TANK WARS

Technical Design Document

This technical design document describes the technical details and requirements for Tank Wars, a multiplayer top down RTS game in which players have to destroy the other players' base camps to win.

Contents

Section 1 – Game Overview

1.1 Game Summary

Tank Wars is a multiplayer top-down RTS, where players have to destroy the base camp of the other players in order to win the game. Players will move, attack and buy units to advance in the game and will navigate a map of tiles to reach the other players.

1.2 Platform

The game will be developed for Windows/Mac/Linux and will be able to be controlled with a keyboard and mouse.

Section 2 – Development Overview

2.1 Development Environment

*The game will be made using the **ASGE Game Framework** and C++.*

2.2.1 Development Hardware

The game will be developed on Windows computers.

- *CLion requires a minimum of Windows 7, 2GB of RAM and 2.5GB of disk space (plus 1GB for caches).*
- *GitKraken requires a minimum of Windows 7, 4GB of RAM and 5GB of disk space.*

2.2.2 Development Software

*We will be using **CLion** to develop C++ code. Version control will be handled by **Git** with the repository hosted by **GitHub**. We will also be using **GitKraken** to provide a graphical interface for Git.*

*We will be using Trello for project management and to track our progress, our Trello board can be found here: **https://trello.com/b/6CQNjRyr/llp-game-3***

We have also made a Gantt Chart for time management.

2.2.3 External Code

We will be using the ASGE Framework for C++ written by James Huxtable to provide basic game functionality such as the game loop, sprite rendering and input events.

*We will be using **Toby Jones' networking library** to implement networking into our game.*

Section 3 – Game Mechanics

3.1 Main Specification Requirements

3.1.1 Networking

*Our game should be networked to allow us to create a multiplayer game. We plan to make our game for 2-4 players. We will be using **Toby Jones' networking library** to add networking into our game. On our main menu screen we will have two options: Host and Join.*

Host will allow you to create a server locally which players can then join if they have the correct IP address of the computer the server is hosted on (and are on the same wifi connection).

Join will allow you to enter an IP address and connect to the server on that machine.

3.1.2 Threading

Another technical requirement of our game is to use threading. We will be threading our input so that it is polled and updated in a different thread to the game.

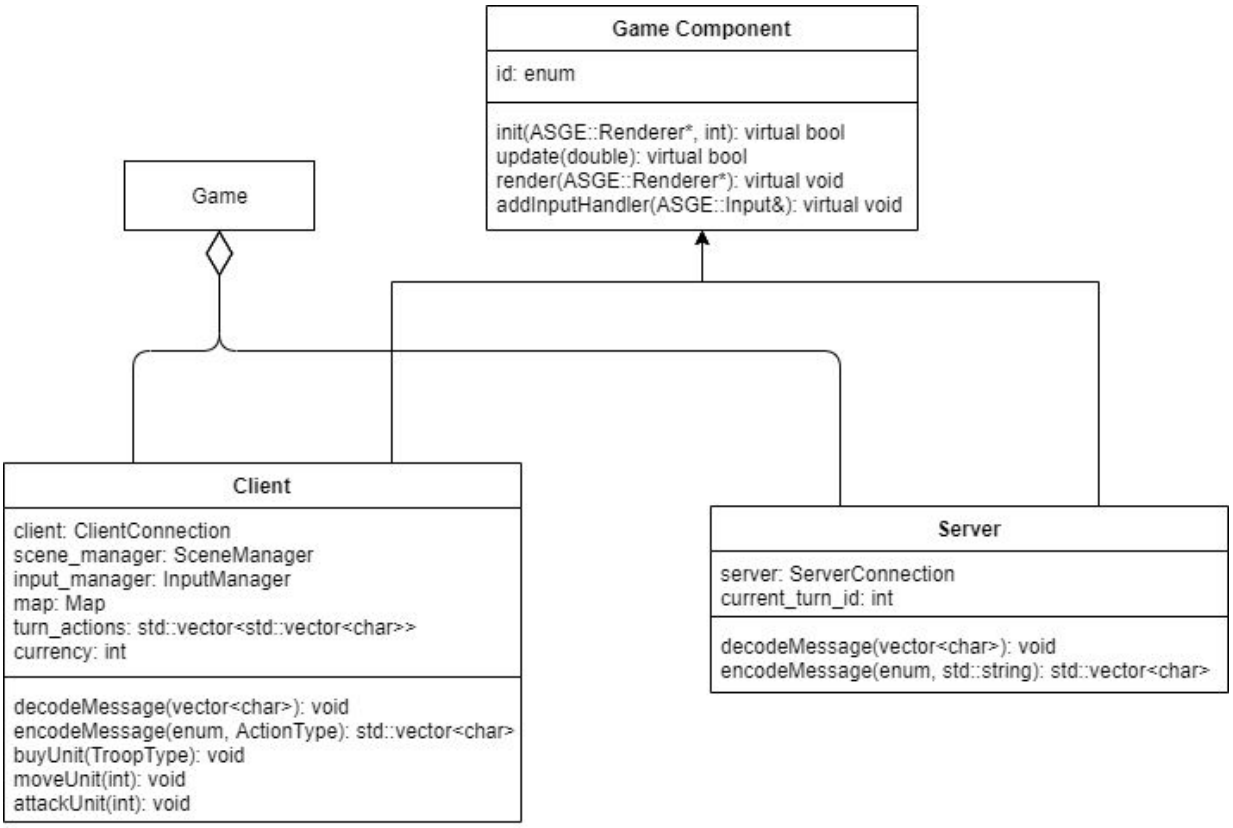
3.1.3 Turn-Based

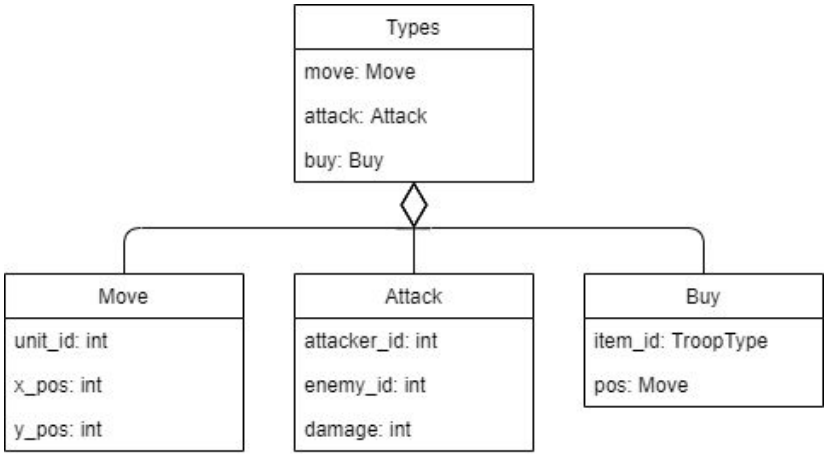
One major requirement of making an RTS is that players take turns to do actions. We will be setting this up by having an integer value for the ID of the player whose turn it currently is. We can then disable any players from making moves if the current turn ID doesn't match their ID.

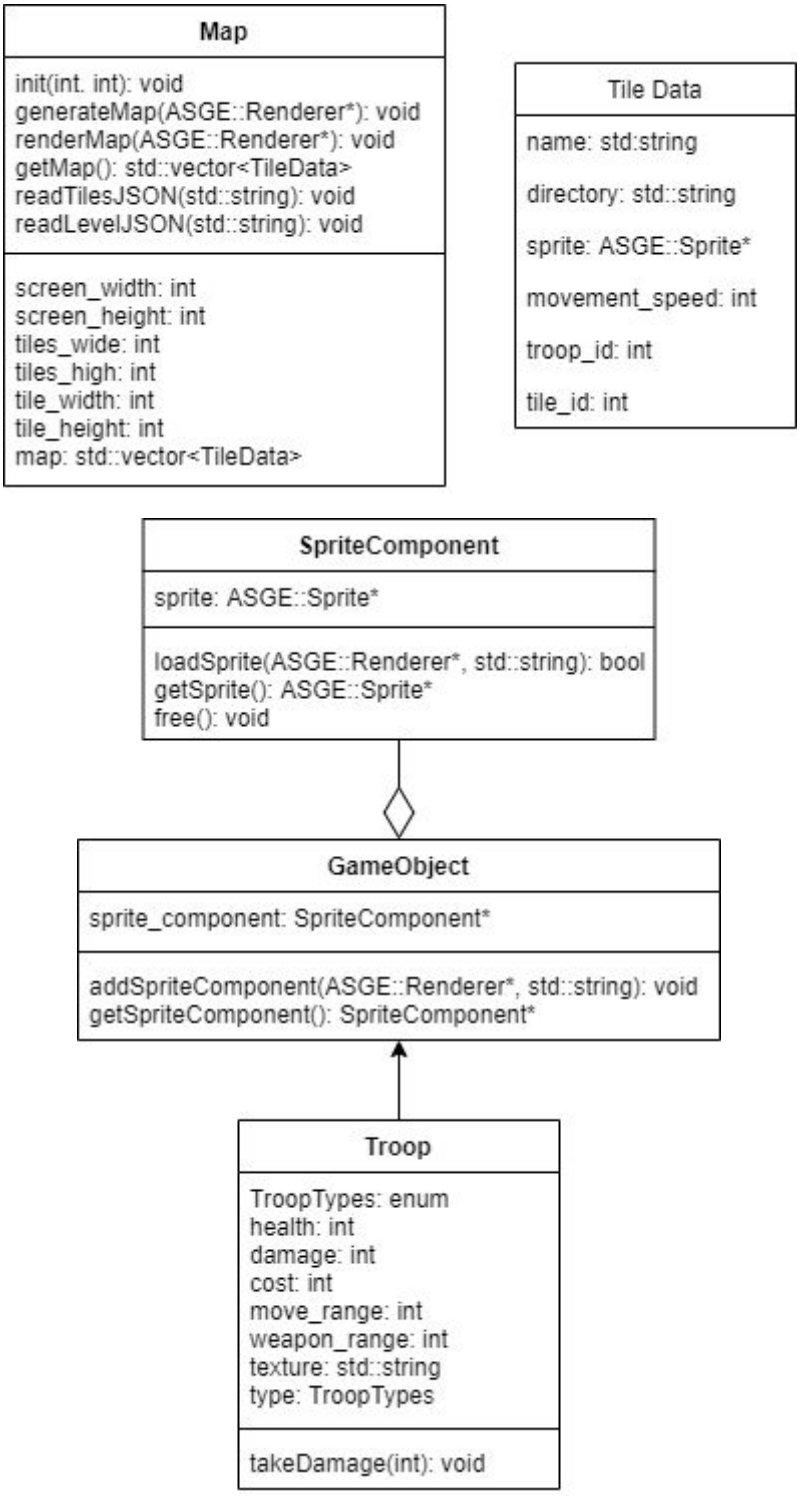
3.1.4 Units

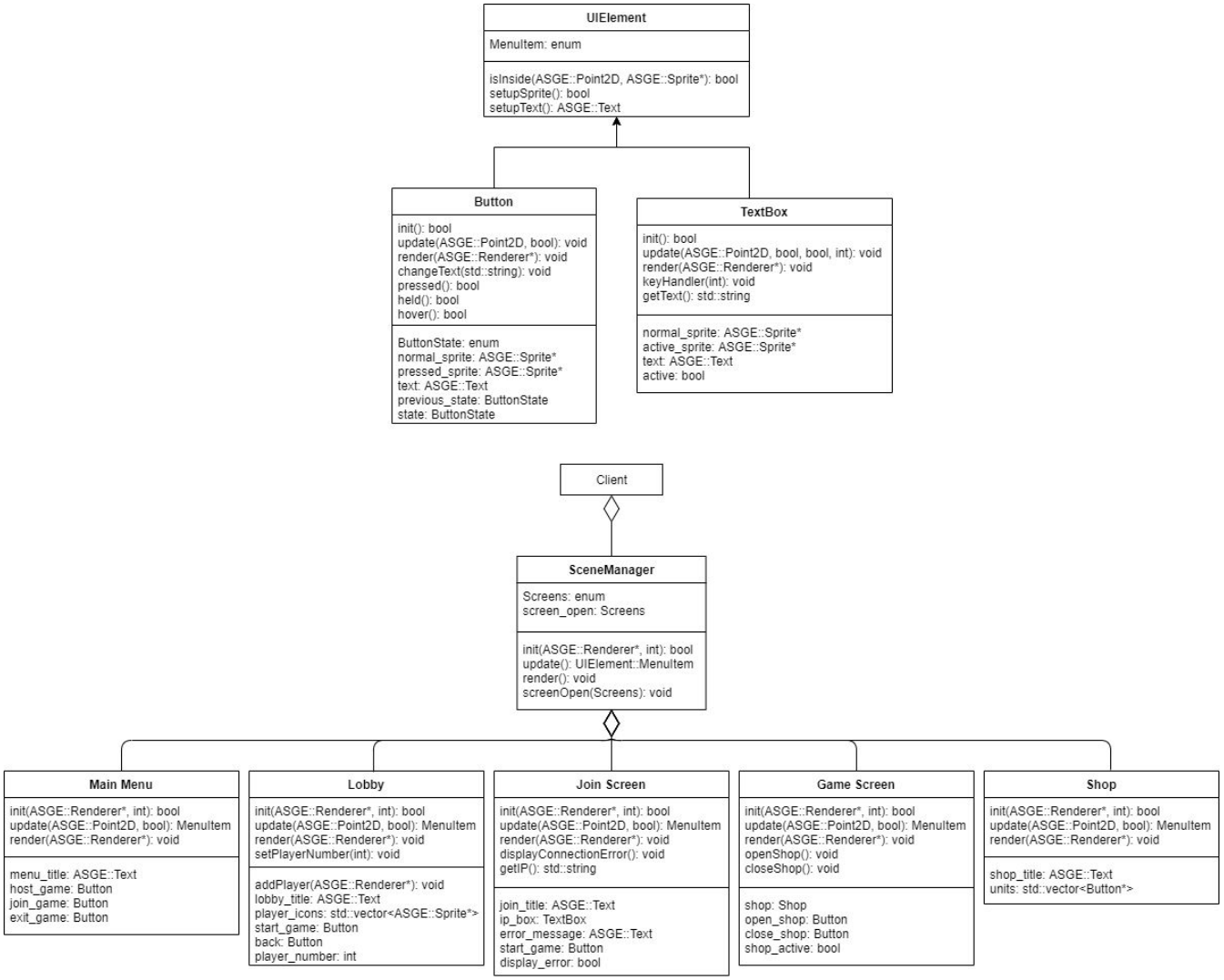
Our units will have different statistics depending on the type of unit they are. Units will be able to move and attack and the player will be limited on the amount of actions they can do by having a certain amount of time units per turn.

3.2 Architecture

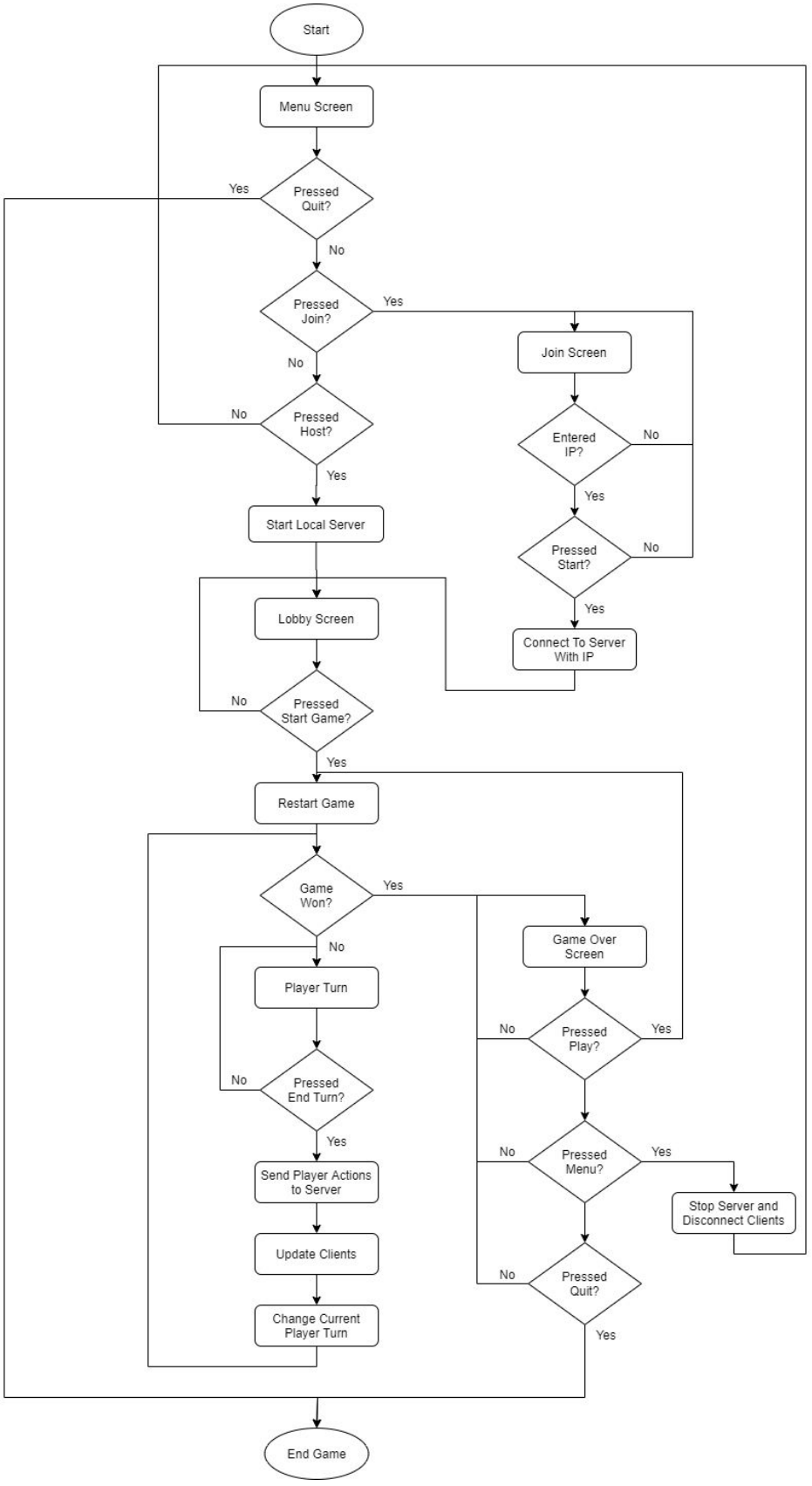





Map
init(int. int): void
generateMap(ASGE::Renderer*): void
renderMap(ASGE::Renderer*): void
getMap(): std::vector<TileData>
readTilesJSON(std::string): void
readLevelJSON(std::string): void
screen_width: int
screen_height: int
tiles_wide: int
tiles_high: int
tile_width: int
tile_height: int
map: std::vector<TileData>
Tile Data
name: std:string
directory: std::string
sprite: ASGE::Sprite*
movement_speed: int
troop_id: int
tile_id: int
SpriteComponent
sprite: ASGE::Sprite*
loadSprite(ASGE::Renderer*, std::string): bool
getSprite(): ASGE::Sprite*
free(): void
GameObject
sprite_component: SpriteComponent*
addSpriteComponent(ASGE::Renderer*, std::string): void
getSpriteComponent(): SpriteComponent*
Troop
TroopTypes: enum
health: int
damage: int
cost: int
move_range: int
weapon_range: int
texture: std::string
type: TroopTypes
takeDamage(int): void


UIElement
MenuItem: enum
isInside(ASGE::Point2D, ASGE::Sprite*): bool
setupSprite(): bool
setupText(): ASGE::Text
Button
init(): bool
update(ASGE::Point2D, bool): void
render(ASGE::Renderer*): void
changeText(std::string): void
pressed(): bool
held(): bool
hover(): bool
ButtonState: enum
normal_sprite: ASGE::Sprite*
pressed_sprite: ASGE::Sprite*
text: ASGE::Text
previous_state: ButtonState
state: ButtonState
TextBox
init(): bool
update(ASGE::Point2D, bool, bool, int): void
render(ASGE::Renderer*): void
keyHandler(int): void
getText(): std::string
normal_sprite: ASGE::Sprite*
active_sprite: ASGE::Sprite*
text: ASGE::Text
active: bool
Client
SceneManager
Screens: enum
screen_open: Screens
init(ASGE::Renderer*, int): bool
update(): UIElement::MenuItem
render(): void
screenOpen(Screens): void
Main Menu
init(ASGE::Renderer*, int): bool
update(ASGE::Point2D, bool): MenuItem
render(ASGE::Renderer*): void
menu_title: ASGE::Text
host_game: Button
join_game: Button
exit_game: Button
Lobby
init(ASGE::Renderer*, int): bool
update(ASGE::Point2D, bool): MenuItem
render(ASGE::Renderer*): void
setPlayerNumber(int): void
addPlayer(ASGE::Renderer*): void
lobby_title: ASGE::Text
player_icons: std::vector<ASGE::Sprite*>
start_game: Button
back: Button
player_number: int
Join Screen
init(ASGE::Renderer*, int): bool
update(ASGE::Point2D, bool): MenuItem
render(ASGE::Renderer*): void
displayConnectionError(): void
getIP(): std::string
join_title: ASGE::Text
ip_box: TextBox
error_message: ASGE::Text
start_game: Button
display_error: bool
Game Screen
init(ASGE::Renderer*, int): bool
update(ASGE::Point2D, bool): MenuItem
render(ASGE::Renderer*): void
openShop(): void
closeShop(): void
shop: Shop
open_shop: Button
close_shop: Button
shop_active: bool
Shop
init(ASGE::Renderer*, int): bool
update(ASGE::Point2D, bool): MenuItem
render(ASGE::Renderer*): void
shop_title: ASGE::Text
units: std::vector<Button*>

3.3 Game Flow


Start
Menu Screen
Pressed Quit?
Yes
No
Pressed Join?
Yes
No
Join Screen
Pressed Host?
No
Yes
Entered IP?
No
Yes
Start Local Server
Pressed Start?
No
Yes
Lobby Screen
Connect To Server With IP
Pressed Start Game?
No
Yes
Restart Game
Game Won?
Yes
No
Game Over Screen
Player Turn
Pressed Play?
No
Yes
Pressed End Turn?
No
Yes
Pressed Menu?
No
Yes
Stop Server and Disconnect Clients
Send Player Actions to Server
Update Clients
Change Current Player Turn
Pressed Quit?
No
Yes
End Game

3.4 Game Objects and Logic

3.4.1 Map and Grid

The map will be read in from a JSON file, allowing us to easily edit and change the map. The map will be based on a grid of tiles which the units will be able to move between. The map will have different types of tile (e.g. grass, sand, road) that will affect the move speed of the various units.

3.4.2 Troops

There will be four different types of troop in the game:

Each troop will have their own unit statistics including health, attack damage, move range, attack range and cost. Each player will have their own colour of units and to help distinguish troops, units that the player owns (can control and move) will have a white border to highlight them.

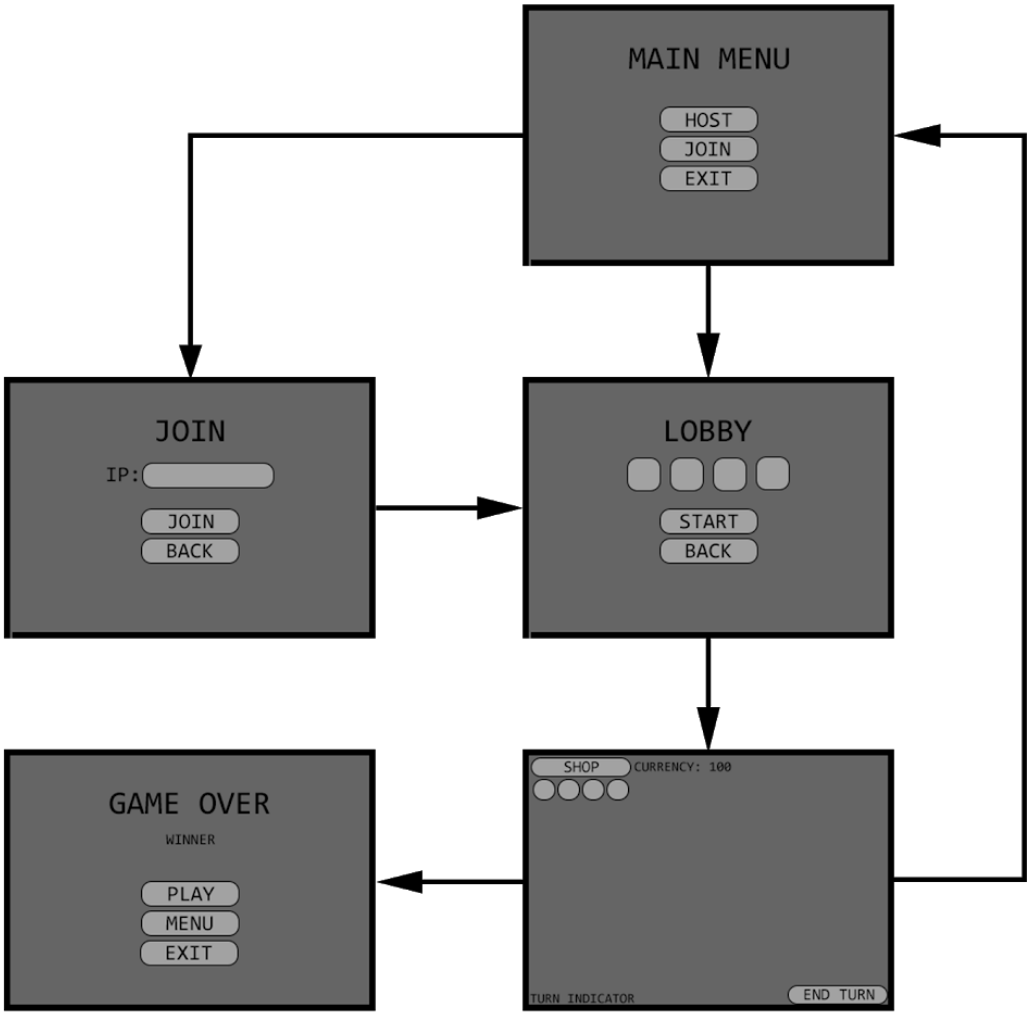
3.4.3 Scene Manager

The scene manager will manage the different scenes of the game. It will update the scenes as needed and render the current scene.

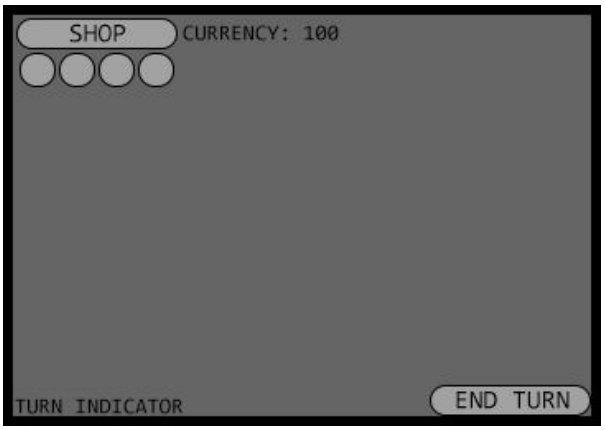
Section 4 – UI and Controls

The game will have a resolution of 1280px wide by 720px tall as this is a standard game size and will fit most monitors.

4.1 Screen Flow



4.3 Game Screen



The game screen will show the current view of the map, the player will be able to use WASD to move around and show different parts of the map on the screen. The UI will show the shop, where players can buy new units, whose turn it currently is, currency and an end turn button.

Then screen will also show information about when a current unit, when there is a unit selected. This information will show the units stats as well as the areas it can move to and any in range units it can attack.

4.4 Controls

The game will be used with a keyboard and mouse since this is most suitable for the style of game.

4.4.1 Keyboard

WASD: *Move Camera*

4.4.2 Mouse

Movement: *Move Cursor*

Left Button: *Select*

Right Button: *Deselect*