Connect Four Software Requirements Specification

Steven Nguyen, David Vessels, and Hoang Chau

Version 0.0

Finalized 06.19.2022

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SRS is to outline a GUI Connect Four Application. The intended audience includes the development team, the professor(CSE 350 Intro. To Software Engineering), and the "clients".

1.2 Scope

The product name: Connect Four

Connect Four will be a desktop application allowing for:

- **Human vs Human, Human vs AI, and AI vs AI gameplay with a Graphical Interface representing the game "Connect four".**

Intended use for this application is primarily to serve as a windows game application. That said, some secondary uses include: a reference for future application projects, the application of a new c++ graphic library for future reference/documentation, and lastly for general educational purposes.

1.3 Definitions, Acronyms, and Abbreviations.

App - Application

GUI - Graphic User Interface

SRS - Software Requirements Specification

AI - Artificial intelligence here will refer to the algorithm or bot being applied to emulate a player.

1.4 References

IEEE 830- 1993

1.5 Overview

This document is laid out in a modified IEEE 830-1993 style. The following information is in this document:

- Section 2: General description of product, including system, hardware and software interface, dependency analysis, and apportioning of requirements.
- Section 3: Functional & Non-functional requirements. Note that all non-functional requirements are grouped into one category, and not distributed among categories, as in IEEE830. Section 3 also covers use cases and deleted requirements.
- Section 4 describes the requirements analysis process.

2. The Overall Description

This software will be a desktop app representation of Connect Four. Within the primary stage of development - Development surrounding necessary client requested features, the app will consist of two pages.

The Home Screen Page will allow for user customization of gameplay. Options include gameplay modes (Player vs Player, Player vs AI, and AI vs AI), Connect Four Token colors, Starting player selection, and AI difficulty.

On the Play Screen Page there are four major groups of elements:

- Game Board: A Connect four board where the user or AI will place tokens on their turns.
- Game Log: Will log player turns with the ability for user review. This will also be the location for turn notifications, placement warnings, and win announcements.
- Game Scoreboard: Wins will be logged here between players as long as the user remains on the Play page.
- Game Buttons: Buttons include “Undo Move”, “Restart Game”, and “Home Screen”.

2.1 Product Perspective

This Connect Four Software is self-contained as a desktop application.

Comparing this software to other implementations of Connect Four virtually, the gameflow should be relatively similar to the standard game. The primary difference in implementation should be the graphics, user interaction feel, and low cost performance for the application to run.

2.1.1 System Interfaces

N/A

2.1.2 Interfaces

The application will interact with users purely via GUI with no special interface requirements. This application will be limited to windows systems x64 that are compatible with c++ 20.

The software should be extremely cost efficient in terms of performance. There will not be any required optimization.

2.1.3 Hardware Interfaces

The application will purely use mouse controlled interaction.

2.1.4 Software Interfaces

Name: **Visual Studio 2019 (c++ 20 community edition)**

Mnemonic **VS**

Specification number **N/A**

Version number **2019**

Source **Microsoft**

This software will be the development environment used to build the application.

Name: GitHub

Mnemonic: N/A

Specification number N/A

Version number Enterprise Server 3.5

Source Microsoft Corporation - GitHub Inc

This software will be used for source control within the development team.

Name: SFML

Mnemonic

Specification number N/A

Version number SFML 2.5.1 -> Visual C++ 15 (2017)-64-bit

Source SFML Team

This is an external low level GUI library. This library is a dependency to the higher level graphic library we will be using. It is responsible for the creation of the window which the application will be built on.

Name: SNWidgets

Mnemonic

Specification number N/A

Version number N/A

Source Steven Nguyen

This is a mid-level GUI library which is responsible for the creation of the various pages, and widgets used within the application.

2.1.5 Communications Interfaces

As the client specifications did not include online/remote in the necessary requirements, no communications interfaces are listed.

2.1.6 Memory Constraints

Through review of other publicly available Connect Four applications, we have found a range of 20MB to 84MB for application size. From this we have determined the design should not exceed the upper range of 84MB. As the application furthers, this value should be re-evaluated to be closer to the minimum value in the researched range.

2.1.7 Operations

This application will only have one user class: the player class.

On the homepage of the application, the user will use various widgets to modify game options.

On the play page of the application, the user is responsible for dropping tokens on their turn. As well, there will be buttons that include “Undo Move”, “Restart Game”, and “Home Screen” to navigate between pages and the game state.

N/A no data will be saved between application launches.

2.1.8 Site Adaptation Requirements

N/A

The application is self contained and requires no adaptive changes to operate.

2.2 Product Functions

User interaction with the application will be split between the Home page and the Play page.

Home page - The user has various buttons controls to change game presets prior to starting the game. Presets include:

- **Player1 and Player2 types: either Human or AI/Bot**
- **Player1 and Player2 colors: will be listed in dropdown**
- **Which player starts: Player1 or Player2**
- **If AI/Bot is used: difficulty options between easy, medium, or hard.**

Play page - The user will be able to place during their turn using their mouse above the game board.

The user will also have access to the following widgets:

- **Home button (takes the user back to home page**
- **Restart Game (clears the board, starting a new game)**
- **Undo Move (reverts the last played move)**
- **Game Log (contains turn history and game announcements)**
- **Score Board (contains winning history for players)**

2.3 User Characteristics

Users must be able to operate a windows computer. As well, the game will not come with a tutorial. Thus, it would help if the user has basic knowledge of Connect Four.

2.4 Constraints

Copyright for potential background music.

Maximum of 84MB disk storage required.

Design choice to use c++ 20.

2.5 Assumptions and Dependencies

All dependencies are listed in categories above.

2.6 Apportioning of Requirements.

Build V.1:

1. A Home page that allows the user to change the following presets:
    - AI/Bot difficulty level: easy, medium, hard
    - Player1 and Player2 type: Human or AI/Bot(option will be fully implemented in Build V.2)
    - Color1 and Color2: either red and black
    - Starting Player: either Player1 or Player2
    - A Start button to open the Play_Page
2. A Play page that contains a game board that allows the user to place tokens (No turn or game system implemented). As well, the following buttons should be implemented:
    - Home Screen button: opens home page.
    - Restart Game button: clears the board.
3. A Play page that allows for Human vs Human gameplay. This includes a working Game Log, win/lose announcements, and the undo button.

Build V.2:

1. Finish the implementation of Human vs AI, AI vs AI (The AI will play randomly).

Build V.3:

1. Remote multiplayer gameplay. Add a new Main page that allows selection between offline and multiplayer mode. Offline mode will bring the user to the previously made Home Page - nothing will change in this instance. Multiplayer will connect the user to the Connect Four Server.
2. Game board syncing between two users. (No turn system will be implemented at this time).
3. Implement a turn system, Game Log, and Scoreboard.

3. Specific Requirements

3.1 External interface requirements

This section specifies the user interfaces to the system. All user interfacing is done through a Graphic UI. The pages are shown below.

3.2 Classes/Objects

This section lists classes constructing the application.

3.2.1 SF.RenderWindow

This class is responsible for displaying the application window

Attribute: No outstanding attributes N/A
Functions:

- void draw(object) //Draws SF objects and images to the app window
- void clear() //Clears the app window
- void display() //Copies render buffer to screen

3.2.2 SNPage

SNPage contains page elements and display functions. All graphics are written through here.

Attribute: vector<SNElement> elements, vector<SNElement> post_elements
Functions:

- void draw_page() // All displays are written through here.
    - main_window.clear() // Clears application screen
    - for loop to draw all elements and post_elements
    - main_window.display() // Copies graphic buffer to app screen

3.2.2 SNElement

SNElements are contained within SNPages. The elements are responsible for the user interaction with the application. This is a parent class for various element types including: SNButton, SNLabel, SNContainer, GameLog, and Board**.**

Attribute: double x y w h, and string type
Functions:
- virtual void draw_element() // each child object of element will have a unique drawing method.

3.2.3 SNButton : SNElement

SNButton is a child of SNElement. It handles clickable type elements.

Attribute: bool is_selected, void (*function)()
Functions:
- void draw_element() // Unique draw method

3.2.4 SNLabel : SNElement

SNLabel is a child of SNElement. It handles titles like display.

Attribute: string str
Functions:
- void draw_element() // Unique draw method

3.2.5 SNRadio_Button : SNElement

SNRadio_Button is a child of SNElement. It handles multiple choice type scenarios.

Attribute: N/A
Functions:
- void draw_element() // Unique draw method (Includes imaging of a radio_button symbol).

3.2.6 SNContainer : SNElement

SNContainer is a child of SNElement. It stores other elements. Generally will be used for post application launch element creating and usage.

Attribute: vector<SNElement> elements, string title

Functions:
- void draw_element() // Unique draw method

3.2.1 GameLog : SNElement

SNGameLog is a child of SNElement. It stores game events to be displayed specifically on the play page of the application.

Attribute: vector<string> text
Functions:
- void draw_element() // Unique draw method

3.2.1 Board : SNElement

Board is a child of SNElement. The Board class will be the board displayed on the play page. It will include turn handling, GameLog writing, and other board manipulation methods.

Attribute: int[6][7] board, GameLog log
Functions:
- void turn_controller(string) // Buttons will send strings to this turn controller to process and display board changes.
- void reset_board() // Clears the board and game log.
- void undo_move() // Refers to game log to undo the previous turn.
- void draw_element() // Unique draw method.

3.3 Performance Requirements

N/A.

Considering the size of the project and that the application is written in a highly efficient language (c++), there aren't any benchmark concerns.

3.4 Design Constraints

The development team is only constrained by the development tools chosen. This includes the use of the SFML library, SNWidgets, c++ version 20, visual studio 2019, and github.

3.5 Software system attributes

The software is built for windows systems capable of running x64 and c++ version 20.

3.6 Other requirements

N/A

4. Change Management Process

Between the specified development versions, github will be used to log developmental changes. At the completion of each development version we can then re-evaluate the position of the program. This re-evaluation should contain communication with the customer on the “whole” number development versions at minimum.
After consulting with the customer the project lead, Steven Nguyen, will distribute customer notes for re-evaluation to the team. The project lead will make final decisions on the requirement changes.
Note that the project lead has been decided based on highest experience within the outlined development tools.
Upon completion of a “whole number development version, consultation with the customer and the development team, a new SRS with notes on requirement changes will be submitted as a part of the github commit.

5. Document Approvals

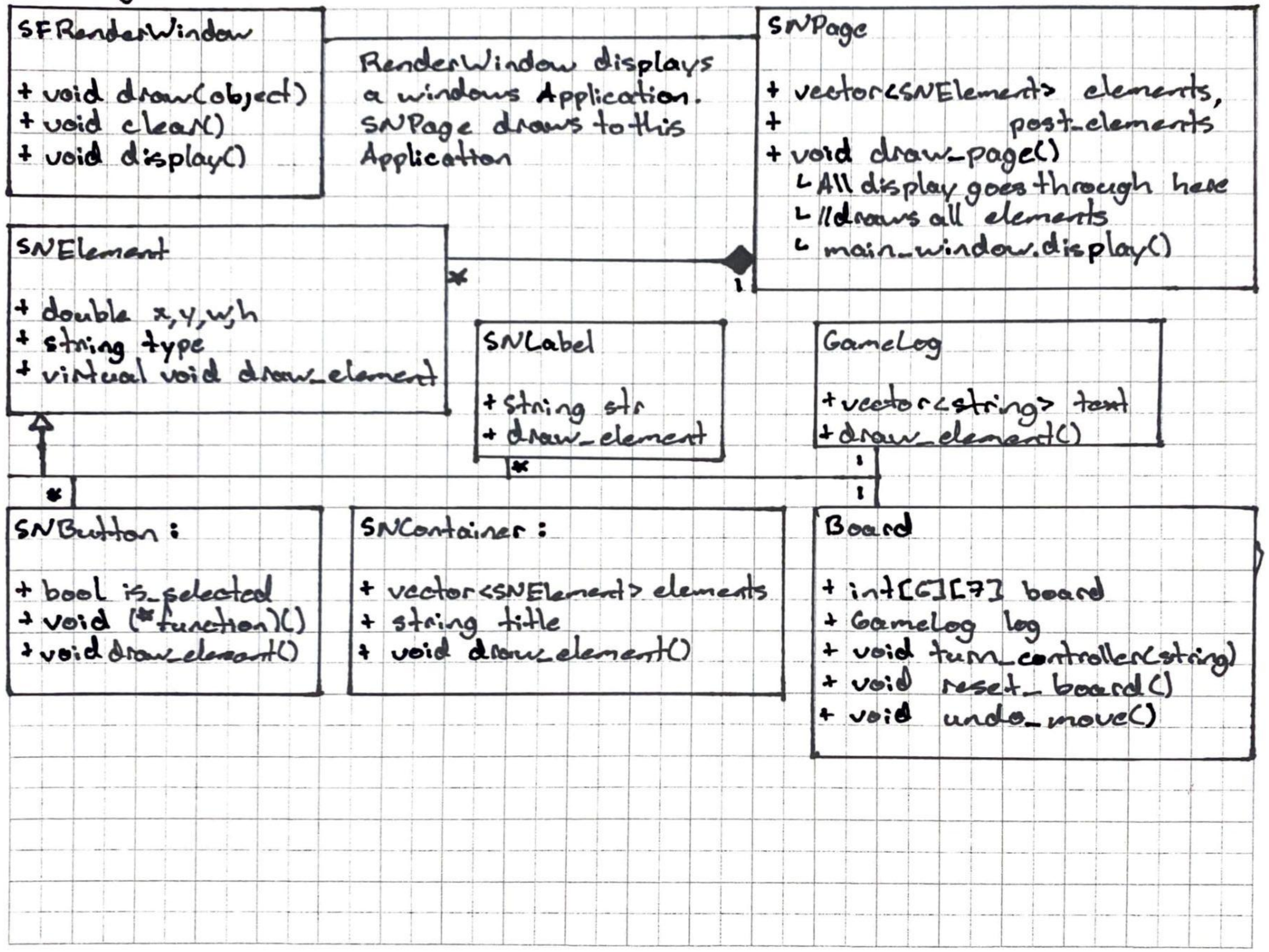
Documents should be approved by the CSE 350 professor: Dr. Harry Zhang

6. Supporting Information

The supporting information will include reference diagrams from the diagrams. Note that these diagrams should be included in the SRS. The UML and Sequence Diagram is attached below.

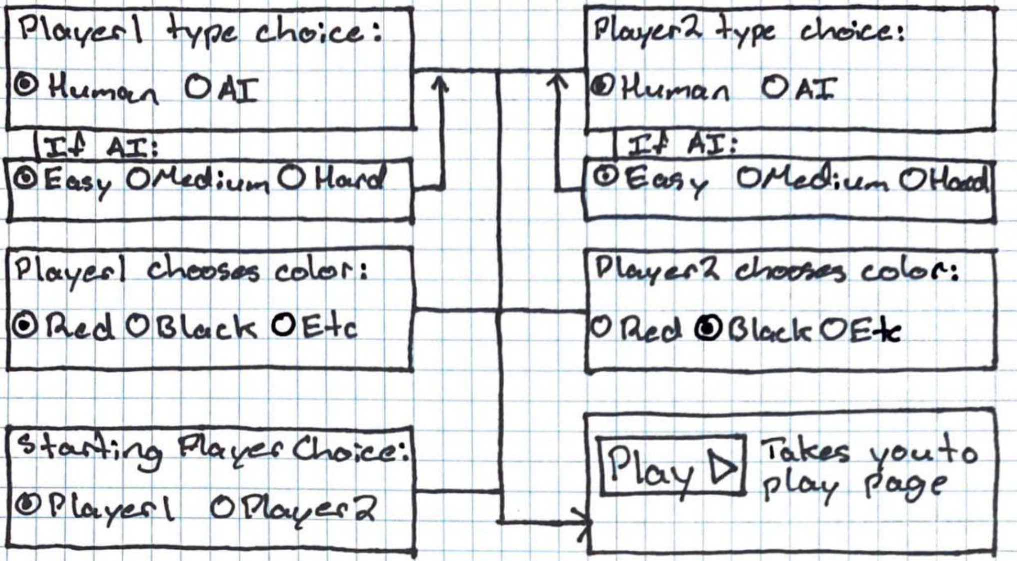
UML Diagram

Figure #1



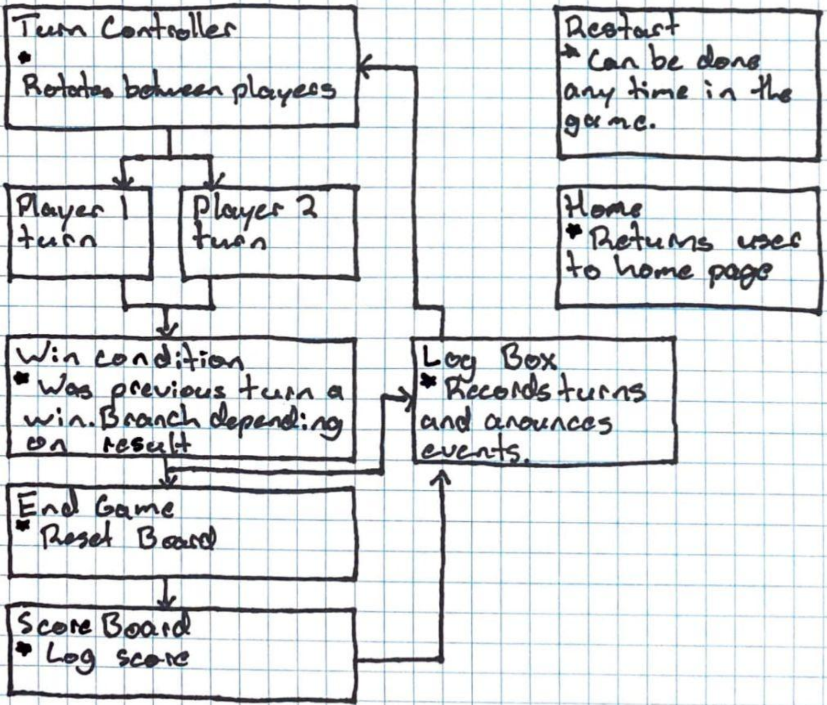
Sequence diagram. Home page: Figure #2

*Interactions are represented withe Buttons or Radio_Buttons



Play page:

*This flow is more so game sequence flow.


Turn Controller
* Rotates between players
Player 1 turn
Player 2 turn
Win condition
* Was previous turn a win. Branch depending on result
End Game
* Reset Board
Score Board
* Log score
Log Box
* Records turns and anounces events.
Restart
* Can be done any time in the game.
Home
* Returns user to home page