Interim Report: Rekop Poker

James Scully (4304469)
psyjs20@nottingham.ac.uk
G400 Computer Science

Project Supervisor: Milena Radenkovic

Contents

1 Introduction

When one looks for a casual poker game that offers power to the actual end user, rather than attempting to fish money it can be quite difficult to find. From the authors own personal experience and observations with free and open software, it can be quite difficult to implement in some areas (such as games), however the ability to control a game how you would like and the ability to not have to rely on companies servers, trudge through in-app payments and ads is a unseen experience today, yet is revelled.

2 Motivation

Rekop poker aims to provide an alternative to other multi-player applications on the Android app store as many of these prevent user customization and introduce in-app purchases, with no ability for players to play locally on their own network.

3 Related Work

4 Description

5 Methodology

Test-Driven Development has been essential so far in the development of the poker algorithm. Evaluating Texas Hold'em hands has many vectors of errors and being able to write tests before and alongside implementation helped out immensely.
Take this snippet from the testing whether a hand is a straight or not:

```
assertTrue(new TexasEvaluator("0D 0D 2D KD AD QD JD").isStraight());
assertTrue(new TexasEvaluator("AD KD QD JD 0D QD JD").isStraight());
/* Falses */
assertFalse(new TexasEvaluator("0D 0D 0D 0D AD QD JD").isStraight());
assertFalse(new TexasEvaluator("2D 5D 9D JD 0D QD JD").isStraight());
```

Regarding the vectors of error, having instantenous feedback, whereby it was easy to debug providing breakpoints, the actual outcome and generally a sense of direction.

6 Design

Overview

The project itself is being broken down into three compartments - a backend that satisfies the core Texas Hold'em gamemode, the server which will provide the connections needed for players to play together on their own networks and the Android app, which links both of these together with the graphical interface.

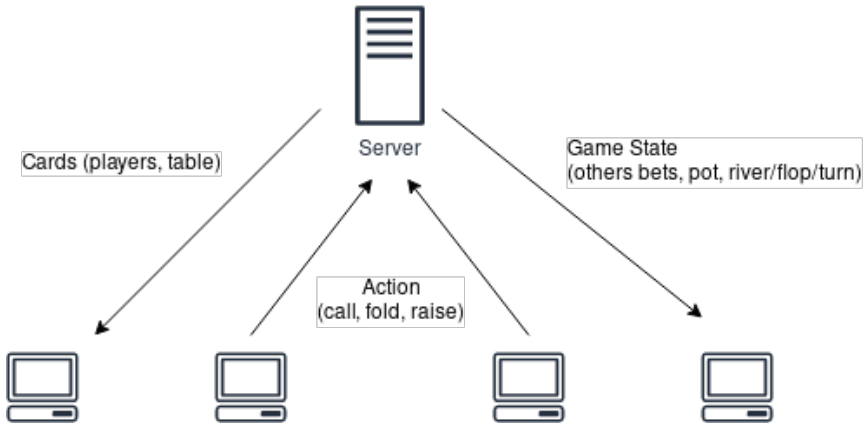
Backend

The backend is written solely in Java as the language itself is platform agnostic - Android being reliant upon Java (bytecode). It has been broken down in such a way that game modes other than Texas Hold'em can be added to it, enabling further development down the line.

For example, we've created data classes for Faces and Suits, each containing values for easy comparison. Furthermore, we've created a Deck class that allows for the user to pull a random card from a standard 52-card deck and ensure that it hasn't been pulled before. These building blocks allow for future development by providing the basics needed in a standard card game.

Server

Initially, an idea as defined in our Vision and Scope document was to use a UPnP / Peer to Peer approach, however this is infeasible as a key requirement of this project is disconnection-tolerance. This is because if the host is to lose connection permanently or temporarily, the rest would be unable to play. This is where we've opted for a client-server approach; as this allows atleast for one solid host, separate from the clients whom are likely to suffer connection issues.
The servers design is such that on the start of the game, the server knows exactly what cards are going to be placed on the actual table, i.e. the initial flop (3 cards), turn and river albeit these will not be released to the player until necessary. It is responsible to send what cards each player has, and will utilize the previously mentioned Deck object to deal hands out.
Therefore, the only messages required by the client to send are their actions, such as how much they are betting, whether they fold or raise.



Algorithms

Hand Strength - The algorithm for hand-strength is very basic and as such is ineffcient, however this is not the main focus of the project. The base of our algorithm is in a TexasEvaluator class, whereby we test each potential result i.e. straight, flush in their own methods. We then sequentially run these methods in order of ranking, so that we can return a result.
It is inefficient, however it is not the total focus of the project to ensure that it is efficient.
Win evaluation - Win evaluation can be done by sorting each players hand by result, i.e. Royal Flush, Four of Kind, Straight. If each player has a unique result, then we can simply take the top result, in this case Royal Flush. If two players contest the highest result, e.g. a Straight, then we'll look at the highest card in the TResult object, which determines each players result and the highest card.
If two players contest the highest result, but have the same highest card, then the pot is split as is the normal result in poker.

7 Implementation

8 Progress

Management

Management has been key in this project, of which I have not performed greatly. Time has been spent on modules which are lesser credits and although that coursework must be done, it shouldn't take as much time as it does. To combat this, I've essentially tried to keep doing atleast one task a day - whether it be debugging a function or atleast analysing what the error is, or writing a function.

Contributions and Reflections

A key goal of all software is that it must be efficient - both in terms of code complexity and performance. Upon undertaking this project it was naively assumed that, given how easy it is to recognize the outcome of a poker round in reality, it wouldn't be too difficult to implement efficiently through code.
It is therefore that the algorithm currently used is very compartmentalized, and potentially slower than some implementations (such as look-up tables or doing bit-wise operations). For example, it takes between 50 - 100ms for one result to be calculated on a desktop computer. Since most servers will be running on one, it doesn't particularly matter about the performance as much as the other aspects of the program.

Reflecting upon how I assumed Test-Driven Development would take much of our time before and during up and that we did not have "too much time to write tests and develop a fully-functional program", this couldn't have been further from the truth. Having instant feedback on whether it passed, what the result was and the ability to debug into it was essential.

Although, initially I was under the assumption that each test case was going to have to be made up of newly created objects. Frustrated that test cases were taking too long and a nuisance to write, I created a factory-esque pattern within the evaluator class itself to resolve, for example, "0D 0D 2D KD AD QD JD" into an actual hand + table cards.

The example below shows how writing test cases becomes much easier through this method, rather than creating a new object for each test case.

```
TexasHand FLUSH_FOK = new TexasHand(
    new Card(Suit.CLUBS, Face.FOUR),
    new Card(Suit.CLUBS, Face.FOUR),
    new Card(Suit.CLUBS, Face.FOUR),
    new Card(Suit.CLUBS, Face.FOUR),
    new Card(Suit.CLUBS, Face.ACE)
);
...
public void isFlush() {
    assertTrue(FLUSH_FOK.isFlush());
```

```
assertTrue(new TexasEvaluator("4C 4C 4C 4
    C AC QS JC").isFlush());
```

9 Bibliography