Milestone 1: Design

Justin Parratt (jwp258), Brian Shi (bcs84), Kirk Thaker (kt485), Ram Vellanki (rsv32)

1 System Description

We plan to develop a multiplayer (distributed) Scrabble game that can be played with either human or computer players.

Key features:

- AI
- GUI
- multiplayer (distributed)
- English (not OCaml) dictionary manipulation using a Trie

We will make a Scrabble game which allows users to play Scrabble, but with added features such as detecting if words are valid or not. We also plan to implement an AI to play words that maximize score according to tile/word bonuses and point values of letters. The AI will also have different levels of difficulty. One way to implement this might be choosing suboptimal words for lower difficulty levels.

These features will be implemented by using a prefix tree because the efficiency of the data structure is especially important for the AI.

For the server-client interface, we will leverage several OCaml packages (listed in the **External Dependencies** section of this document) to persist data across multiple players and multiple instances of games as well as provide an HTTP interface for multiplayer functionality over the internet. Our server will be exposed over a public IP address such that any system on Cornell's network can access and play our game via our web application.

For the user interface, users should be able to view their current available letters as well as the current board and should be able to perform moves. We plan on implementing a text interface to start with, and upgrading it to a graphical interface if time permits.

With regards to the comment on the usefulness of a trie - we felt that a trie was necessary because the AI would be doing lookup of many words when evaluating potential moves. A trie would also allow the AI to quickly determine what words can be made from an existing word on the board by adding some additional tiles to the end of the word. This is elaborated in more detail in the **Data** section of this document.

2 Architecture

There are three primary components (units of execution) in this system.

- Server - provides an interface for accessing/modifying game state over HTTP
- Client (Browser) - provides an interface for playing a game through a GUI
- Database - provides a persistent storage mechanism for saving information of multiple games and multiple players

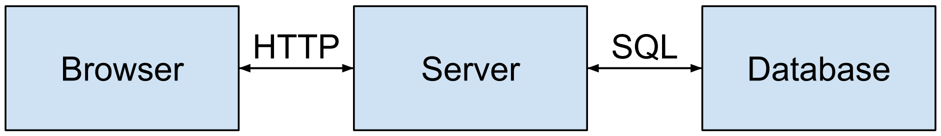

Fig. 1: Component and Connector Diagram

3 System Design

Please see the next page for the module dependency diagram (MDD).

Grid - 2-D representation of board characters (including coordinates of score tiles)

Dictionary - English dictionary stored in a data structure (implemented as a Trie)

AI - logic to generate a move based on an input state (uses **Dictionary** to find a valid move)

Game - logic to perform actions in a game, such as adding/removing players and performing a move (using **Dictionary** to validate)

ORM (object-relational mapping) - accepts OCaml records and stores them in the appropriate tables of the database using SQLite (i.e. acts as a virtual object database)

GUI - converts GUI input (from the browser) to moves, uses the Client to process those moves, and reflects changes in the GUI by updating a graphical scrabble board (no .mli file provided because it is the entry point of the application)

HttpServer - an abstraction of **cohttp** for simpler development of HTTP server APIs

XHRClient - an abstraction of **cohttp** for simpler development of XML HTTP requests

Server - accepts moves in the form of HTTP requests then uses Game to produce a new game state, returns the changes, and saves changes to the database (no .mli file provided because it is the entry point for client requests but see documentation below this list of modules); this module will issue Server-Side Events to clients as a means of more efficient updates for game state

ScrabbleClient - communicates with the Server's HTTP API to join games, create games, retrieve the changes in game state etc. (i.e. serves as an abstraction to make the separation between client and server seamless); this module will also leverage Server-Side Events for more efficient updates of game state

In order to provide a more comprehensive design of our HTTP API (specifically the **Server** module above), we have developed an outline of our initial API design: http://docs.scrabble1.apiary.io

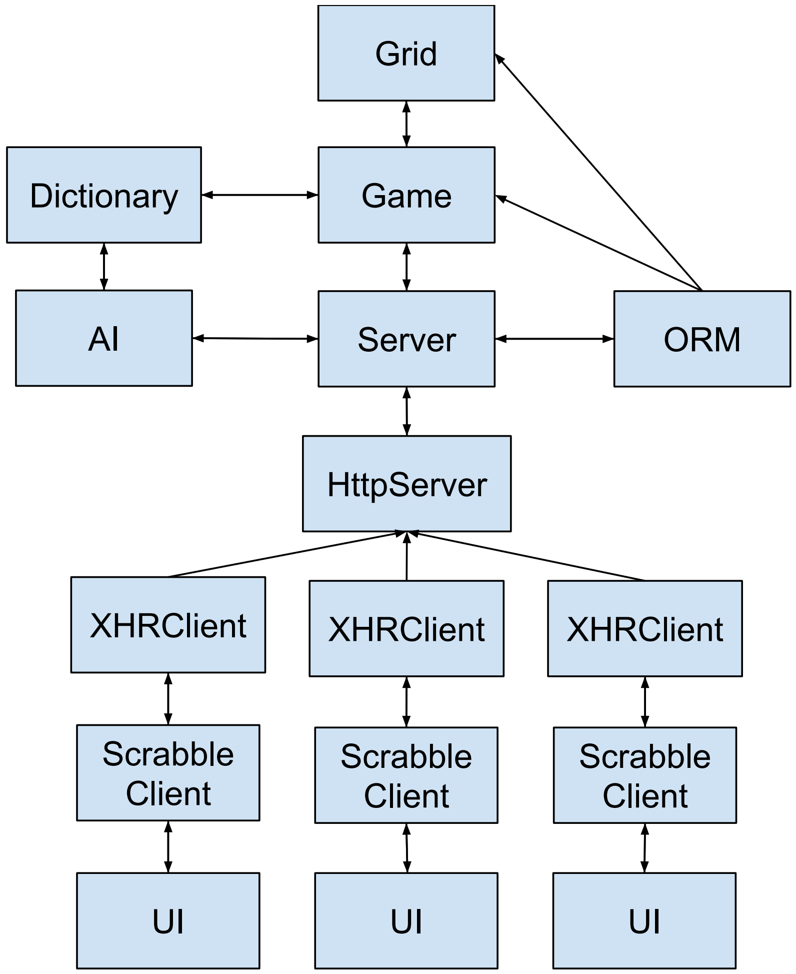

Fig. 2: Module Dependency Diagram

4 Module design

The following .mli files have been included: ai.mli, dictionary.mli, game.mli, grid.mli, httpServer.mli, oRM.mli, scrabbleClient.mli, and xHRClient.mli.

5 Data

The system will maintain persistent data for game information including its players, remaining tiles, turn counting, and of course the current board state. Here are some of the more intricate details of the data representations.

- Database - information will be stored in SQL tables as relational data
- HTTP API - data will be transferred as JSON
- Dictionary - a trie will contain the dictionary data (the efficiency of this data structure is particularly useful for the AI). Specifically, we will insert the English dictionary into two different tries - one where words are in normal order, and one where words are

in backwards order (e.g. OCaml would be inserted into the trie as l-m-a-C-O). The forwards-trie would be used for word validation as well as for the AI to find words where letters can be added onto the end of an existing word. For example, adding "d" onto the end of "dance" yields the new word "danced". The backwards-trie would be used for the AI to find words where letters can be added onto the front of an existing word. For example, adding "un" onto the front of "balanced" yields the new word "unbalanced". This cannot be done with the forwards trie, since it would be very slow to figure out the words that have "balanced" at the end.

6 External dependencies

OCaml Packages

- ounit - for unit testing
- lwt - for concurrent programming
- sqlite3 - for persistent storage of data
- cohttp - for the HTTP client and server
- yojson - for serializing/deserializing data to and from JSON
- js_of_ocaml - for developing a browser-compatible GUI

Other Dependencies

- Node.js (with npm) - for serving the OCaml GUI (produced by web technologies along with **js_of_ocaml**) and routing HTTP requests to the OCaml server running on a local port (all on a machine on Cornell's network)

7 Testing plan

All the non-web modules (e.g. board, dictionary, ai, and game) will be tested using black-box and glass-box unit tests. To ensure accountability, the black-box tests for a module will be written by someone who did not work on it (i.e. cross-testing). The GUI will be tested by alpha and beta testing including individuals outside of CS 3110. The HttpServer and XHRClient modules will be unit tested against mocked servers or standardized URLs (such as google.com). The remaining modules (e.g. ScrabbleClient, and Server) will be tested upon integration to the primary system.