UVG
Redes
17 de agosto, 2020

Integrantes: Oliver Mazariegos, José Pablo Viana, Estuardo Ureta

Documento resumen: Proyecto 1

Juego de cartas: Old Maid

Explicación e historia:

Old Maid es el nombre de un juego de cartas simple que requiere al menos dos jugadores pero puede llegar hasta 8 jugadores.
La traducción lógica para Old Maid es <Solterona>.
En esencia una solterona es una mujer que no está casada. Es una frase despectiva que se utiliza para describir a una mujer que se considera "vieja" y que no tiene marido.
La frase <solterona> ha existido al menos desde la época victoriana (aunque podría ser desde antes). En aquel entonces, la idea de que una mujer permaneciera soltera era impactante. El juego ha existido desde dicha época (Parlett, 2004).
En resumen, el juego requiere que los jugadores intenten evitar quedarse atrapados con la tarjeta Old Maid. Mientras bajan parejas por turnos para que acabe el juego solo con dicha tarjeta.
El juego reforzó las reglas sociales de la época. Implica que las mujeres deben casarse cuando sean jóvenes para no terminar siendo una solterona. También implicaba que los hombres debían casarse con una mujer joven para evitar quedarse atrapados con una solterona (Parlett, 2004).
Old Maid se juega con una baraja estándar de 52 cartas. Se consideraba que la reina negra era la vieja solterona. Una de las otras tres reinas se eliminó del mazo para asegurarse de que la carta de Old Maid no coincidiera (Kansil, 2002).

Descripción:

Detalles de protocolo:

En esencia se desarrolló un protocolo sobre TCP basado en un 3-way-handshake. Ya que se buscaba implementar la función de multi jugadores en nuestro juego de cartas, era necesario que se compartieran estados y que existiera uno general.
Con esto en mente primero definimos que el intercambio de información de mensajes (sean estos: texto, mano, username etc.) cuentan con una sección llamada HEADER. En esta es donde se encuentra la longitud del mensaje y en la segunda parte el mensaje. Esta segunda básicamente es un diccionario. Estos mensajes tienen una estructura básica que cuenta con: el tipo, mensaje y usuario. Por lo tanto esta modalidad se implementó con toda la información que se debe transmitir para el funcionamiento del multijugador.

Protocolo

```
dprotocol = {
    'type': 'cardPick',
    'oldmaid': oldmaid
```

```
}

dprotocol = {
    'type': 'room',
    'type2': type_msg,
    'roomID': roomid,
    'players': players
}

dprotocol = {
    'type': 'useraccepted',
    'username': username,
    'roomID': 1
}

room = {
    "players": [client],
    "time_connection": [time]
}

dprotocol = {
    'type': 'message',
    'username': username,
    'message': message
}

dprotocol = {
        'type': 'you_can_play_now',
        'players': OldMaid.getPlayers(),
        'oldmaid': OldMaid
    }

dprotocol = {
    'type': 'all_pairs_down',
    'oldmaid': oldmaid
}

dprotocol = {
    'type': 'pairsOk',
    'oldmaid': oldmaid
}

dprotocol = {
    'type': 'signin',
    'username': my_username
}
```

```
dprotocol = {
    "type":"signinok",
    "username":username,
    "roomID":roomID,
    "winner": 0,
    "turn": 0,
    "hand": []
}

dprotocol = {
    "type": "roomid",
    "username": my_username,
    "roomid": roomID
}

dprotocol = {
    "type":msgtype,
    "message": message,
    "username": username
}

dprotocol = {
    "type":'updateMessage',
    "hand": hand,
    "username": username,
    'hasQueen': hasQueen,
    'numTurn': numTurn
}

dprotocol = {
    "type":"sendpair",
    'pair': pair,
    'hand': hand
}

dprotocol = {
    "type":'im_done',
    "room_id": roomid,
    "username": username,
    "hand": hand
}

dprotocol = {
    "type":"pickCard",
    'cardpos': cardpos,
```

```
    'room_id': roomid
}

dprotocol = {
    'type':"error",
    'error': error
}
```

Link a documento con diccionario y guia de protocolo: https://docs.google.com/document/d/1DK29HZ7POETEu4KUkKcmrgEaqQT9JHIwpxHBIffnqc0/edit?usp=drivesdk

Instalaciones:

No se requiere la instalación de ninguna librería ya que las utilizadas vienen en el estándar.

Librerías utilizadas:

- import socket
- import select
- import errno
- import sys
- import pickle
- import threading
- import time
- import itertools
- import random
- import datetime

Versiones y para la ejecución:

Local:

Para la ejecución local del programa es importante primero descargarlo en github y clonar.
En la ubicación de la carpeta correr en la terminal el archivo server.py para inicializarlo.

Online:

Ip: 18.222.142.27
Se ha implementado el uso de Amazon Web Services (aws).
Esto para montar el servidor en la nube y se incluyen todos los datos de conexión.

Para correr el servidor en local

```
python server-test.py
```

Para correr el cliente en local

```
python client-test.py
```

Para correr el cliente con el servidor en amazon ec2

python client-cloud.py

Corriendo cliente.py para cada uno de los usuarios, seguir instrucciones para cada uno.

Retos y dificultades:

A continuación se listan algunos retos y proyecto que se presentaron durante el progreso del proyecto acompañadas con una breve descripción:

- La primera complicación fue la comunicación con el servidor:
    - Ya que el servidor debe aceptar nuevas conexiones de los clientes. Se buscó alguna forma de identificar a nuestros usuarios como únicos. Aunque bien se podría mostrar a los usuarios por dirección IP, la mayoría de las personas tienden a crear algún tipo de nombre de usuario. Por lo que nuestro servidor primero permite a los clientes conectarse y elegir un nombre de usuario. Aprendimos más allá de esto que, el servidor recopila los mensajes entrantes y luego los distribuye al resto de los clientes conectados.

- La segunda y tercero fue la creación de los Rooms y cómo asignar personas a un room:
    - En client.py, tras ingresar usuario, se despliega un menú y cuando el jugador selecciona la opción de jugar se crean los Rooms para que el server ya lo pueda operar. No estábamos haciendo que el server reconociera si es un Room nuevo o si ya estaba en otro Room. Aprendimos que cada vez que un usuario se conecta al servidor, se crea un hilo separado para ese usuario y la comunicación del servidor al cliente se lleva a cabo a lo largo de hilos individuales basados en objetos de socket creados por el bien de la identidad de cada cliente.

- Cuarto como hacer que todos compartan un solo Deck u OldMaid por así decirlo:
    - Se comenzó instanciando el OldMaid individualmente en cada cliente y notamos que así no funcionaba. Llegamos a la conclusión, realizando pruebas, que el server debía de encargarse de esto y por lo tanto se corrió al mismo solucionando el problema.

- La mayoría de problemas residieron en la manipulación de los turnos en el juego en la cual nos topamos un while infinito que permite a veces terminar el juego y a veces no dependiendo de las cartas y ya que estas son generadas de manera aleatoria esto fue una complicación.

- Se empezó utilizando una librería de alto nivel y al preguntar se nos notificó que esto no era permitido por lo que se hizo con sockets en Python. Ni siquiera es necesario

instalar nada más que Python. Esto presentó un reto ya que se invertió energía en una implementación y empezar una de 0 y en bajo nivel fue complicado. No obstante por la misma razón pensamos que el objetivo del proyecto fue alcanzado en cuanto a conocimientos.

Referencias:

1. Kansil Q. (ed.), Bicycle Official Rules of Card Games (2002)
2. Parlett D., The A–Z of Card Games, 2nd ed. (2004; 1st ed. published as Oxford Dictionary of Card Games, 1992)
3. Rigal B, Card Games for Dummies, 2nd ed. (2005).