CSCE 315 Project 2 Design Document

Harry Zhang

Mathew Horton

Katrina Tuazon

Jacob McIntosh

**Purpose**

The ultimate goal for this project is to create a program that can successfully run a functioning Reversi game. The requirements for this program include the ability to pit two Artificial Intelligence against each other, the ability for a user to play against an AI, and the ability to pit two AI over the network against each other. A functioning Graphic User Interface is required on the client side to display the state of the game and provide game options including, but not limited to, new game, AI difficulty, and undo. This program will also require a server side that any game palyer can connect to. The client needs to be as thin as possible so the AI and all of the logic within the game should be on the server side.

Beyond the immediate deliverables of this project, the purpose is to become familiar with a SCRUM development environment. SCRUM is used quite frequently as a code development technique thus it is important to be familiar with it. Additionally, iterative development methods like SCRUM can be more powerful than the traditional waterfall method used especially when coding as it provides a much more flexible development environment.

**High Level Entities (Interaction and Description)**

The beginning of the interaction between the high level entities is between the client and the server. All of the game code is contained within the server, and the server is responsible for executing the game and keeping the game state. On the other hand, the client is responsible providing input to the server allowing the game to properly function. The threads within the server will each be responsible for serving a specific client. They will wait for input and the calculations will be made determining the current game state. Finally, a “move” will be executed and sent back to the client.

For the server to calculate a game state and to execute a move it will rely on the other two major entities, the game and the AI. The client’s interaction with the game and AI entities is relatively limited in that it never directly interacts with either entity, but acting through the server the client entity does interact with both the game and AI. The input the client gives to the server will modify the game entity which will in turn cause the AI entity to “react” differently in that a different optimal move will be calculated. Thus the client’s action propagate to both the game and the AI and actually drive both of them.

* + **Server**
    - **Sockets**
      * Sockets will be used to communicate with the client.
    - **Child Processes**
      * Each individual child process will handle
  + **Client**
    - **GUI**
  + **Game**
    - **Board**
    - **State**
  + **AI**
    - **Search Algorithm**

**Low Level Entities**

* **Server**
  + - **Sockets**
    - **Child Processes**
  + **Client**
    - **GUI**
  + **Game**
    - **Board**
    - **State**
  + **AI**
    - **Search Algorithm**

**Benefits, Assumptions, and Risks (5-6 top benefits of the design, a list of ALL known risks/issues and a list of ALL assumptions.)**

* + **Benefits**
    - Provides multi-threading on the server and allows simultaneous play for performance reasons
    - Varying search parameters and different searching methods makes it easy to set and change game difficulties
    - A more user-friendly user interface is provided the Graphic User Interface
    - Maintainability and debugging is made easier through the use of various classes
  + **Assumptions**
    - The server running this program is in an Unix environment, and it capable of producing and running more than one thread
    - The environment running this program is capable of graphics
  + **Risks**
    - Cheating and interference is a possibility since the communication between the server and user is not encrypted
    - The fork() command is only available on Unix based operating system, but the server is very reliant on the fork command.
    - The game cannot be played purely from the command line; it can only be played through the Graphic User Interface. Users that do not have access to a function windowing system will be unable to run this program.

**Post Production Notes (Difficulties, Design Changes, and Lessons Learned)**

* + **Difficulties and Solutions**
    - Trying to understand how to create an AI to compete against the player proved to be challenging. We had no idea what an AI would look for in making moves other than how many of the opponent’s pieces it could jump. After researching Reversi strategy, we came to the conclusion that using a weighted board to help determine the value of a move passed the number of jumps it could make.T This weighted system put emphasis on controlling the corners ahead of jumping the opponent’s pieces. The primary difference between the hard and medium AI’s, on the other hand, is simply the depth of the min-max traversals that it makes in considering moves.
    - Client-GUI Design
  + **Design Changes**
    - Choosing the GUI designer (FLTK, QT, JAVA, etc.) was a frustrating prospect. Many of us were familiar with FLTK and we knew it would work on the Unix systems, but we also knew that FLTK was especially prone to Linker errors should it be implemented incorrectly by the compiler. JAVE was an early choice that was rejected so that we would stick with the same code (C++) throughout the entirety of the project. We switched from using QT to using FLTK and back to QT before we finally were able to settle with the designer that was most effective.
    - The server design was especially subject to change in order to simplify the entire project. There are now no forking calls, no sub-processes, just that basic functions and socket implementation that we gave it.
  + **Lessons Learned**
    - Importance of work logs and constant group communication
    - Learning and understanding a designer prior to use and implementation