GoMoKu项目设计文档

> **成员：陈昕(21S)；沈衍羽(21S151079)**
>
> **项目语言：Python3**
>
> **项目URL：本地启动后，进入网页http://localhost:6790/index.html**

System Demanding

In addition to the basic rules described in the problem description, the GoMoKu game needs to implement the following functions:

- Support multiple pairs of different players to play games at the same time
- Support real-time detection of game results and prompt the final result for two players
- After one side quits during the game, the other player will be informed "Connection Lost"

System Structure

- images：picture folder
- log：Server log folder。
  - `server.log` is the log generated by `server.py` (HTTP web resource related logs)
  - `gameServer.log` is the log generated by `gameServer.log` (game log);
- index.html：web page html file
- server.py：Web server file. Responsible for monitoring HTTP and sending web resources.

  Not responsible for playing the game. listening port number 6790
- **gameServer.py**：The backend service program of the Gobang game. listening port number 6791

```python
# control the GoMoKu game play
async def GamePlay(black_socket, black_name, white_socket, white_name):

# every websocket from browser will create a new coroutine
async def main(websocket):
    #...

if __name__ == "__main__":
    start_server = websockets.serve(main, "0.0.0.0", 6791)
    loop = asyncio.get_event_loop()
    print("game listening 0.0.0.0:6791")
    try:
        loop.run_until_complete(start_server)
        loop.run_forever()
    #...
```

- **client.js**：Client interaction script file, which interacts with `gameServer.py.`

```javascript
window.onload = function game()

// update the board in the browser
function drawBoard(board_sequence)

// inform the gameServer that this player is ready
function initPlayer(event)
```

```
8
9  // response to the click
10 function clickBoard(event)
11
12 // when websocket receive a new message, extract the opcode and data
13 //     Info: a new game start and init opponent's information
14 //     Step: confirm a move and update the board
15 //     Done: someone win the game
16 //     Lost: handle connection lost
17 function onMessage(event)
18     /...
```

- gamePlay.py： Realize *ChessBoard* class, including chess move and game victory condition detection. This class is instantiated by `gameServer.py`

```
1  class ChessBoard(object):
2      # use array to record the moves
3      def __init__(self):
4
5      # player move
6      def move(self, row, col, color):
7          #...
8
9      # check if new move will win the game
10     def win(self, row, col, color):
11         #...
```

Tolerate invalid moves & support multiple pairs of players concurrently

The concurrency in this project is implemented by using the built-in library *asyncio*, which uses event loop and coroutines to coordinate asynchronous events. There are two types of coroutines in the event loop: *main* and *GamePlay*

main(websocket):

Every websocket from browser will create a *main* coroutine. The *main* coroutine follows a producer-customer pattern and maintain a global player queue. In the main coroutine, the new arrived player will try to find opponent in the player queue. If there is no other player waiting in the player queue, then this new player will be put into the queue. This *main* coroutine will wait for the close of websocket. If there is a player waiting in the queue, then a new *GamePlay* coroutine will be create to manage the game play between these two players.

GamePlay(black_socket, black_name, white_socket, white_name):

Every game is represented by a *GamePlay* coroutine and coordinated by the event loop, so *gameServer* can support multiple pairs of players concurrently. The *GamePlay* corotines use ChessBoard object to verify, record every move and detect victory. If the move is invalid, it will be discarded. If the move is verified, then the update will be broadcasted to both players and the browsers will change the boards.