Andrews University

Wordi: Fast and Scalable Word Game AI with Extensible Messaging and Presence Protocol (XMPP) and Dynamic Programming

Kendall Hopkins, Eric Scott, Stephen Oxley, Danny Ruiz
Department of Engineering and Computer Science
Andrews University

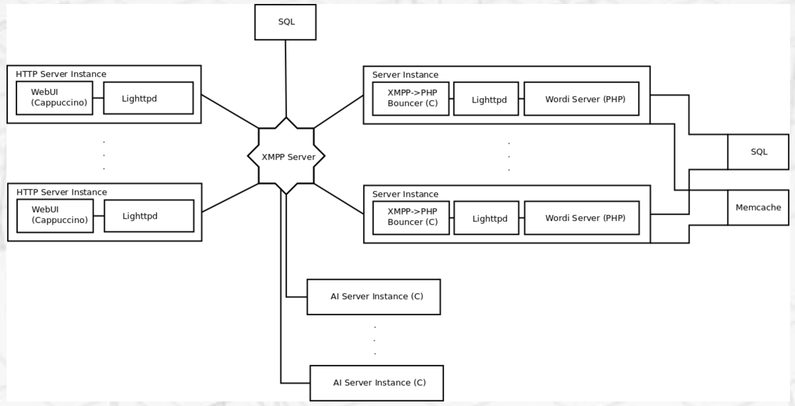
Advisor: William Wolfer

Abstract:

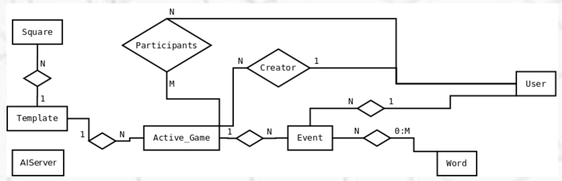
We developed a multiplayer word game for the Internet, called Wordi, with a scalable server-side infrastructure and a fast AI engine. This system contains elements that improve upon the state of the art for large-scale deployments of Scrabble(tm)-like online word games. The server system uses the Extensible Message and Presence Protocol (XMPP), formerly known as Jabber, to manage asynchronous messages from clients. The AI agents act as clients and use a series of heuristics to expand the agent strategy beyond the naïve greedy method. A dynamic programming approach is applied to efficiently calculate valid plays checked against a dictionary, which is stored as a Directed Acyclic Word Graph (DAWG). We present a novel algorithm for converting between a prefix tree and a DAWG. The Graphical User Interface for human players is written in the JavaScript-based Cappuccino Web Framework, and is browser and operating system independent.

Server Infrastructure:

Server Infrastructure: Events (plays, disconnections, etc) can potentially be generated by any client at any time, making the communication between client and server inherently asynchronous. To handle this, we provide message-oriented middleware (MOM) via the XMPP, a multi-purpose generalization of the Jabber instant messaging protocol. The XMPP server scales to handle thousands of parallel connections on a single machine. Game states are controlled by PHP instances running on potentially separate servers. We wrote an 'XMPP-PHP bouncer' in C to translate between XMPP and the HTTP requests PHP understands, allowing PHP to serve as a backend for a bidirectional event system.

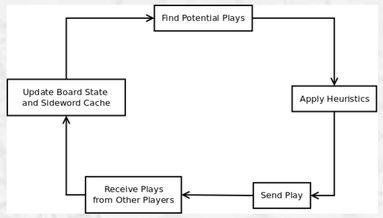


The database accessed through the PHP server is organized around users, active games, and events. A game state is the sum result of all events associated with a game. Boards are predefined in templates consisting of board dimensions and "dead" and "multiplier" (bonus) squares. The authoritative dictionary is stored in the *Word* table, and certain events may be associated with one or more words. The database also stores references to active AI servers that can provide non-human players for new games.

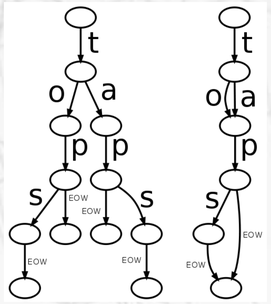


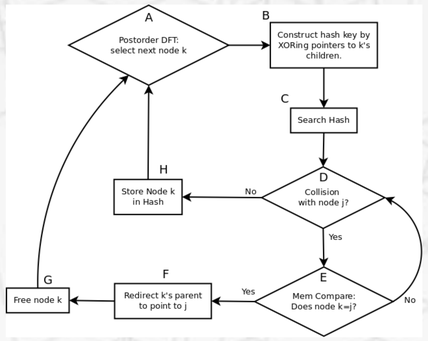
AI:

The AI uses a single-threaded eventing system to manage games, initiate segments of the game cycle (ex. turns, board updates), and process events for active agents while other agents are waiting on the server

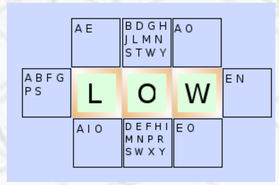


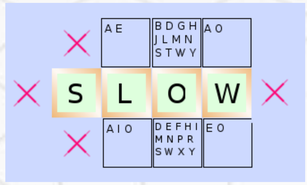
At the beginning of a game the AI agent downloads the dictionary from the server and expanded into a prefix tree (trie), which is easily searched for valid words matching a certain pattern. The prefix tree is then converted into a Directed Acyclic Word Graph (DAWG) via a novel O(n) algorithm, reducing the dictionary's memory footprint by ~90% without requiring modification of the traversal algorithm.





The play finding process consists of traversing the board, finding words that will fit in playable squares, and ensuring that "sidewords" created orthogonally to the played word are valid. To accomplish this efficiently, two caches are stored in each playable square indicating what letters can be placed there by horizontal and vertical plays, respectively, to make valid sidewords. The cache for a square can be checked quickly via bitwise comparison operators while playing across the square, and only needs updated when words adjacent to it are changed by a committed play. In this way the sideword checking process goes from being the most computationally intense part of a naïve implementation to being negligible in a dynamic programming approach.

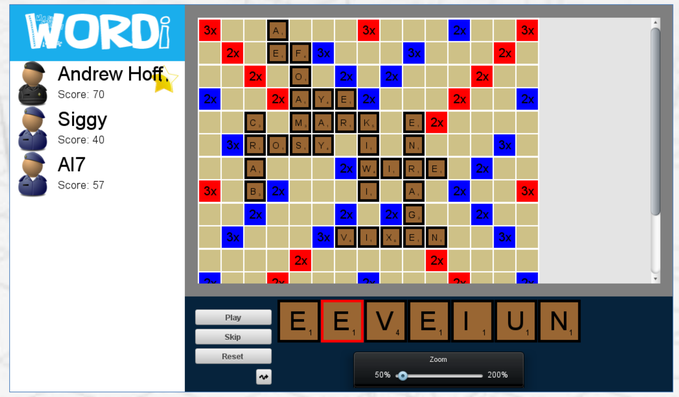




Finally, some improvement over the greedy method is offered by a series of heuristics based on facts such as the ratio of vowels to consonants in the AI's hand, which may lead to playing a word that doesn't maximize the short-term score, or may induce the agent to discard its hand and draw a new one. The weighted values assigned to the heuristics are determined by optimizing performance over many trial games -- ex. by simulated annealing or genetic algorithms.

GUI:

The web interface is coded with the Cappuccino Web Framework (http://www.cappuccino.org), which provides support for full object-oriented programming in JavaScript via Objective-J, and abstracts the development of web applications away from the HTML/CSS layer. Our code structure uses the Model-View-Controller (MVC) architectural pattern to keep the application logic clean and facilitate concurrent front-end/back-end development by multiple programmers.



Future Work:

- The system's performance bottleneck is in the PHP server that manages database queries. The server was written before we implemented the XMPP paradigm, and could be rewritten in C++ to improve performance. At the very least, replacing standard PHP with HipHop (Facebook's PHP implementation) is worth considering.
- The events are currently encoded via JSON, whose syntax does not play nicely with XMPP's escape characters. XML-RPC is a remote procedure call protocol that would be a cleaner solution than JSON, cost less CPU for encoding/decoding, and potentially require less bandwidth.
- Even though the Trie to DAWG algorithm is novel and clearly useful in certain applications, for this project it would probably be more efficient to generate a DAWG directly, since we will never need to edit the dictionary in-game.
- The best word game AI's can look ahead a limited number of moves to determine better strategies and maximum-scoring end-games, much like in chess. We have experimented with tackling this NP-hard problem, but have not come up with a full-scale implementation. Also, to our knowledge no Scrabble-like game AI uses machine learning to classify and predict its opponent's strategies.

People:

Kendall Hopkins
Principal Meisterator
Server/AI/GUI

Eric Scott
AI, Sideword Caching, Documentation, Poster

Stephen Oxley
AI, Trie to DAWG, Heuristics

Danny Ruiz
GUI