Software Requirements and Design Document

For

Group 2

Version 2.0

Authors:
Kendall Fretwell
Miles Brosz
Grant Leatherman
Caroline Mangrum
Joseph Riley

1. Overview (5 points)

Sketch & Strike is a simultaneous turn-based multiplayer game where 2 players engage in battle on a dynamically player created and altered terrain. The game introduces a unique mechanic where players can draw their actions within a limited time period, allowing them to create weapons and alter terrain such as creating walls or digging through obstacles. The player can also choose to attack or move their character. The outcome of each round is determined by the statistical result of an attack on the health bar assigned to each player.

Upon connecting, the host is able to change game settings to their liking. The host then shares their IP and secure code for the second player to join.

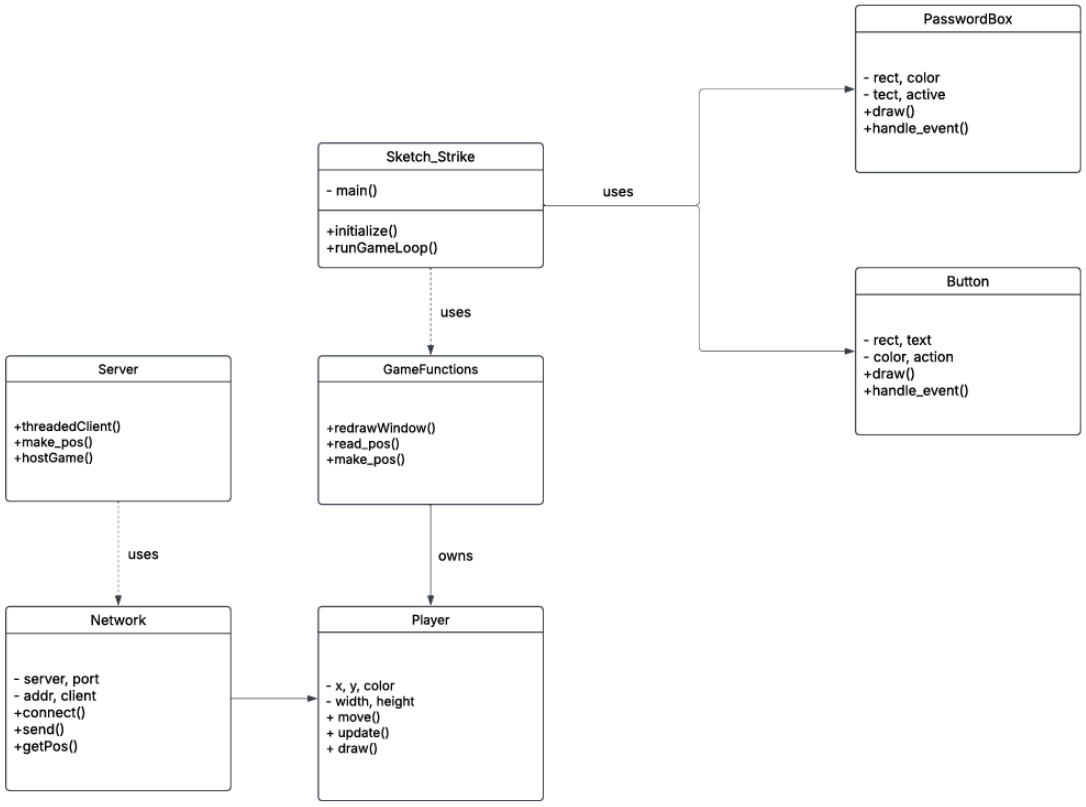
2. Functional Requirements (10 points)

1. *Multiplayer – High*
   The game's core functionality is built around multiplayer interaction
2. *Terrain Manipulation – Medium*
   Players need the ability to dynamically alter the terrain each turn via drawing actions
3. *Turn Implementation – High*
   Turn management ensures that both players' actions are processed after the host's chosen time interval
4. *Player Movement – Medium*
   Players must be able to move their characters across the terrain in order to avoid and launch attacks
5. *Modifying Objects – Medium*
   Players must be able to adapt their drawn objects
6. *Animations – Low*
   This provides visual feedback and enhances the player's experience
7. *Game Settings – High*
   The host must be able to change settings to their liking
8. *Collision Detection – High*
   Objects and characters interact based on the environment so the logic must be reflecting this

3. Non-functional Requirements (10 points)

1. *Performance – necessary for synchronization between players*
2. *Portability – ensure the game design can run on different OS*
3. *Maintainability – allow for easy changes and updates*

4. Use Case Diagram (10 points)



Server: This class represents the game server responsible for managing connections and processing client data. It includes methods for initializing the server instance (initialize()), handling socket connections (handle_sock()), and executing timed events (timeEvent()). The server likely manages network traffic, synchronizes game state across clients, and handles player connections/disconnections.

Network**:** This networking component handles all communication protocols between client and server. It maintains server connection information (server, port) and client socket data (addr, client). Key methods include connect() for establishing connections, send() for transmitting data packets, and getPktIn() for retrieving incoming network messages. This class abstracts the lower-level network operations from the game logic.

Search_Strike: This appears to be the main game application class. It contains the entry point (main()) and initialization functions (initialize(), runGameLoop()). It likely orchestrates the various game components and controls the overall application flow, serving as the central hub that connects UI elements, game logic, and networking.

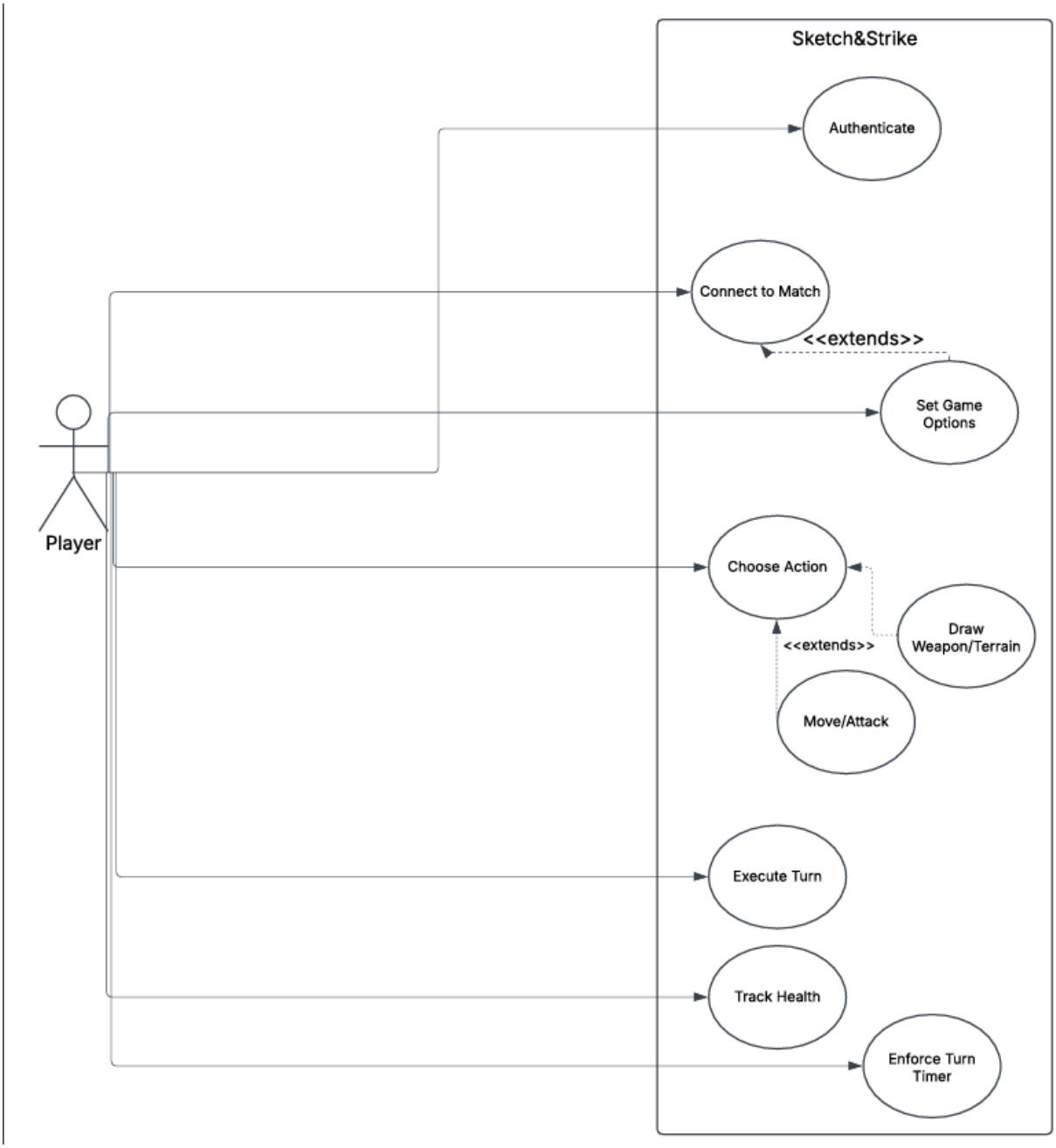
GameFunctions: This utility class provides core gameplay functionality and mechanics. Methods include update() for processing game state changes, read_pos() for handling position data, and make_pos() for creating position objects. It implements the actual game logic, rules, and interactions between game elements.

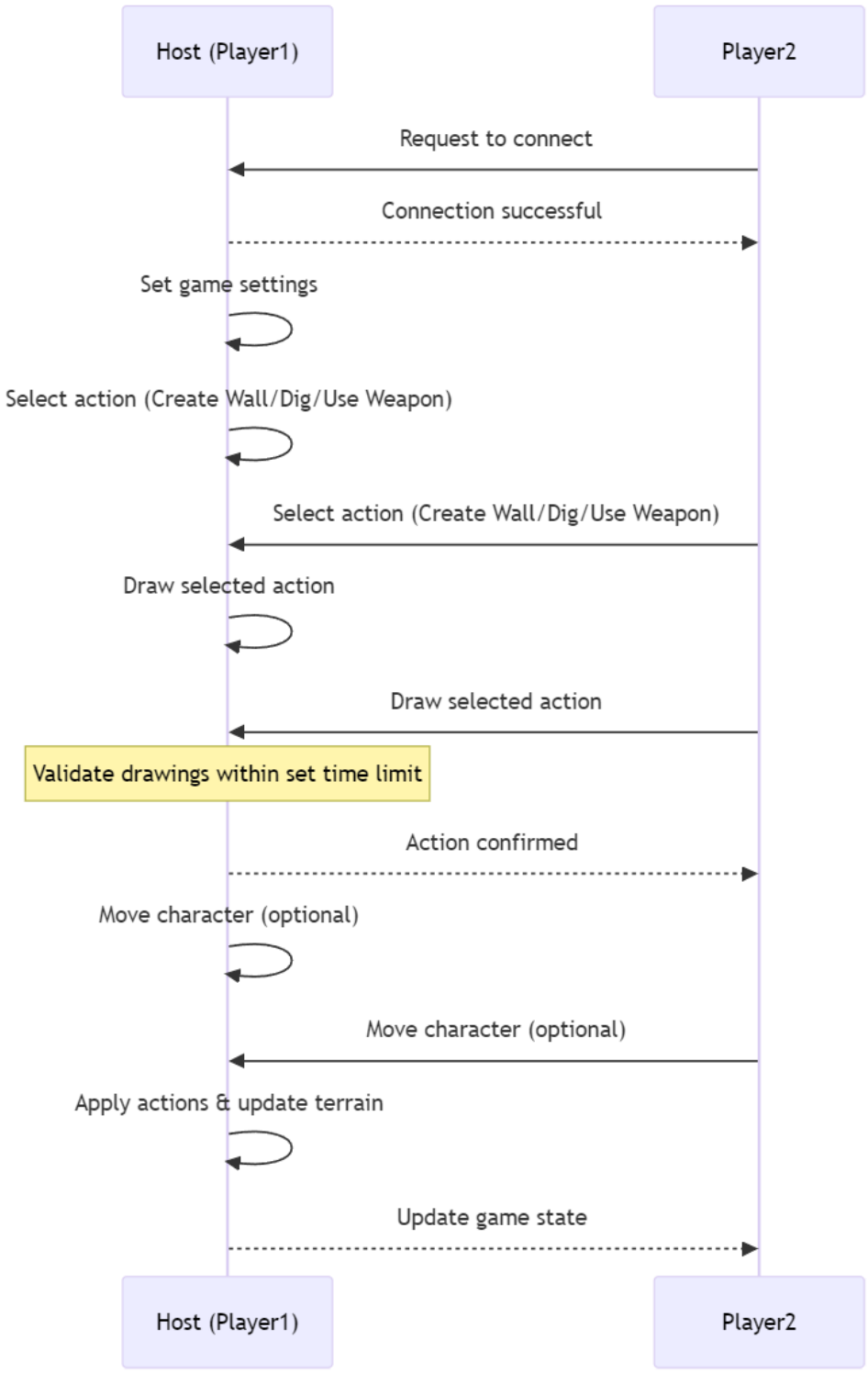
Player: This class represents player entities within the game. It tracks position information (x, y), visual attributes (color), dimensional properties (width, height), and state information. Methods include move() for position updates, update() for state changes, and `draw()` for rendering the player. The Network class uses this to synchronize player states across the network.

Button: A clickable UI element that triggers actions. It maintains visual properties (rect, text), behavior properties (color, action), and interaction methods (draw(), handle_event()). This component handles user interactions and initiates corresponding game actions when activated.

PasswordBox: A UI component for password input with protection features. Properties include rectangle dimensions (rect, color), interaction states (text, active), and methods for handling user input events (handle_event()). This likely masks password characters and validates user credentials.

5. **Class Diagram and/or Sequence Diagrams (15 points)**


Sketch&Strike
Authenticate
Connect to Match
<<extends>>
Set Game Options
Player
Choose Action
Draw Weapon/Terrain
<<extends>>
Move/Attack
Execute Turn
Track Health
Enforce Turn Timer



---

6. Operating Environment (5 points)

The application should work on windows, macOS, and Linux. Players should be able to connect to each other regardless of their operating system as long as they have the game downloaded.

7. Assumptions and Dependencies (5 points)

List any assumed factors (as opposed to known facts) that could affect the requirements stated in this document. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.

Network stability will be crucial for synchronization between players.
Graphic rendering must occur smoothly for accurate representation of user input.

By the end of increment 1, you are expected to have a first version of the functional requirements (something that the system shall do) and non-functional requirements (any requirement referring to a property of the system, such as security, safety, software quality,
performance, reliability, etc.) specified, the use cases involved in the current state of your project and a preliminary version of the class diagram or sequence diagram.
b. By the end of increment 2, you are expected to have an updated and revised version of your
requirements, use cases, class diagram or sequence diagram, as well as the textual descriptions of all the use cases in your use case diagram.