

《人工智能》课程五子棋人工智能程序规范

一、任务描述

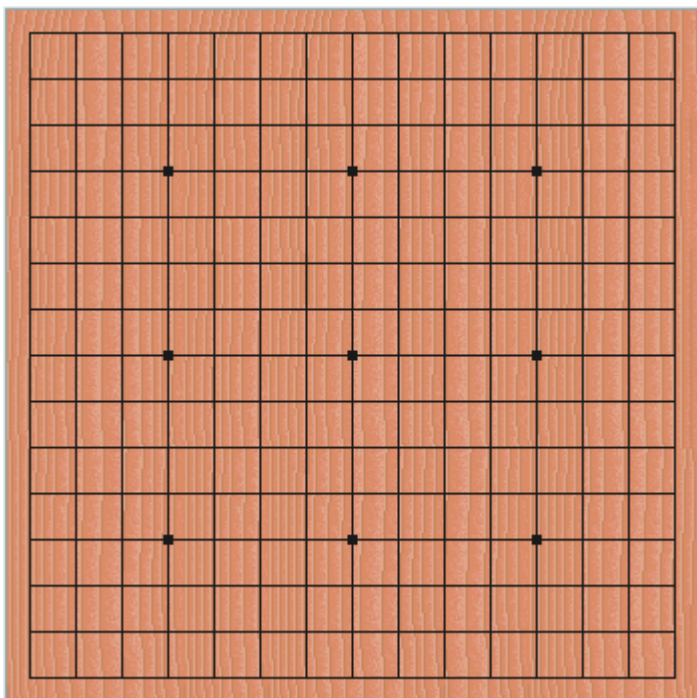
实现不同的五子棋人工智能程序的自动对垒及人机对垒过程,自动对垒过程中无需人工干预。自动对垒时,五子棋程序必须连接到作为服务器的裁判程序中,所有的对垒步骤都会记录在裁判程序的一个文件上,作为证据保留。五子棋人工智能程序的开发语言不作限制。

此份文档所定义的规范仅对自动对垒的情况有效


二、相应规范


1 界面

统一棋盘规定如下:



棋子参考如下:

白子: 

黑子: 

规定棋子均落在交叉点上,行,列总数均各为 15。行,列值均从 0 开始,到 14 结束。规定左上角为 (0, 0), 右下角为 (14, 14)。

2 进程间通信

为实现自动对垒过程,五子棋程序和裁判程序进程间应该能够通信,这里规定统一采用 socket 进行通信。每一个五子棋进程,都必须实现连接到裁判程序的功能。设定连接端口为 9527 端口。裁判程序只接受两个客户端的连接请求。

考虑到项目检验过程中,不同程序会在同一机子上执行,要求设置五子棋程序默认连接 IP 为 127.0.0.1 (也应可修改)。

3 通讯接口定义

规定自动对垒中必须实现的通讯命令, C 代码形式的 socket 通信命令定义如下:

所有通信命令均以 byte (8 bit 字节传送)

```
//define commands for the socket communication
```

```

#define COMM_MSG_DONOTHING      0    //do nothing, ignore it!
#define COMM_MSG_REJECTED      1    //last request is rejected
#define COMM_MSG_ACCEPT        2    //last request is accepted
#define COMM_MSG_FIRST         3    //go first when the game starts
#define COMM_MSG_SECOND        4    //go second when the game starts
#define COMM_MSG_GAME_REQUIRE_START 5 //request for game start
#define COMM_MSG_GAME_START     6    //game start
#define COMM_MSG_CHESS          7    //information for chess
#define COMM_MSG_TIMEOUT        8    //time out when one peer takes too many time in a bout
#define COMM_MSG_WIN            9    //win
#define COMM_MSG_LOSE          10    //lose
#define COMM_MSG_DRAW           11    //draw

```

棋子颜色值定义如下：

```

//define color for the chess
enum chess_color{
    Chess_Clr_Black = 0,
    Chess_Clr_White = 1,
}

```

各通讯命令详解如下：

COMM_MSG_DONOTHING

客户端或服务端可以收到此命令。收到此命令时，忽略此命令。

COMM_MSG_REJECTED

客户端可以收到此命令。当客户端提交给服务器的命令是一个非法值或是提交违反五子棋下棋规则的命令时，客户端会从服务器端收到此命令。此命令后面紧跟 4 个字节的消息，其中字节序列的 C 形式定义如下(byte 为字节, 即 8 bits)：

```

//define reject_info for COMM_MSG_REJECTED
struct reject_info{
    byte reason;          //can be REJECT_REASON_UNWANTED or REJECT_REASON_INVALID_CHESS
    byte solution;        //can be SOLUTION_NOACTION or SOLUTION_ACTION_REPEAT
    byte parameter;       //can be one of the command begin with COMM_MSG_
    byte reserved;        //not used
};

```

其中 reason 的取值可以是 REJECT_REASON_UNWANTED 或 REJECT_REASON_INVALID_CHESS。其具体定义如下：

```

//type of reject reasons
//unwanted command, please solve this by reference to field solution of reject_info
#define REJECT_REASON_UNWANTED      1
//the position of the chess is invalid, e.g. taken position, position outside the chessboard,
//forbidden step for first-go player (禁手)
#define REJECT_REASON_INVALID_CHESS 2

```

其中 solution 的取值可以是 SOLUTION_NOACTION 或 SOLUTION_ACTION_REPEAT。其具体定义如下：

```

//type of reject reason solutions

```

```
#define SOLUTION_NOACTION          0    //no action needed
#define SOLUTION_ACTION_REPEAT    1    //please repeat this action with required parameters
```

其中 parameter 的取值可以是以 COMM_MSG_打头的数值，即：

```
//define commands for the socket communication
#define COMM_MSG_DONOTHING        0    //do nothing, ignore it!
#define COMM_MSG_FIRST           3    //go first when the game starts
#define COMM_MSG_SECOND          4    //go second when the game starts
#define COMM_MSG_GAME_REQUIRE_START 5    //request for game start
#define COMM_MSG_CHESS           7    //information for chess
```

其中的 reserved 字段保留不用。

✚ COMM_MSG_ACCEPTED

客户端可以收到此命令，当客户端提交给服务器的命令是一个合法的命令时，服务器会向客户端发此回应命令确认。

✚ COMM_MSG_FIRST

客户端或服务器可以收到此命令。服务器收到此命令时，向发送的客户端发送 COMM_MSG_ACCEPTED 命令后，会设定该客户端为先手，同时给另一客户端发送 COMM_MSG_SECOND 命令，设定另一客户端为后手。客户端收到此命令时，设定己方为先手。默认情况下，先连接服务器端者为先手，后连接者为后手。

✚ COMM_MSG_SECOND

客户端或服务器可以收到此命令。服务器收到此命令时，向发送的客户端发送 COMM_MSG_ACCEPTED 命令后，会设定该客户端为后手，同时给另一客户端发送 COMM_MSG_FIRST 命令，设定另一客户端为先手。客户端收到此命令时，设定己方为后手。默认情况下，先连接服务器端者为先手，后连接者为后手。

✚ COMM_MSG_GAME_REQUIRE_START

服务器端可以收到此命令。当服务器端已经有两个客户端连接，且棋局处于未开始或是已结束，此请求被接受，发送请求的客户端收到 COMM_MSG_ACCEPTED 消息，否则收到 COMM_MSG_REJECTED 消息。服务器端发送 COMM_MSG_ACCEPTED 消息后，会向两个客户端分别发送 COMM_MSG_GAME_START 消息。

✚ COMM_MSG_GAME_START

客户端可以收到此命令，当客户端收到此命令后，客户端必须将棋局设为初始状态，先手者然后发送 COMM_MSG_CHESS 消息走第一步棋。

✚ COMM_MSG_CHESS

服务器和客户端可以收到此命令。此命令后面紧跟 8 个字节的消息，其中字节序列的 C 形式定义如下(byte 为字节, 即 8 bits)：

```
//define the information for each step
struct chess_info{
    byte index;          //index for the chess, the client do not need to fill this field
    byte color;          //color of the chess
    byte row;            //row of the chess
    byte col;            //col of the chess
    byte reserved[4];    //reserved, just ignore it
};
```

服务器收到此命令及字节信息后，在内部进行处理后，如果消息正确，则发送 COMM_MSG_ACCEPTED 给客户端确认后，将此消息转发给两个客户端。否则发送

COMM_MSG_REJECTED 消息（一般情况由于客户端发送了已经走过的棋子或是其他原因）。此时请客户端重新发送 COMM_MSG_CHESS 及正确的 8 字节后续信息。

🚩 COMM_MSG_TIMEOUT

客户端可以收到此命令。为限制每一步的响应时间而定义。当客户端长时间不响应服务器的消息（即棋局开始后，客户端没有在规定时间内下出一棋子，人工智能长时间不能给出一个解），收到此命令后，客户端还会继续收到 COMM_MSG_LOSE 消息，另一客户端会收到 COMM_MSG_WIN 消息。棋局结束。

默认 TIMEOUT 的最大时间间隔为 10 秒。

🚩 COMM_MSG_WIN

客户端可以收到此命令。客户端收到此消息后，另一客户端必定收到 COMM_MSG_LOSE 消息。棋局结束。

🚩 COMM_MSG_LOSE

客户端可以收到此命令。客户端收到此消息后，另一客户端必定收到 COMM_MSG_WIN 消息。棋局结束。

🚩 COMM_MSG_DRAW

客户端可以收到此命令。平局。客户端收到此消息后，另一客户端必定收到 COMM_MSG_DRAW 消息。棋局结束。

Socket 收发数据时，请严格按以上信息的定义实现。

4 日志文件

作为服务器的裁判程序，对每走的一步棋，都记录了其信息，以日志形式记录。日志记录为 ASCII 码形式。具体格式如下：

```
//format of the log file
log_chess    index1    color1    row1      col1      time1
log_chess    index2    color2    row2      col2      time2
...
log_chess    indexN    colorN    rowN      colN      timeN
log_result   reasons
```

其中 log_chess 和 log_result 的值定义如下：

```
//definition for log type
enum log_type{
    log_chess    = 0, //type for chess informaiton
    log_result   = 1, //type for result, an reason will follow this field
};
```

当一局结束时，停止记录。此日志文件生成，作为证据备份。