|  |
| --- |
| UNIVERSITY OF TROMSO |
| Advanced Database |
| Assignment 1 – Old Maid |
|  |
| **By Simon Engstrøm Nistad** |
| **18.10.2012** |

|  |
| --- |
| 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. |

# 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. 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
* Implement a system that can successfully play a round of the old maid game
* The code should be tested against fellow students

## 1.3 Technical background

### 1.3.1 Old Maid

Old Maid[[1]](#footnote-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 are 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. The player left with the “Old Maid” card loses the game.

# Design and implementation

The solution is written in the high level language Python[[2]](#footnote-2) to make it easy and fast to implement.

The first version was implemented has 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 easy to handle different asynchronous communication issues easy and the code could easily be extended without too much of an effort.

### The event system

The event system is the core of the whole design and the event class has 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 an pre-registered event type with optional extra data to the queue.

The dispatcher is used for adding new event handlers and handle incoming events from 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.

The event component is the actual event hand it 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 is 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.

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 an 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 - Shows the 3 commination channels needed to play the game

By combining the three parts over the systems works as a player fulfilling the requirements of the API handed out in the assignment text.

# Results

## 4.1 Measurements

The measurements done

## 4.2 Bugs

There are some bugs

## 4.3 Discussion

The assignment seemed really easy in the beginning, but as I started on the project I fast realized that there were a lot of issues that had to be taken in to account. But I guess that was the main purpose of the assignment. 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, big parts of the code had to be rewritten to fit the actual commands and responses used by the server.

All in all the assignment proved to show what is was supposed to that interoperability is hard, and there are a huge change of miss interpretation of the API by other players, and of course this again breaks the whole system.

# Conclusion

All in all the assignment was fun to work with, and it gave the opportunity to refresh a lot of field like socket program 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.

1. http://en.wikipedia.org/wiki/Old\_maid\_(card\_game) [↑](#footnote-ref-1)
2. http://python.org/ [↑](#footnote-ref-2)