Assignment 5 - Distributed Connect4, Augmented Design Summary

This document summarizes the rationale behind all augmented design decisions and contract deviations in regard to our previously submitted Distributed Connect4 design document (parts 1 and 2). In addition, this document outlines the test framework we have designed, beyond contract descriptions, and lists all known errors, faults, defects, and/or missing functionality.

Augmented design decisions

For the most part, the design was not deviated from. We modified the format in which we stored data in the database and the way in which we check for game outcomes. Game outcome checks were moved to the server.

Contract deviations

For the most part we did not deviate from initial contracts proposed in part 2. No contracts were modified; however, more contracts were made for the client.

A deficiency in this area was that no contracts were explicitly provided for the UI layer. In order to test other sections, the UI layer was explicitly mocked. This is because the GTK library, given its C nature, does not provide an API for mocking user events. Therefore, GUI testing was conducted manually over the course of the application development.

Test framework

Our test framework utilizes the rubygem `test-unit` and serves to verify the representation and manipulation of the gamestate as well as the recognition of winning states. Originally, we hoped to include a suite of automated of UI tests; however, we were unable to do so before the deadline. As a result, we resorted to manual UI testing using informal checklists. At the time of submission, all tests were passing.

Our automated tests are located in the test folder. The test architecture is designed according to a random testing scheme, whereby game-states or game-types are generated dynamically and run many times. These tests were designed to use mock databases and uis in order to decouple code in the test cases. The full suite was run within a continuous integration pipeline upon each commit.

Additional tests were created in db_test that are not a part of our automated tests, these tests create a test server and perform all client requests and assert based on the semi-random data within the server.

As with any graphical application, it is important to ensure UI functionality. In the absence of automated UI testing, we instead ran through a series of informal checklists. These tests were developed according to common gameplay scenarios. For example, one manual test executed had the following steps:

1. Configure an online Toot/otto game on two separate machines
2. Play the game until a winner is determined -- at some point during the match exit the client (on one machine) and restart the client, continuing the game where it was left off
3. Verify that both clients are notified of the correct winner
4. Return to the league stats and verify that each player's stats have been updated accordingly

UI tests focused on verifying functionality from the end-user's perspective as well as general usability.

Error, faults, defects, and/or missing functionality

We provide all basic functionality for a online Connect4 or TOOT and OTTO game.

In addition to the requirements we provide the following additional features:

- Forfeiting - a player may opt to forfeit a game at any point in time
- Variable difficulty computer agents - during offline game configuration, a user may select between a beginner and advanced computer opponent