University of Hull

A Networked Poker Game and AI

Initial Report

Author:
Zachary Palmer

Supervisor:
Dr. C. Kambhampati

Submitted for the BSc in Computer Science

October 15, 2017

Contents

1 Introduction

The developed software will comprise multiple inter operable sub projects, a server, a client, and an AI.

The project will intend to evaluate the use of Haskell (Hudak et al. 1992) in creating large software projects spanning a range of areas, specifically in AI development and interconnected systems.

Poker has been chosen as the target game for this software, as due to being an imperfect information game (Sandholm 2010), is challenging for Artificial Intelligence to solve.

In addition, certain aspects of Poker are interesting to explore, for example demonstrating that random picking or shuffling of cards is indeed random, and not slightly gamed by the software.

2 Aims & Objectives

- Investigating the capability of Haskell, a functional programming language, in programming networks, stateful code, and AI
- Creating a rich GUI with a high level toolkit, allowing modular replacement of assets at users request
- Investigating a number of different Artificial Intelligence programming paradigms for poker

Motivations behind the decision to use Haskell include:

- A strong type system, allowing rapid development and refactoring
- Functional programming, which encourages terse, expressive code
- Simple and safe multi threading, due to immutable by default data types, requiring very little change to application code to take advantage of multi-core processes
- High level support for managing complex state and other side effects, using monads (Benton, Hughes, and Moggi 2000)
- High level support for updating and viewing nested state, using the "lens" library and lenses paradigm (Bohannon, Pierce, and Vaughan 2006)
- A rich library system allowing massive amounts of duplicate code to be avoided

Significant libraries to be used:

- transformers-This library provides a high level interface for managing stateful code, using monad transformers, and implicit parameters
- lens-This library allows painless updating of deeply nested structures, creating very expressive and concise code
- network-This library has a socket interface, and will be used for the client and server, offering lots of power, whilst having a very simple interface
- hsqml-A Haskell interface to the QtQuick graphics library, which allows fast prototyping with JSON syntax
- binary-A library for encoding arbitrary data types into a binary format for efficient message passing between client and server
- quickcheck-A library allowing creation of random instances of custom data types, to be used in testing functions with a wide variety of inputs

3 Background

Server

The server will host all the game logic, and will take input from the client, update its internal state, and propagate the changes to all the other clients. This section of the project will require managing a complex state, message passing and gracefully handling failure when network conditions cause clients to disconnect unexpectedly.

Generation of random numbers is an interesting topic in Poker. Computers cannot generate true random numbers, and instead have multiple methods to create "Pseudo Random Number Generators". These are intended to be implemented in such a way that they are indistinguishable from true randomness. However, depending on the implementation, they can be insecure, allowing users to calculate later numbers after observing those that have come before. Sometimes, the user can ensure they get the random number they wish for by exploiting the timing of their actions. I will approach this issue by potentially using multiple sources of random numbers, and picking suits and values separately. These methods will be examined to see how they affect the cyclic nature of random number generation, along with card distributions. Utilising multiple decks behind the scenes rather than one reused deck could also make the software harder to manipulate by users.

Client

The client code will be relatively simple, just taking simple inputs from the user, and passing it to the server, and interpreting the servers responses. It will, however, take significant effort to make a good looking GUI, potentially featuring animations and different themes. Due to Haskell having immutable state by default, which is poor for many graphics libraries, I have decided to use the QtQuick graphics library, which uses JSON to describe a UI. The interaction between Haskell and QtQuick is minimal and easy to use.

AI

As mentioned, AI is a very interesting subject in poker, which has been researched heavily. The most advanced Artificial Intelligence algorithms are currently approaching pro level game play. Multiple different approaches will be attempted, such as a rule based AI (Watson and Rubin 2008) to begin with. After, more advanced techniques could be implemented, such as Bayesian probabilities (Billings, Davidson, et al. 2002), or neural networks (Teofilo and Reis 2011).

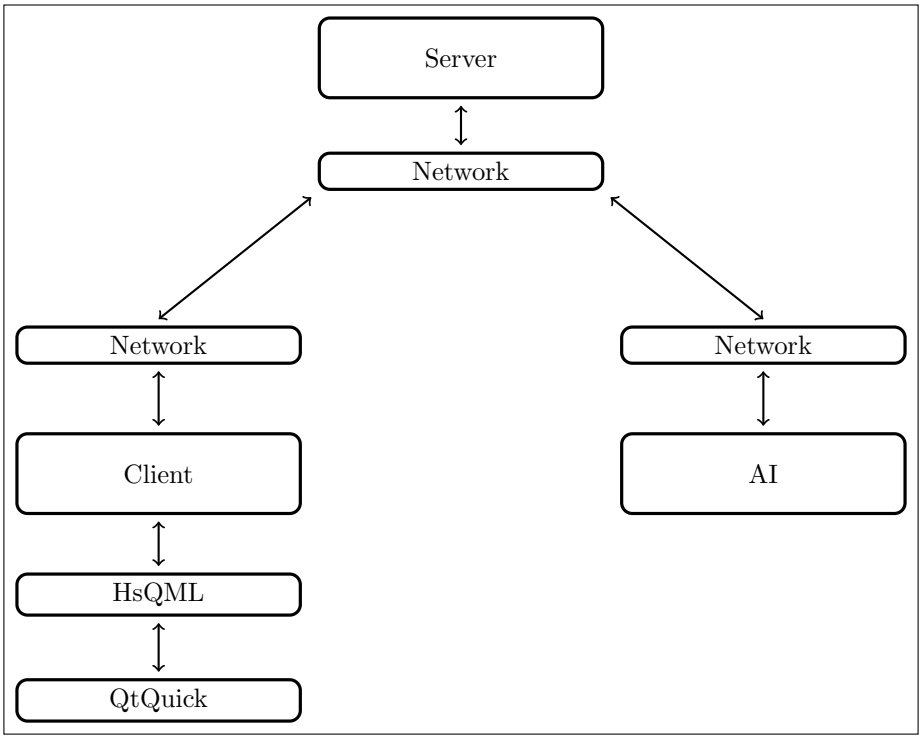
These can be examined to see their effectiveness, measured using the formula:

$$\frac{\text{Profit/Big Blind}}{\text{Hands/100}} \tag{1}$$

The AI will be designed to be modular, so the server operator can run AI players to supplement the normal players, and the users can locally run the

AI to improve their game play, along with a local server which hosts the actual game play. It will function the same as the client, except instead of taking input from a user, it will of course calculate its own input. It will interface with the server over the network normally.

Below is a diagram indicating the design and interface of the programs. The smaller boxes indicate libraries that will be utilised.



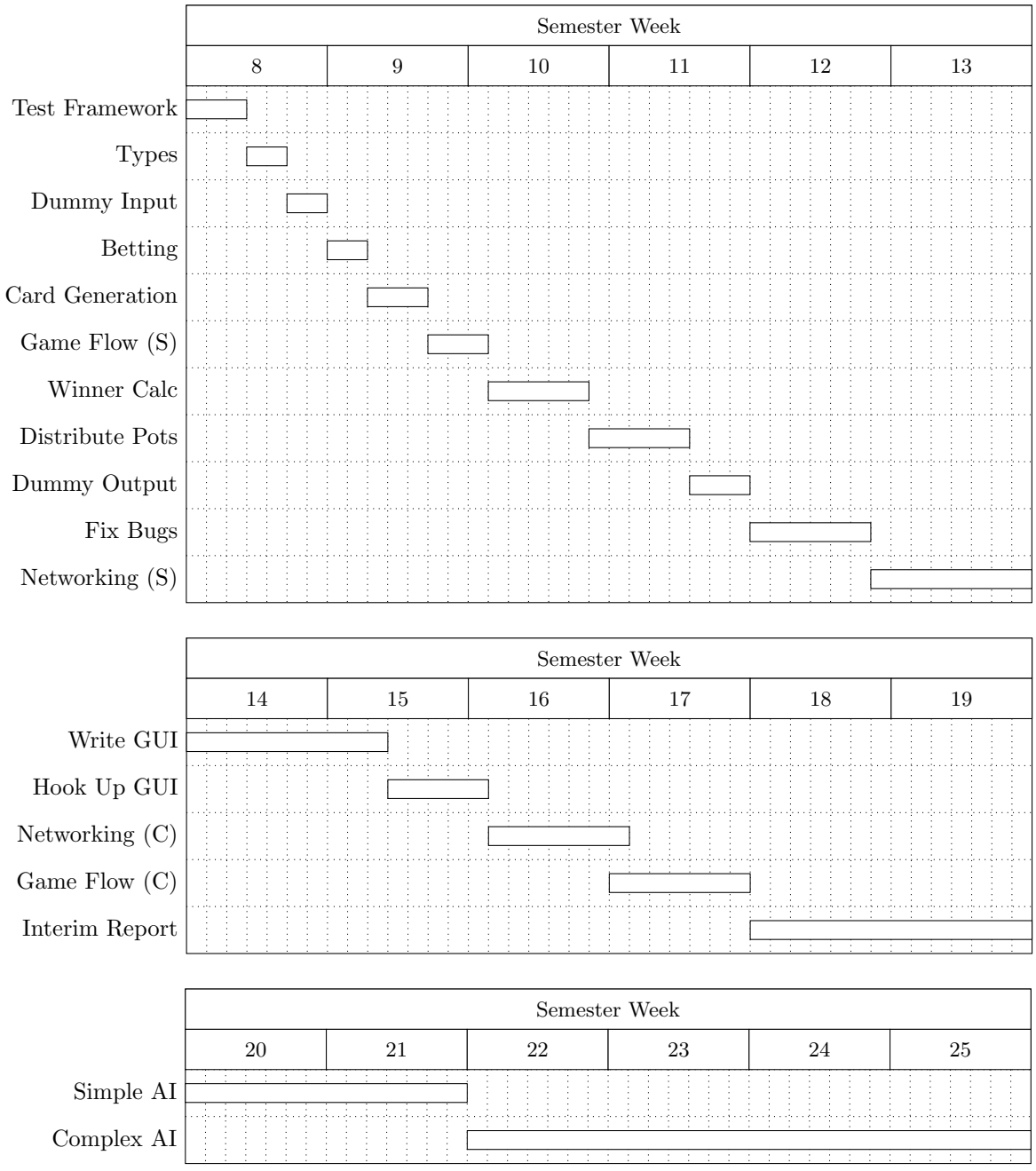
4 Task List

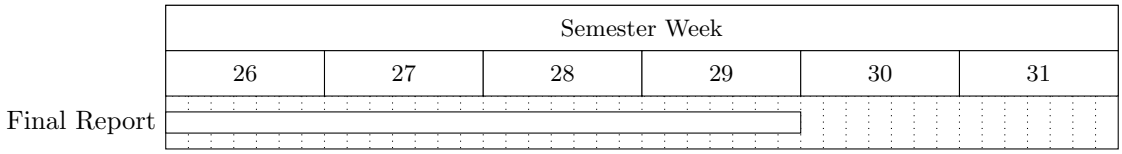
The below task list is split up into four sections, the server, the client, the AI, and report writing. (S) = Server, (C) = Client.

Task Name	Description	Days
Test Framework	Set up automated testing with QuickCheck	3
Types	Make server data types, instances, and monads	2
Dummy Input	Take console input for testing	2
Betting	Implement adding and removing of chips	2
Card Generation	Randomly take cards for players and table	3
Game Flow (S)	Give cards, ask for input, allow bets	3
Winner Calc	Calculate who has the best hands	5
Distribute Pots	Hand out chips, handle split and sub pots	5
Dummy Output	Add console output for debugging	3
Fix Bugs	State is pretty complex, bugs are inevitable	6
Networking (S)	Add network output and input on server	8
Write GUI	Describe the GUI in JSON with QML	10
Hook Up GUI	Connect input and output of GUI to Haskell code	5
Networking (C)	Send actions to server, receive state updates	5
Game Flow (C)	Implement stripped down game on client	7
Simple AI	Begin with a simple rule based representation	14
Complex AI	Use remaining time to explore advanced AI methods	28
Interim Report	Write the interim report deliverable	14
Final Report	Write the final report	28

5 Time Plan

(S) = Server, (C) = Client




Semester Week
26
27
28
29
30
31
Final Report

6 Risk Analysis

In the below table, severity means the impact times the likelihood.

Risk	Impact	Likelihood	Severity	Mitigation	Recovery
Hard Drive Failure	High	High	High	Backups on another machine	Restore from backups
Act Of God, Fire	High	Low	Medium	Off site backups on GitHub	Restore from GitHub
Running Out Of Time	Medium	Medium	Medium	Focus on core functionality	Skip ambitious parts
Buggy Code	Medium	High	Quite High	Automated testing	Fix bugs when flagged
Code Complexity	Medium	Medium	Medium	Loosely coupled code	Split into sub modules
Poor Performance	Medium	Low	Quite Low	Constantly test performance	Rewrite slow code
Unexpected Tasks	Medium	Medium	Medium	Careful planning	Alter time plan

References

Abou-Saleh, F, J Cheney, and J Gibbons (2015). "Notions of Bidirectional Computation and Entangled State Monads". In: *Lecture Notes in Computer Science*. Vol. 9129. Springer-Verlag, pp. 187–214.

Benton, N, J Hughes, and E Moggi (2000). "Monads and effects". In: *Lecture Notes in Computer Science*. Springer-Verlag, pp. 42–122.

Billings, D, A Davidson, et al. (2002). "The challenge of poker". In: *Artificial Intelligence* 134.1-2, pp. 201–240.

Billings, D, D Papp, et al. (1998). "Opponent modeling in poker". In: *Fifteenth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, pp. 493–499.

Billings, D, L Pena, et al. (1999). "Using probabilistic knowledge and simulation to play poker". In: *Sixteenth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, pp. 697–703.

Bohannon, A, BC Pierce, and JA Vaughan (2006). "Relation lenses: A language for updatable views". In: *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Association for Computing Machinery, pp. 338–347.

Davi, BMJ et al. (2010). "Matching Lenses: Alignment and View Update". In: *ACM SIGPLAN Notices* 45.9, pp. 193–204.

Davidson, A et al. (2000). "Improved opponent modeling in poker". In: *Proceedings of the International Conference on Artificial Intelligence*. Vol. 1-III. CSREA Press, pp. 1467–1473.

Hudak, P et al. (1992). "Report on the Programming Language Haskell: A Non-strict, Purely Functional Language Version 1.2". In: *SIGPLAN Not.* 27.5, pp. 1–164.

Korb, KB, AE Nicholson, and N Jitnah (1999). "Bayesian poker". In: *Uncertainty in Artificial Intelligence, Proceedings*. Morgan Kaufmann Publishers, pp. 343–350.

Rubin, J and I Watson (2011). "Computer poker: A review". In: *Articial Intelligence* 175.5-6, pp. 958–987.

Sandholm, T (2010). "The State of Solving Large Incomplete-Information Game, and Application to Poker". In: *AI Magazine* 31.4, pp. 13–32.

Teofilo, LF and LP Reis (2011). "Building a No Limit Texas Hold'em Poker Agent Based on Game Logs Using Supervised Learning". In: *Lecture Notes in Artificial Intelligence*. Vol. 6752. Springer-Verlag, pp. 73–82.

Watson, I and J Rubin (2008). "CASPER: A Case-Based Poker-Bot". In: *Lecture Notes in Artificial Intelligence*. Vol. 5360. Springer-Verlag, pp. 594–600.