UNIVERSITY OF TROMSO

Advanced Database

Assignment 1 – Old Maid

By Simon Engstrøm Nistad

26.02.2013

In this assignment we will prove that interoperability is hard by creating client playing The all popular card game “All maid”. The implementation is based on a sparse documentation sheet handed out in the assignment text.

1.0 Introduction

1.1 Overview

The assignment is to implement the card game "Old Maid". The actual purpose of the assignment is to demonstrate how hard it is to achieve interoperability between systems based on a pre-specified protocol. The game API is based upon a sparse documentation sheet that explains the overall communication and rules of the game.

1.2 Requirements

The requirements for this assignment are:

- The client should implement the API described in the assignment text
- The code should be able to successfully play a round of the game
- The code must be tested against fellow students

1.3 Technical background

The reader of this report should have basic knowledge of programming including network programming and the usage of sockets. The implementation is written in python, so the reader should also be familiar python programming language.

1.3.1 Old Maid

Old Maid[1] is a Victorian card game for two to eight players. The game is known in Norway under the name "Svarte Petter". The game could be played with either a normal card deck or a dedicated deck holding pictures of matching pairs. The game rules are simple and goes as follows:

1. Each player in turn draws a card from the deck on the table
2. When all cards are drawn the player who draws the last card offers his hand to the player on his left side.
3. When a player receives a pair of cards, he can discard the pair to the table.
4. When all the cards have been discarded, the player left with the "Old Maid" card loses the game.

2.0 Design and implementation

The solution is written in the high level language Python[2], the language was used to make it easy and fast to implement a solution for the assignment.

The first version was implemented as a simple sequential program, but this implementation was scratched because of different issues with the communication part that would be non-trivial to implement without using threads. The second and current solution is an event based design where everything happens as events. This made it easier to handle different communication issues.

[1] http://en.wikipedia.org/wiki/Old_maid_(card_game)
[2] http://python.org/

The event system

The event system is the core of the design and is built using three core components:

- **Event queue**
- **Event Dispatcher**
- **Event**

The event queue is a simple queue implementation for holding incoming events. Whenever a thread wants to knottily the core of some event that needs to be handled, the thread can simply push a pre-registered event to the queue.

The dispatcher is used for registering new event handlers and handle incoming events that are added to the event queue. A registered event handle holds an event type and a method to be executed when an event of that type is received in the event queue.

An event in the queue holds two things; the event type and a dictionary holding all of the optional event data.

The card system

The next important part of the implementation is the Card handling system. The card systems are based on two classes:

- **Card**
- **Hand**

The card class represents a single card and holds all the necessary details and method for comparing and getting values. The Hand class hold the logic of all cards currently hold by the client. It has two lists; card on hand and cards that are pairs. Whenever a new card is added to the hand, it checks if this would be a pair, and if it is, the pair is placed in the pair list, and a got pair event is registered to the event queue. The event handler will then receive the event and discard the pair to the card server. Also note that whenever a card I placed into the deck, the hand is shuffled as required in the assignment text.

The player system

This is where all of the information about the other players is stored. The information hold for each player is: The player's address and port, the player number and name, and the last currents know state (out or in). The states are updated by the get status command to the card server.

Communication system

Figurer 1 shows the three connections needed to be maintain to be able to play the game. The card server handles all the actual deck and the state of the players. Note that the responsibility to notify the card server of state changes is fully in the hands of each player, and could therefore be wrong. By state it means if a player is still in or out of the game.

The communication "system" is written using the python socket library. For the left and card server communication, the player act as a standard client. For the right player part, the player acts as a simple

multithreaded server, spawning new threads with a handler for each connection. By doing this, the system is notified with an event for every incoming message received from the right hand player.

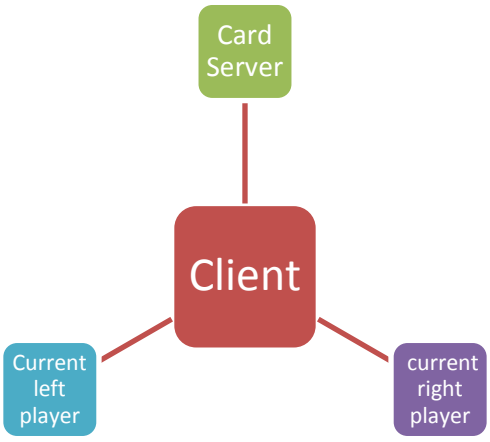

Figure 1 - Shows the 3 commination channels needed to play the game

Playing the game

The first thing the client will do is to join a new game on the card server, when receiving the list of the players playing this current round, an entry for each of the players are made in the player class. A right-hand server is started for listening for connections from the right player, and a connection to the left player is also setup. If there are any errors during this phase, the client will do a given number of retries before shutting down.

The client is design so the game is spilt into two phases; first the draw in cards from the deck part, and then the actual game. The player will discard cards whenever receiving a new card, this means that the possibility that the client could run out of card while drawing from the deck, this of course does not mean that the player has won the game, so this is the main difference in the first part.

Whenever the client either receives a “last card” command or “offer hand” message from the right player the second stage of the game is set. If the command from the card deck is “last card”, the client will offer hand to the next player, and if it’s an offer hand message from the right player pick a card as described in the API.

Whenever a the client receives an “out” message from the left player, the player information in the list of players is updated and a connection to the next left player not out of the game is setup, if all the other players are out of the game, and the Client have the last card “old maid” he has lost the game.

3.0 Discussion

The implementation has been tested against fellow students solution and I 90% of the runs the game completes successfully with no errors. In the remaining 10% of the runs the game is normally broken by the occasionally deadlock or a random socket exception from one of the players in the game.

The assignment seemed really easy in the beginning, but when I started on the project I fast realized that there were a lot of issues that had to be taken into account. The API handed out of course differs from the actual API run on server, so after writing the code tested on a test server created by fellow student, some parts of the code had to be rewritten to fit the actual commands and responses used by the game server.

Some of the main issues making the code work with others solution is that everyone seems to interpret the API in some different way, like the representation of card, do they use numbers for all cards or do the use string or a combination with numbers and strings.

Another issue was what to do when it was your turn and you have just received a card so you could discard the last cards on hand. This implementation will offer the next player the hand but saying that it has 0 cards. This again leads to another problem. Should the next player then send a "pick card" command, or should just offer its hand to the next player, or should it try to connect to the right players, right player and pick a card.

All in all the assignment proved to show what is was supposed to, that interoperability is hard, and there are a huge chances of miss interpretation of the API by other players, and of course this again breaks the whole system. The code was quite hard to write, not by the complexity, but of all the issues that had to be taken into account. There are still probably a lot of cases that could happened that are not taken into consideration.

4.0 Conclusion

To conclude the assignment was fun to work with, and it gave the opportunity to refresh a lot of topics like socket programing that I haven't used in a while. The finished "product" more or less works as intended, but again this is also highly up to the other player's implementation and interpretation of the API.