Server.Server Class Reference

Server class: More...

Public Member Functions

def **__init__** (self, broadcast_port=None, service_port=None)

def **broadcast_error** (self, data, sock, address=None)

def **create_game** (self, data, sock)
called by request handler <function=_handle_broadcast> when incoming message has <var=command> = CREATE Returns: Success / Failure message More...

def **poll_games** (self, data, sock, address)

def **join_game** (self, data, sock)
called by request handler <function=_handle_broadcast> when incoming message has <var=command> = JOIN Returns: Success / Failure message More...

def **start** (self, data, sock)

def **roll** (self, data, sock)
called by request handler <function=_handle_request> when incoming message has <var=command> = ROLL generates two random numbers in range(1,7) Returns: ROLL message More...

def **buyRequest** (self, data, sock)

def **pass_go** (self, sock)

def **remove_player** (self, sock)

def **quit** (self, data, sock)

def **chat** (self, data, sock)
called by request handler <function=_handle_request> when incoming message has <var=command> = CHAT Sends the message <var=text> that is in <var=values> onto other users, attaching values like username onto the message Returns: does not return just passes More...

def **turn** (self, data, sock)
send message TURN to all clients informing them of whose turn it is More...

def **go_to** (self, data, sock)
inform all players where one player is {values: {palyer: int player_id, tile: int tile } } More...

def **pay** (self, p_from, p_to, amount, sock)
transaction between player and player or bank and player if to or from are None {"command": "PAY", "values": { "from": int player_id or None, "to": int player_id or None, "amount": int amount } } More...

def **sell** (self, data, sock)
called by request handler <function=_handle_request> when incoming message has <var=command> = SELL 'sells' the properties defined in <var=ids> inside <var=values> Returns: PAY message More...

def **sendJail** (self, sock)
[player] goes to jail More...

def **time** (self)
Timeout method. More...

def **gameOver** (self, players)
sends 'game over' message to client(s) More...

Public Attributes

- **incomming**
- **connection_queue**
- **timer**
- **game**
- **discover**
- **service**
- **incomming_thread**
- **service_sock**

Static Public Attributes

int	**BROADCAST_PORT** = 44470
int	**SERVICE_PORT** = 44469
string	**BOARD_FILE** = "text/full_board.txt"
int	**CLIENT_DECISION_TIME** = 60
int	**GO_CASH** = 50
int	**GETOUT** = 200

Private Member Functions

- def **_run_incomming** (self)
- def **_incomming_messages** (self)
- def **_enqueueMessage** (self, data, con)
- def **_service** (self, port)
- def **_open_broadcast** (self, broadcast_port)
- def **_send_answer** (self, data, sock, address)
- def **_send_answer_tcp** (self, data, sock)
- def **_push_notification** (self, data, exclude=None)
- def **_move_player** (self, playerID, spaces)
  moving the player around the board More...
- def **_handle_card** (self, card, sock)
  card handling More...
- def **_proccess_position** (self, tile, sock)
- def **_waitResponse** (self, command, sock)
- def **_buy** (self, space, player)
- def **_onPropertySpace** (self, space, sock)
  logic for response to landing on a particuar space depending on its attributes (owned etc) More...
- def **_playGame** (self)
  autoplay method More...

Private Attributes

- **_game_over**
- **_timeout**

Detailed Description

Server class:

Constructor & Destructor Documentation

◆ __init__()

```
def Server.Server.__init__ (  self,
                              broadcast_port = None,
                              service_port = None
                            )
```

Member Function Documentation

◆ _buy()

```
def Server.Server._buy (  self,
                          space,
                          player
                        )
```

private

◆ _enqueueMessage()

```
def Server.Server._enqueueMessage (  self,
                                     data,
                                     con
                                   )
```

private

◆ _handle_card()

```
def Server.Server._handle_card (  self,
                                  card,
                                  sock
                                )
```

private

card handling

_incomming_messages()

def Server.Server._incomming_messages (self)

_move_player()

def Server.Server._move_player (self,
playerID,
spaces
) private

moving the player around the board

Returns

new_space the new position

_onPropertySpace()

def Server.Server._onPropertySpace (self,
space,
sock
) private

logic for response to landing on a particuar space depending on its attributes (owned etc)

_open_broadcast()

def Server.Server._open_broadcast (self,
broadcast_port
) private

_playGame()

def Server.Server._playGame (self) private

autoplay method

_proccess_position()

```
def Server.Server._proccess_position (  self,
                                        tile,
                                        sock
                                      )
```

private

◆ _push_notification()

```
def Server.Server._push_notification (  self,
                                        data,
                                        exclude = None
                                     )
```

private

◆ _run_incomming()

```
def Server.Server._run_incomming (  self )
```

private

◆ _send_answer()

```
def Server.Server._send_answer (  self,
                                  data,
                                  sock,
                                  address
                               )
```

private

◆ _send_answer_tcp()

```
def Server.Server._send_answer_tcp (  self,
                                      data,
                                      sock
                                   )
```

private

◆ _service()

```
def Server.Server._service (  self,
                              port
                           )
```

private

◆ _waitResponse()

```
def Server.Server._waitResponse ( self,
                                  command,
                                  sock
                                )
```

◆ broadcast_error()

```
def Server.Server.broadcast_error ( self,
                                    data,
                                    sock,
                                    address = None
                                  )
```

◆ buyRequest()

```
def Server.Server.buyRequest ( self,
                               data,
                               sock
                             )
```

◆ chat()

```
def Server.Server.chat ( self,
                         data,
                         sock
                       )
```

called by request handler <function=_handle_request> when incoming message has <var=command> = CHAT Sends the message <var=text> that is in <var=values> onto other users, attaching values like username onto the message Returns: does not return just passes

◆ create_game()

```
def Server.Server.create_game (  self,
                                 data,
                                 sock
                               )
```

called by request handler <function=_handle_broadcast> when incoming message has <var=command> = CREATE Returns: Success / Failure message

◆ gameOver()

```
def Server.Server.gameOver (  self,
                              players
                            )
```

sends 'game over' message to client(s)

Parameters

self the object pointer

String players the player that has won

◆ go_to()

```
def Server.Server.go_to (  self,
                           data,
                           sock
                         )
```

inform all players where one player is {values: {palyer: int player_id, tile: int tile } }

◆ join_game()

```
def Server.Server.join_game (  self,
                               data,
                               sock
                             )
```

called by request handler <function=_handle_broadcast> when incoming message has <var=command> = JOIN Returns: Success / Failure message

pass_go()

```
def Server.Server.pass_go ( self,
                            sock
                          )
```

pay()

```
def Server.Server.pay ( self,
                        p_from,
                        p_to,
                        amount,
                        sock
                      )
```

transaction between player and player or bank and player if to or from are None {"command": "PAY", "values": { "from": int player_id or None, "to": int player_id or None, "amount": int amount } }

poll_games()

```
def Server.Server.poll_games ( self,
                               data,
                               sock,
                               address
                             )
```

quit()

```
def Server.Server.quit ( self,
                         data,
                         sock
                       )
```

remove_player()

```
def Server.Server.remove_player ( self,
                                  sock
                                )
```

◆ roll()

```
def Server.Server.roll ( self,
                         data,
                         sock
                       )
```

called by request handler <function=_handle_request> when incoming message has <var=command> generates two random numbers in range(1,7) Returns: ROLL message

◆ sell()

```
def Server.Server.sell ( self,
                         data,
                         sock
                       )
```

called by request handler <function=_handle_request> when incoming message has <var=command> = SELL 'sells' the properties defined in <var=ids> inside <var=values> Returns: PAY message

◆ sendJail()

```
def Server.Server.sendJail ( self,
                             sock
                           )
```

[player] goes to jail

◆ start()

```
def Server.Server.start ( self,
                          data,
                          sock
                        )
```

◆ time()

def Server.Server.time (self)

Timeout method.

Parameters

self the object pointer

turn()

def Server.Server.turn (self,
data,
sock
)

send message TURN to all clients informing them of whose turn it is

Member Data Documentation

_game_over

Server.Server._game_over private

_timeout

Server.Server._timeout private

BOARD_FILE

string Server.Server.BOARD_FILE = "text/full_board.txt" static

BROADCAST_PORT

int Server.Server.BROADCAST_PORT = 44470 static

CLIENT_DECISION_TIME

int Server.Server.CLIENT_DECISION_TIME = 60 static

connection_queue

Server.Server.connection_queue

discover

Server.Server.discover

game

Server.Server.game

GETOUT

int Server.Server.GETOUT = 200 static

GO_CASH

int Server.Server.GO_CASH = 50 static

incomming

Server.Server.incomming

incomming_thread

Server.Server.incomming_thread

service

Server.Server.service

SERVICE_PORT

int Server.Server.SERVICE_PORT = 44469 static

service_sock

Server.Server.service_sock

timer

Server.Server.timer

The documentation for this class was generated from the following file:

- Monopoly-master/Monopoly-master/**Server.py**