Online Onitama

Advanced Programming
Gilberto Echeveria Furio
Ruben Cuadra A01019102
26 November 2018

COMPONENTS

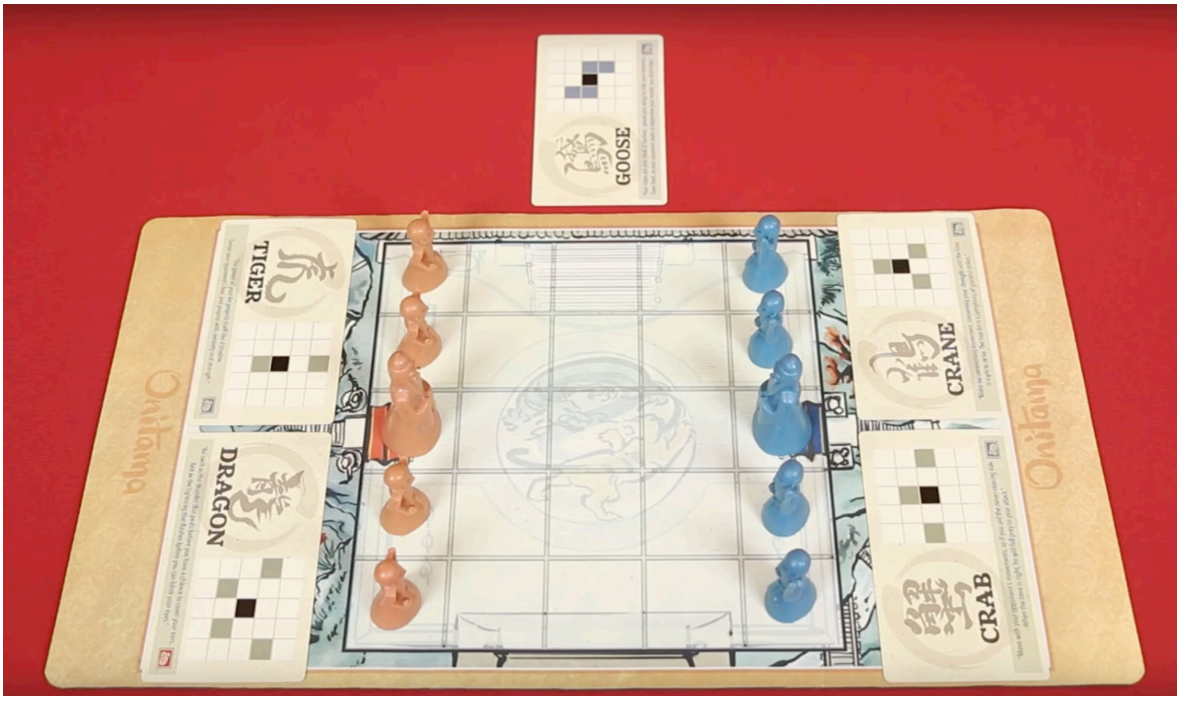
The project consists of an online version of the Japanese strategy board game called “Onitama”, it consists of 2 parts:

- Game Master (server) : in charge of listening for incoming players, assigning players to a table, starting matches, delivering cards, assigning turns, keeping scores and choosing the winners.
- Player (client): in charge of connecting to a server, showing the GUI with the status of the match and choosing cards to play

The server accepts multiple connections and can play against players (PvE) or match players and be the moderator (PvP)

RULES

It is a 2 players strategy game, the board is a grid of 5x5, each player has 5 “tokens”, 1 is the “master” and the remaining 4 are “students”, at the beginning of the match the Game master takes out 5 cards and gives 2 of them to each player and leaves the remaining one in “stand by”, the cards contain valid movements for the tokens, an initial configuration for the game is shown in the picture:



The tokens are moved according to the player cards, we pick one token and one movement, from there we have multiple valid cells to do the movement, if the cell is free we just move there, if the cell is occupied by an enemy token we move there and take the token out of the board, we can't move to that cell if it is occupied by other of our tokens or if it is out of the borders, each time a movement is done we take the used card movement and switch it with the stand by card, meaning that in each game there are only 5 valid movements and they end up rotating between players. After each movement the player's turn ends and beginning the opponent's turn

The goal of the game is to win by eliminating the opponent's master or by moving our master to the initial opponent's master position

Turn steps:

1) Choose movement to do
2) Choose token
3) Choose a valid cell to move according to the movement & token
4) Move token to cell (remove opponent's token from the board if it is the case)
5) Switch done movement with the "stand by" movement

After step 5 we check if there is a winner, otherwise it starts the other player's turn

Video explanation: https://www.youtube.com/watch?v=IFRewjcngwU

ENVIRONMENT

- Server: Tested with clang-1000.11.45.5 and gcc 8.2.0, it uses the library of pthread and other standard libraries.
- Client: Requires Python 3.6.x (String interpolation not allowed by Python <3.6) and Pygame 1.9.4
-

This environment works in Mac OS 10.13 High Sierra and Below, there is a bug with MacOS 10.14.1 Mojave in which the GUI shows an empty screen, this is a problem with pygame and it should be fixed inside the library

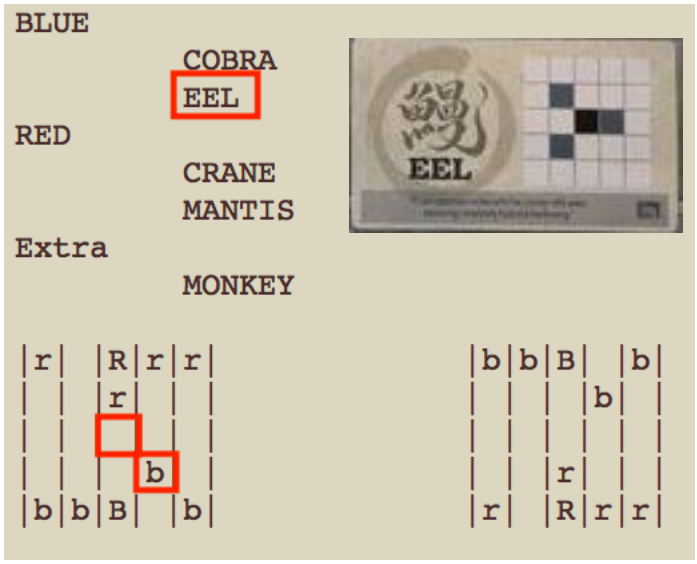
STEP BY STEP

- **Server**: It is written in *C* and the main file is *server.c*, the code starts initializing the required data structures and generating tables for users, it then starts listening in the desired port, when it receives an incoming connection the code launches a new thread for attending the request and it keeps waiting for more connections. The thread will communicate with the client, the first thing a client must send is the type of match that they want to play, been *Player Versus Environment* or *Player Versus Player* the options, once decided the client should send the desired *difficulty* (PVE) or *table* (PVP).
    - PVP: Once that the server got the table it will check the status of it, the 3 possible statuses are *EMPTY*, *FREE* and *PLAYING,* which represent the current number of players in the table, if a client wants to join to an *EMPTY* table it will be assigned and left to wait for a new client (thread ends here). If the table is *FREE* the server will check that both players are still connected and then use a mutex lock for blocking the table, this thread will start the game and be in charge of moderating the match, for match interaction check the section below. Once that the match is over the thread will update the table status to *EMPTY* and it will also unlock the mutex, thread ends here.
    - PVE: In this case a temporal board is created and the thread starts playing against the client, for match interaction check the section below, each time the server has to play it will launch a new process (popen), the server calls a python script that returns us the best movement given certain board and difficulty, for more about this script check the section below. Once that the match is over the server destroys the temporal board and finishes the thread of the match.
- **Client**: It is written in *Python 3.6* using the *pygame* library, the main file is *client.py*, at the beginning the code spawns 2 threads, 1 for the GUI and other for the client-server communication, at the beginning the user writes in the terminal the options (PVP/PVE - Difficulty/Table), if the options are correct the server will start the match and sends us a board, from here we launch the graphical interface and the whole interaction will take place here. Once that the match is over, the client closes the connection with the server, finishing communication thread; the GUI thread finishes until the user closes the window. Once both threads are done the script finishes

- **Match Interaction**: The server is the game master, at the beginning of a match it will send to the client(s) their respective colors and the 5 starting cards, the player with the blue color is the one that it will always start, in PVP the server will wait for the current playing player movement, when it sends a correct movement then the server will do an update to its local board and then notify to the other player and will change the current playing player to the other one, after each movement it will check if it is a game over and if it is it will notify the clients and will finish the match, in the server the game logic is written in the *Onitama.c* file while in the client side it is in *board.py* inside of the package called *Onitampy*

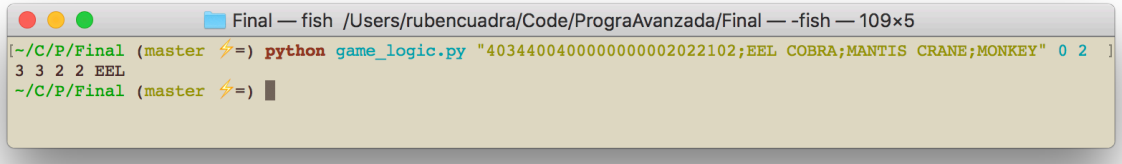
- **Board**: The problem was solved using Object oriented programming, the Onitama Board is a matrix of 5x5 filled with numbers. 0 Empty; 1 and 2 for Blue master and students ; 3 and 4 for Red Master and Students. The board has values(the matrix) and an array of cards, the cards are objects with a name and an array of Movements, The Movements are tuples with movement vectors, for example, the tuple for UP is (-1,0), meaning that we go 1 row up and we move 0 columns.

- **PVE AI**: The PVE is done by a script in python that receives a state of the game (Board/Cards & Player) the script uses the *Minimax* algorithm, the difficulty given by the client represents the number of levels in the search three, this code is in *game_logic.py,* we call it like this, “ENCODED_BOARD;C1 C2; C3 C4;C5” {PLAYER} {DIFFICULTY}



C1 - C5 represent the cards, the first 2 are for the blue player, then red player and then the sand by card. Player is 0|1 for Blue|Red

At the end the script sends us 4 numbers and a Name, the numbers are FromRow, FromCol, ToRow, ToCol and the name is the card that should be used for that movement.



The heuristic is based in the number of remaining tokens and in the Manhattan distance from the master to the dojo

EXAMPLES
SERVER

- Server: ./server {PORT_NUMBER}

```
make
./server 8989
```

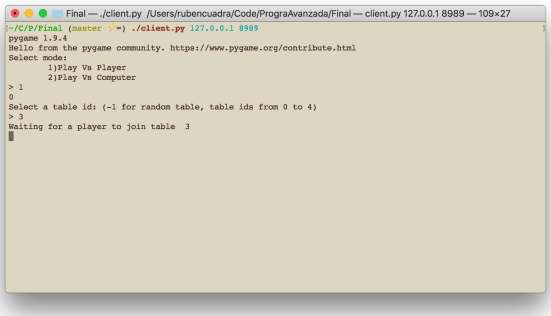
```
~/C/P/Final (master ⚡🖐)
0                                ls
Proposal.pdf     client.py       fatal_error.c   makefile        playground.py    server.c
README.md        codes.h         fatal_error.h   onitama.c       requirements.txt sockets.c
__pycache__      codes.py        game_logic.py   onitama.h       screenshots      sockets.h
assets           env             gui.py          onitampy        server
~/C/P/Final (master ⚡🖐)
1                               make
gcc server.c -c -o server.o -Wall -g -std=gnu99 -pedantic
gcc fatal_error.c -c -o fatal_error.o -Wall -g -std=gnu99 -pedantic
gcc sockets.c -c -o sockets.o -Wall -g -std=gnu99 -pedantic
gcc onitama.c -c -o onitama.o -Wall -g -std=gnu99 -pedantic
gcc server.o fatal_error.o sockets.o onitama.o -o server  -lpthread
~/C/P/Final (master ⚡🖐)
5                               ./server 8989

=== ONITAMA SERVER ===
Server IP addresses:
lo0: 127.0.0.1
en0: 192.168.1.75
en7: 169.254.64.251
STARTING
Server ready
```

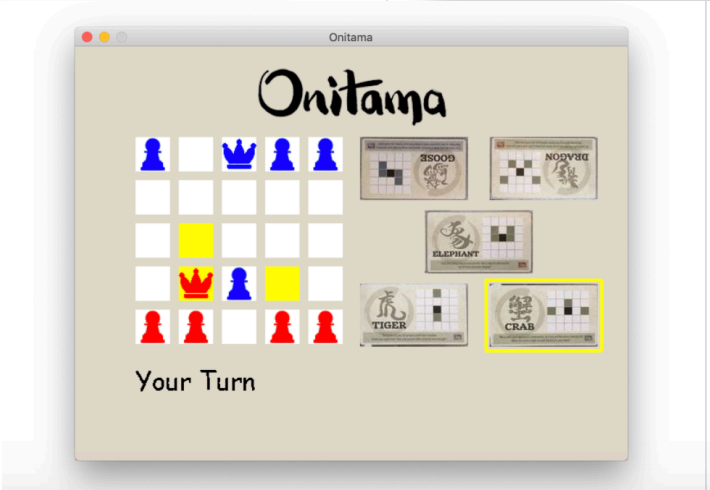
CLIENTS

- Client: python client.py {SERVER_ADDRESS} {SERVER_PORT}

```
python3 client.py 127.0.0.1 8989
```

```
~/C/P/Final (master ⚡🖐)
1                          python client.py 127.0.0.1 8989
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
Select mode:
        1)Play Vs Player
        2)Play Vs Computer
> 2
0
Select difficulty:
        1)Normal
        2)Hard
> 2
```





TOPICS USED:

- Dynamic Memory: Used for tables/boards and for passing parameters to new threads
- Pointers: Pretty much everything in the server is handled with pointers and params receive the addresses
- Process creation: The PVE part starts a python script that calculates the best movement (popen)
- Inter process communication: We wait until the spawned proc finishes and then we get the movement from it (fgets)
- Signals: We catch the SIGINT signal on the server side so that we are able to finish everything and clean memory
- Threads: Server spawns threads to attend clients, this threads also play against them
-

REFERENCES

http://www.arcanewonders.com/wp-content/uploads/2017/06/Onitama-Rulebook.pdf

https://de.wikipedia.org/wiki/Onitama

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/