Team 25% Bodyfat Final Proposal

The Application

Our game will resemble a horde-like shooter with some rpg aspects. Players will first create accounts on our server and login with their credentials to connect. At this stage they will be able to create their own characters based on their playstyles. This is done by choosing various skills on a tree. They can play solo games or multiplayer games, view their stats, view highscore boards, and communicate with others through a game chat.
A game will start with the players in the center of the map. The first wave of monsters will spawn and start going after the players. Every monster kill will increase a players score, which will be translated into exp and/or skill points. Once a wave is eliminated a new wave with more or tougher monsters will spawn after a short delay. This will continue until the players are defeated. Different game metrics like score, waves, number of kills, etc. will be recorded and viewable on the stats page. Game maps will also be randomized.

Every character will have the opportunity to unlock every weapon and will be able to cycle through unlocked weapons. Ammo will be found at certain locations on the map during gameplay. Players can upgrade the power and traits of the guns through the player skill tree. Players will also be able to upgrade their character traits in game through leveling up. The current character skill tree will be viewable and editable on the skill tree page.

Players will be able to communicate with their teammates and see the actions of their teammates in real time. This will allow for true cooperation during the game.

Guns will vary with respect to power, range, speed of fire, etc. Enemies will vary with respect to health, movement, attacks, etc.

User Analysis

One of the primary features of our application is the way in which we reward our players. After hours of competitive analysis one of the key aspects that captivates players is a good reward system, customization, and in-game upgrades. Our game will attempt to address each of these issues as well as integrating a high score leaderboard. By doing so we are developing a compelling app that appeals to all types of gamers, those that play casually and those that are more intense and concerned about in game status. Ultimately, our target users are those that are familiar with horde style game modes and would like to carry this experience on the go, as we are developing a mobile application.

Technical Analysis

Our game is quite computationally heavy on the server side since this is where calculations such as player movement, enemy AI and movement, bullet collisions, and game chat processing will occur. The server will be responsible for synchronizing each of the player's screens in order to promote a good multiplayer experience. We will be using MongoDB to store player accounts including their stats and skill tree choices, and chat logs. This will be continually updated after each game session. The client side will host the game canvas, UI elements, and

lobby/matchmaking interface. There will be a focus on smooth animations and a polished UI using CSS3 and sprites. Serving as the intermediary between the client and server communication, socket.io will be used for the chat system and multiplayer game engine. We also plan to use PhoneGap to create native versions of the app for smoother gameplay.

MVP Goals

- Users will be able to create an account (username, password) stored on the server
- Users will be able to login to previously created accounts
- Synchronized connections between multiple players
- Ability to invite players into a game lobby
- Ability to chat to players in the lobby
- Ability to upgrade stats upon leveling
- Users will be able to play solo games
- Users will be able to connect to play multiplayer games
- Users will be able to check player stats stored on the server
- Users will be able to upgrade their characters with skill trees stored on the server
- Users will be able to communicate with teammates
- Users will be able to shoot enemies
- Users will be able to pick up power-ups
- Users will be able to use skills
- Users will be able to quit a game

Secondary Goals

- Friends list functionality for game invites
- Ability to communicate through voice
- Added mechanics such as moveable/repairable barriers
- A community forum for the game
- Heroku deployment

Feature List

- User Accounts
    - This will allow players to keep track of their game stats, levels, and online identity.
    - Accounts will be created through a front-end interface where the username and password are relayed to the server which is then saved as an object into a MongoDB database.
- User Authentication
    - This will allow players to log into the game lobby and access their skill trees.
    - Authentication will be handled by the authentication plug-in developed by Evan Shapiro. The username and password combination will be looked up in the MongoDB database and upon success will allow the user access to the game lobby. On failure, it will alert the user that one of the fields was entered incorrectly.
- Player Synchronization

    - This will allow for consistent multiplayer gameplay.
    - Player movements will be sent from the client to the node.js server through a socket using socket.io. The server will then calculate the player's new position and broadcast it to the other players using socket.io. This will appear the clients as if movements were synchronized across multiple gaming platforms. The same idea will be expanded to player shooting and collision detection. Enemy movement will be calculated on the server and then emitted to every player via sockets.
- Chat System
    - This will allow for communication in the lobby.
    - Similar to how player movements are synchronized, sockets will be used to transmit messages between players.
- Lobby System
    - This will show all the players currently in the game lobby.
    - Players that accepted a particular game invite will be added to a special 'Lobby' array on the node.js server. This array will contain all the player objects pulled from MongoDB via the socket connection that accepted the invite.
- Stats/Leveling System
    - After killing enemies, players will receive experience that goes toward their level. Upon leveling up, players will get the option to upgrade one of their stats (health, armor, damage, range, rate of fire, etc...).
    - This will be implemented through a div overlay that appears upon leveling up. The player will select which skill to level and this will send an ajax request to the server. The ajax request will then update the player's level field in the MongoDB database and upon success will update the player's view showing him that he has increased his stat. Levels will also be updated in the player's object in MongoDB.
- Player Movement/Controls
    - Players will be able to move around the game map and shoot in all directions.
    - Movement will be implemented by a pseudo-analog style touch system by which a player will put his thumb on the screen and move it in the direction he wants to move, similar to how physical analog controls work. Shooting will be similar - another analog will be by the other thumb and whichever way the user moves his thumb is the direction he will shoot in and moving the analog away from its resting (center) position will cause his character to fire. This will be implemented code-wise by using HTML5's multitouch and the node.js to calculate the distance and direction delta's from the analog's resting position and translate it into moving the player and shooting bullets.
- Skill Tree
    - Players will be able to customize their gameplay style by selecting particular guns to unlock and which guns to upgrade. The amount of skill points a player has is correlated with the player's level.
    - The skill tree will be implemented by a screen of buttons whereby users touch the

weapon they want to unlock and then the weapon's particular upgrades (rate of fire, bullet spread, damage, pushback, etc..). Once a user is satisfied, submitting their skill tree will send a request to the node.js server which will then update the player's skill object in the MongoDB database.

- Stats Page
    - Players will be able to view their own stats and also other player's stats.
    - MongoDB will keep track of each player object's game stats and save them into multiple session objects. When a player navigates to another person's stats page, an AJAX request will be sent to the node.js server which will then send a query to the MongoDB database and pull that particular player's stats out and send it back to the client for display. DOM manipulation through jQuery will then be used to update the stats page with the selected player's info.
- Enemy AI
    - Enemies will be somewhat intelligent and aim at the players/move towards them.
    - Enemy movement will be calculated by the server using a simple AI algorithm whereby each enemy has a certain range where they will start moving towards the nearest enemy. Aim will be calculated server-side by using the player's position and the enemy's position. Enemies will also have differing stats and abilities implemented by creating various subclasses of the base 'Enemy' prototype.
- In-Game Callouts
    - Players will be able to communicate in game by selecting from four different callouts that each play a different sound and display a message to the other players. These include: "group up, split up, need ammo, help me".
    - This feature will be implemented by the user touching one of four buttons that then displays a warning message in the center of the screen using canvas. Each callout will play its own sound that is broadcast to the other players using socket.io and the HTML5 audio element.
- Create Game
    - Users will be able to make a new game lobby and invite players to it.
    - Upon touching the 'create game' button, this will create a new 'Lobby' object and push it to the node.js server's Lobby array. This lobby array keeps track of the sockets (current players) that are attached with this particular gaming session.
- Join Game
    - Users will be able to quickly join game lobbies that are already made and not yet full.
    - Upon touching the 'join game' button, this will send a request to the server that looks through the 'Lobby' array until it finds one with empty player slots. This will then add the requesting player's socket object to that lobby's array and the player will then be transported to that lobby via a page redirection on the front-end upon the AJAX request's success.
- Ammo/Health Pickups
    - Players will be able to pick up health and ammo that is randomly scattered

throughout the game map.
    - The node.js server will randomly choose locations on the map to add ammo and health pick ups. Once a location is chosen, the client's will render the ammo and health. When a pick up is taken, the client socket will send a message to the server which will then relay the fact that the pick up is no longer there to the other clients using socket.io, and also delete the pick up from the 'PickUps' array on the server.
- UX Design
    - The app will have a clean interface that is responsive to users and is considerate of the mobile environment.
    - CSS design techniques will employ the use of icons and buttons for the UI. Features such as the lobby system will not be crowded due to the low screen real estate and there will be a focus on using div overlays for things like the end-game stats report. CSS3 will be used to ensure the application is visually appealing to users.

Implementation Plan

<u>Week 4</u>

- Josh
    - Socket.io connection between multiple people
    - Chat system between the sockets
    - MongoDB interface on the server for saving user accounts
    - User account creation
- Marco
    - Player movement and shooting
    - Simple enemies
    - Ability to kill enemies
- Eman
    - Skill tree
    - Weapon design
    - Simple animations

<u>Week 5</u>

- Josh
    - Lobby system/invitations (includes creating/joining game)
    - Loading screen
    - Synchronized gameplay between multiple players
- Marco
    - In-game stat upgrade system
    - In-game weapon selection
- Eman
    - Stats page

    - Health/ammo pickups implementation

Week 6

- Josh
    - Saving in-game stats to MongoDB
    - Intelligent enemy AI
- Marco
    - Callouts
    - Ultimate ability
    - Leveling system
- Eman
    - End-game stats screen
    - Enemy differentiation

Week 7

- Josh
    - UI polishing
    - PhoneGap utilization
- Marco
    - Game mechanics polishing
- Eman
    - Animation polishing

List of Technologies Utilized

1. Javascript
2. Canvas
3. HTML
4. CSS
5. DOM Manipulation
6. jQuery
7. AJAX client
8. AJAX server
9. node.js
10. express.js
11. socket.io
12. MongoDB