Executive Summary

This project, with the working title Supa Moba, is a multiplayer online battle arena where teams fight for victory. In each instance of a game, two teams consisting of three players each will be spawned into a map with the goal of destroying the opposing team's base. Each player will be able to choose from a collection of different characters, each with their own appearance, personality, set of abilities and play style. Each team will also have their own set of non-player characters called minions which assists the heroes in battle. The bases of each team will also come with a set of immovable towers and structures that inhibit or attack members of the opposing team. These towers and structures will have to be destroyed before one can attack the core of the base. The game will also contain a set of neutral non-player characters outside of the player bases that will attack any player who enters their camp. Teams can gain experience to level up and gain health, mana, and new abilities. Experience can be gained from killing enemy players, minions, base structures, and neutral camps. Supa Moba will employ a retro 8 bit art style both to reduce the strain on the developers and to provide a new and unique experience in the already well defined MOBA genre.

The value that Supa Moba will provide to its users is the fun that they have. Creating a game that will be fun for all players will require many pieces to come together. First out multiplayer networking must seem seamless such that player movements are always smooth, games are not disconnected, and lag is not significant enough to be noticeable. We must also ensure that there are a large variety of player characters to play, with no characters being more powerful than the others. This will ensure that users can find a style of play that best suits their wants. Finally, players will be able to level up to unlock new abilities and characters so that they stay engaged with the game over time and have targetable goals to work towards.

Customer Archetypes

The sole customer archetype that Supa Moba is geared towards is the player. Player will be the only users of the system and the only people paying to keep the project running. The duty of the player is to be a valuable team member in the game and work towards getting better and winning each match. The motivation of these players will come directly from their enjoyment of the game. To ensure this, we will have to accommodate a wide variety of play styles and ensure that new players will be able to succeed at the game. The enjoyment of the game will suffer if our client-server architecture experiences any errors or significant delays. Mitigating any games crashing or delays or stuttering will be crucial for the user experience to be responsive and fun.

Customer Needs

A detailed list of customer needs is provided. Customer value is linked to each of the customer needs. Value description is realistic and reflect comprehension of the customer perspective.

Easy to use GUI

Starting up the game and navigating menus should be an easy experience for the customer. It is important that they are able to easily access their profile, options, and other basic functions of various menus.

Stable connection during games

Customers may have varying levels of internet connection. The game should have features like predictive rendering and reconnection handlers to ensure the customer will still have a pleasant experience and level of performance when connection goes awry.

Architecture that allows a range of computer specs
It is not necessary for the customer to have high-spec computer hardware to run Supa Moba (Nvidia RTX 2080, 1080 ti, Intel Core I9, etc). The architecture of the game will allow fair experience between average devices so that a lower-spec computer will not have a disadvantage. This also is to consider the budget of the customer as high-spec hardware is expensive.

MOBA standard of array of characters
MOBA centric customers will require a good amount of variability between choice in character. These characters must also have a good balance of abilities that make the game experience worthwhile to the customer in terms of entertainment, scaling skill-cap, and team-play.

Unique map design
This concerns the Battle Arena (BA) part of a MOBA and the playing experience of the customer. The map needs to be a good fit for the 8-bit art style and movement mechanics that are signature to Supa Moba.

Compatibility for various screen sizes
There is a wide range on what screen size and resolution our customer will use when playing Supa Moba because of the wide market of displays. Therefore, there is a need to scale Supa Moba's viewable frame to accommodate many levels of screen scaling without giving wider, thinner, taller, higher res, screens unfair advantages or disadvantages.

Information for character and gameplay mechanics
It is known for a MOBA to have a high learning curve. MOBA centric customers need adequate explanations and data about characters and abilities for the learning process. This information needs to benefit new customers as well as higher-skilled, long time customers.

Deliverables

A detailed list of potential deliverables is provided. Each deliverable includes a brief description, and a projected delivery time.

Object Orientation of Supa Moba
Description: Have the entities, player objects, game state, and networking class diagram put together and planned out.
Delivery: Sprint 1

Client Server Architecture for Six Clients

Description: Implement a Client Server Architecture using LuaSocket that supports 6 client's movements drawn out in real time.
Delivery: Sprint 1

Implement Basic Fighting Mechanics
Description: MOBA standard mechanics include basic ranged attacks on click and special abilities. Implement basic attack and one set of special abilities. This implementation includes hit detection of attacks.
Delivery: Sprint 2
Implement Working Map with MOBA structures (towers, minions)
Description: The base map (unfinished textures) including working minions, towers, and goal structures that respond to the gamestate (active health and attack, win based on destruction, number of downed towers, etc)
Delivery: Sprint 3

Implement Character and Ability Progression
Description: This implementation is for the progression of character level in game as they defeat minions and other players. Examples of this implementation could be a currency, item shop system or team ability leveling system.
Delivery: Sprint 4

Implement External Features: Game Lobbies, Character Select, Character bios, etc.
Description: The GUI and "out of game" experience. This is where a customer will view their profile, read info on characters, change character skins. Also included are pregame screens such as character selection for the six players and endgame screens showing game statistics (kill death ratio, team advantages, etc.)
Delivery: Sprint 5

Link to Project Board
https://tree.taiga.io/project/idiscipulo-supa-moba/backlog

Initial Set of PBIs
See the link to the project board. Each PBI is a user story, which are composed of subtasks. PBI's that are in the **Done** category were completed during Sprint 0. PBI's that are in the **In Progress** category have some or all of their subtasks selected for the current sprint. PBI's that are in the **New** category are planned for future sprints.

Initial Set of Tasks
See the above two.

Development Environment

Version control for this product is git and Github. We chose to use Github over other remote hosting services like bitbucket or gitlab because it offers seamless integration with git and unlimited free private repositories. For the local development environment we decided to let each developer use the OS and IDE of their choosing. This lets each member use the tools that they are most comfortable and efficient with while also ensuring that the product will work across all platforms.

Research

A solution for pathfinding

One of the first problems we required a solution for was mapping player and NPC movement across the map. After researching the methodology of games like Warcraft and StarCraft, we decided to implement a technique we learned about in Introduction to Artificial Intelligence: A* Search Algorithm. By programming a set of navigation points, each with a radius of free movement, the AI can find the closest point to its current position, the closest point to the mouse click position, perform an A* search to generate an optimal path of navigation points, and move freely within the radius of a given point. For movement that occurs within the radius of the same point, the AI can map freely, travelling in a direct line, without worry for collision. The downside of this is that we have to create a system to place navigation points (https://www.geeksforgeeks.org/a-search-algorithm/)

A solution for smooth movement

Our first attempt at fullscreen gameplay led to jerking movement, as the true pixel animation was more exaggerated as it scaled up. Our solution for this was to set a scale factor for the game, then scale the terrain and the character sprites up seperately. This allows us to draw the character sprites on the map with sub-pixel accuracy for much smoother movement, e.g. a scaling factor of four means that we can draw the characters with ¼ true pixel accuracy. Additionally, a scaling factor like this provides a simple method for calculating ability range, mouse click location, and movement (all things potentially affected by scaling).

A solution for a game engine

We have chosen to use the Love2D game engine for Lua. This engine is surprisingly fast, being built in Lua, a lightweight scripting language. This engine allows us to export to all major operating systems (Windows, MacOS, and Linux). It also provides the necessary graphics, input, and audio functionality, while having (almost) no opinionation to the structure of the game. This allows us to explore various design options without being limited by a program like GameMaker (https://love2d.org/).

A solution for networked multiplayer

Our solution for networking clients to the server in a low-latency manner is twofold: multithreading and efficient packets. Whenever the server connects to a client for the first time, it will generate a thread for that client, associated with the client ID. Whenever the server receives a packet from that client, it will unlock the thread to process all relevant game logic. Our packets from the client to the server will contain only the new input information from the client. Our packets from the server to the client will contain basic state information on the game, overriding the current state of the client on arrival. The downsides of this

model are primarily on us, the programmers, as multithreading is a slightly more complex method of updating the game state (https://www.youtube.com/watch?v=v2Q_zHG3vqg).