. (Mandatory)
3.2.3.5.2 A zombie shall attack the player if he is within a distance of 50 pixels.

(Mandatory)
3.2.3.5.3 The player shall loose health when a zombie attacks him. (Mandatory)

3.2.3.6 Ammo

3.2.3.6.1 The player shall not fire if the currently selected weapon has no ammo. (Mandatory)
3.2.3.6.2 The player shall not fire if he needs to reload. (Mandatory)
3.2.3.6.3 If the player fires with an empty clip, it will reload the gun. (Mandatory)
3.2.3.6.3 The player shall not fire while reloading. (Mandatory)
3.2.3.6.4 The player shall fire bullets when pressing a firing key according to the selected weapon's firing rate. (Mandatory)

3.2.3.7 Pickups

3.2.3.7.1 When a zombie dies, it will sometimes randomly drop a pickup. (Mandatory)
3.2.3.7.2 A pickup can exclusively increase one of the following statistics: health, cash, or ammo. (Mandatory)
3.2.3.7.3 The pickup shall be removed from the map when a player walks over the pickup. (Mandatory).
3.2.3.7.4 When a player retrieves a pickup, its benefit is added to the player's stats. (Mandatory)
3.2.3.7.5 A pickup shall be removed from the map after 60 seconds. (Preferred)

3.2.3.8 Powerups

3.2.3.8.1 The user shall be able to purchase powerups at the store (buy screen). (Mandatory)
3.2.3.8.2 The user shall be able to purchase a powerup titled "Body Armor" that provides extra health. (Mandatory)
3.2.3.8.3 The user shall be able to purchase a powerup titled "Max Ammo" that provides extra ammo. (Mandatory)

3.2.3.9 Level Completion

3.2.3.9.1 The player shall be taken to the Buy Screen after completing or failing a level. (Mandatory)
3.2.3.9.2 The client shall send kill data and ammo counts to the server after the player completes or fails a level. (Mandatory)
3.2.3.9.3 The client shall send to the server indication that a player has completed a level.
3.2.3.9.4 The player shall complete a level after killing all the zombies.
3.2.3.9.5 The player shall fail a level when their health reaches 0.

3.2.4 Multiplayer Gameplay

Author: Cole Anagnost

3.2.3.1 Player Versus Player

3.2.3.1.1 The players in a game shall be scored individually (Mandatory)
3.2.3.1.2 The player shall not be able to damage another player (Mandatory)
3.2.3.1.3 The player shall be able to damage another player if a "friendly fire" option is enabled (Optional)
3.2.3.1.4 The players shall be scored as a team if the "group scoring" option is enabled (Optional)

3.2.3.2 Game Completion

3.2.3.2.1 The player shall be taken to the game screen after a multiplayer game is completed. (Mandatory)
3.2.3.2.2 The client shall send kill data and player stats to the server upon game completion (Mandatory)
3.2.3.2.3 The multiplayer game is completed when all players healths reach 0. (Mandatory)
3.2.3.2.4 The multiplayer game is completed when the players kill all the zombies. (Preferred)

3.2.3.3 Connection Errors

3.2.3.2.1 The client shall display an error for the following reasons:
- Connection to server lost (Mandatory)
- Another player loses connection to the server (Mandatory)

3.2.3.2.2 The server shall allow the player time to re-establish connection to the server if connection is lost in-game (Optional)

3.2.5 Match Making

Author: Tas Fox

3.2.5.1 Creating a room (Mandatory)

3.2.5.1.1 Once the player has successfully logged in, and entered the multiplayer lobby the player can choose to create a room.
3.2.5.1.2 The user will be prompted to enter a name for the the room by a popup text box.
3.2.5.1.3 The user shall enter a name for the room and will be automatically placed in the room upon creation.
3.2.5.1.4 The user shall be able to modify and configure the settings of the game.
3.2.5.1.5 The settings the user can modify are: Level, Number of Players, Difficulty

3.2.5.2 Display room list (Mandatory)

3.2.5.2.1 The system shall display all available games in the room panel in the multiplayer lobby.
3.2.5.2.2 The system shall not show games that are ineligible for the player to play
3.2.5.2.3 Factors for ineligibility are:

- Game room is full
- Game has already started
- Player does not have the pre-requisite levels unlocked

3.2.5.3 Display game room (Mandatory)

3.2.5.3.1 Once the player has joined a room the game room screen is displayed
3.2.5.3.2 The game room screen shall display other players also in the room
3.2.5.3.3 The game room screen shall not display players not in the game room
3.2.5.3.4 The game room screen shall display the settings for the game configured by the game creator.
3.2.5.3.5 Players in the room that did not create the room shall not be able to modify the game settings.
3.2.5.3.6 The chat window shall only show chat messages from players in the game room
3.2.5.3.7 If the game creator leaves an occupied game room, then another random player of the room becomes the game creator.
3.2.5.3.8 If there is one player in a game room and they leave, the room is deleted.

3.2.5.4 Start game (Mandatory)

3.2.5.4.1 The game creator can start the game by clicking a "Start" button.
3.2.5.4.2 The game will not start if all users have not indicated their readiness by clicking a "Ready" button.
3.2.5.4.3 The "Ready" button will be depressed after clicking.
3.2.5.4.4 If the game creator changes a setting, a depressed ready button will become clickable again and the player must reclick before the game starts.

3.2.5.5 Display Errors (Mandatory)

3.2.5.5.1 The system shall display an error for the following reasons: Duplicate game name during creation, Game is full, Invalid Game name, Creator tries to start the game, but users are not ready
3.2.5.5.2 For a Duplicate game name error the system will display an error pane that reads "Game name already exists, please enter a new name."
3.2.5.5.3 For a game is full error the system will display an error pane that reads "Game room is full"
3.2.5.5.4 For a Invalid game name error the system will display an error pane that reads "Invalid Game Name."
3.2.5.5.5 If the creator of the game attempts to start the game and all of the users have not checked the ready check box then the system will display a error pane to the creator that reads "The game cannot start until everyone is ready."
3.2.5.5.6 If the creator of the game attempts to start the game and all of the users have not checked the ready check box then the system will display an error pane to all users who have not checked the ready check box that reads "(Game Creators Username) cannot start the game until you are ready."

3.2.6 Persistent User

Author: Tim Flanigan

3.2.6.1 Creating an Account (Mandatory)

3.2.6.1.1 The player shall be able to create a new account that stores his player data, which includes unlocked weapons, levels, and the amount of money obtained.
3.2.6.2.2 The user shall be able to choose a custom username.
3.2.6.2.3 The user shall create a password that is linked with the username choosen, to create a unique account.

3.2.6.2 Logging into an Account (Mandatory)

3.2.6.2.1 The player shall be able to input a username and password.
3.2.6.2.2 The client shall connect to the server, which authenticates and send back the correct user data.
3.2.6.2.3 If the username was incorrect, the client will display "Incorrect username", if the password is wrong the client will display "Incorrect password".

3.2.6.3 Account Passwords (Mandatory)

3.2.6.3.1 Any player creating an account will need to use a password to make sure that no other player is allowed to use that username. Usernames will be linked to the high scores page, so accounts need to be unique.

3.2.6.4 Player Data (Mandatory)

3.2.6.4.1 The player shall have player data attached to the account they created. This will hold the information pertaining to the users. It will store the following:

- Money obtained
- Levels unlocked
- Equipment
- Score

3.2.6.5 Error Messages (Mandatory)

3.2.6.5.1 The client shall recieve an error message for the following reasons:

- Invalid username when creating an account
- Invalid username when logging in
- Invalid password when logging in

3.2.6.5.2 If the user inputs an Invalid username when creating an account, the client will display the message "Username already taken."
3.2.6.5.2 If the user inputs an incorrect username when logging in, the client will display the message "Username incorrect."
3.2.6.5.3 If the user inputs an incorrect password when logging in, the client will display the message "Password incorrect."

3.2.7 Online Chat

Author: Cole Anagnost

3.2.7.1 Main Lobby

3.2.7.1.1 The client shall present the player with the main chat lobby after they select "Multi-player Game" from the main menu (Mandatory)
3.2.7.1.2 The player shall be able to chat with other players in the main lobby (Mandatory)
3.2.7.1.3 The player shall be able to view a list of players currently in the main lobby (Mandatory)

3.2.7.2 Game Room Chat

3.2.7.2.1 The game room shall provide chat capabilities for players in the game room. (Mandatory)

3.2.7.3 In Game Chat

3.2.7.3.1 The player shall be able to chat with the other players during a multiplayer game. (Mandatory)

3.2.8 Unlockable Equipment and Levels

Author: Tim Flannigan

3.2.8.1 Buy Screen (Mandatory)

3.2.8.2.1 The player will purchase equipment that has been unlocked on this screen. (Mandatory)
3.2.8.2.2 Equipment shall become available to the user when they obtain enough money to purchase the equipment. (Mandatory)
3.2.8.2.3 The user shall be able to select the item and view information about it. (Mandatory)

3.2.8.2 Level Select Screen (Mandatory)

3.2.8.2.1The player will be able to select the level to be played on this screen. (Mandatory)
3.2.8.2.2 If the level is unlocked and available to play then it will be selectable. If the level is locked then the button will be grayed out. (Mandatory)
3.2.8.2.3 For the player to unlock a level they will need to complete the previous level in the game. (Mandatory)

3.3 PERFORMANCE REQUIREMENTS

3.3.1 Graphics Performance

3.3.1.1 Frame Limiting

3.3.1.1.1 The game client shall run no faster then 50 frames per second

3.3.1.2 Minimum Frame Rate

3.3.1.2.1 The game client shall not have an average frame rate below 30 frames per second

3.3.2 Network Performance

3.3.2.1 Chat Latency

3.3.2.1.1 Chat messages must be sent to all applicable clients in 1 seconds or less.

3.3.2.2 Client updates

3.3.2.2.1 Game clients on a local network must update within 50 milliseconds

3.4 DESIGN CONSTRAINTS

3.4.1 Size Constraint

3.4.1.1 File Size Limitation

3.4.1.1.1 The games executable shall be no larger than 25 Megabytes.

3.5 SOFTWARE SYSTEM ATTRIBUTES

3.5.1 Reliability

[None]

3.5.2 Availability

[None]

3.5.3 Security

[None]

3.5.4 Maintainability

[None]

3.5.5 Portability

[None]

3.6 OTHER REQUIREMENTS

[None]

APPENDIX A

A-1 SCREEN FLOW DIAGRAM

//Insert Diagram from Screenshot Assignment

A-2 SCREENSHOT MOCKUPS

//Insert Mockups from Screenshot Assignment