Ai\_chess report

Mohamad Hosein Hoseinee

9712762670

در این کد از برای موارد داخلی شطرنج از کتابخانه

com.github.bhlangonijr.chesslib

استفاده شده است.

در ابتدای اجرای برنامه شی game ازکلاس Game ساخته می شود .

کلاس Game شامل موارد زیر است :

private final Player whitePlayer;  
private final Player blackPlayer;  
private final Board board;

این بازی را به صورتی میسازیم که سفید یک عامل هوشمند استفاده کننده از الگوریتم Min-max با عمق 3 است و مشکی بازیکنی است که در هر مرحله یک حرکت به صورت رندم انجام می ده و هوشمندی ندارد.

همچنین چینش مهره ها به صورت نرمال است.

Game game = new Game(new Agent(Side.*WHITE*), new RandomPlayer(Side.*BLACK*), GameVariation.*normal*);

پس از ساخت این کلاس با فراخوانی

game.play()

اجرای بازی شروع می شود

در حلقه while این بازی تمامی عملیات لازم برای بازی صورت می پذیرد.

while (true) {

ابتدا تابع play برای بازیکن سفید فراخوانی می شود ک به معنی شروع اولین حرکت برای اوست و او باید حرکت خود را اعلام نماید ( در داخل این تابع بخش اصلی پروژه که مربوط به هوش مصنوعی است پیاده سازی شده است) :

whiteMove = whitePlayer.play(board.clone());

پس از اینکه بازیکن سفید حرکت خود را اعلام نمود اکنون بررسی میکنیم در صورتی ک حرکتی انتخاب کرده و همچنین شی board اجازه این حرکت را می دهد پس این حرکت ثبت شده و چاپ شود در غیر این صورت چون بازیکن سفید حرکت مجازی ندارد بازیکن مشکی به عنوان برنده بازی شناخته می شود.

if (whiteMove == null || !board.doMove(whiteMove)) {  
 return blackPlayer;  
}else{  
 System.*out*.println("white move = " + whiteMove);  
}

پس از انجام این حرکت به بررسی اینکه آیا بازی مات یا مساوی شده است یا خیر میپردازیم ، در صورتی ک هیچ یک رخ نداده باشد همین عملیات برای بازیکن مشکی صورت می پذیرد و مجددا حلقه اجرا می شود.

این حلقه تا زمانی برقرار خواهد بود که یا بازیکنی قادر به انجام حرکت مجازی نباشد یا توابع isMated() , isDraw() که به بررسی مساوی بودن یا مات شدن یکی از بازیکنان می پردازند برابر با true شوند.

حال قصد داریم به بررسی تابع play که طی آن بازیکن سفید و مشکی حرکت خود را انتخاب میکنند بپردازیم.

@Override  
public Move play(Board board) {  
 List<Move> moves = board.legalMoves();  
 Move bestMove = null;  
 int bestMoveValue = Integer.*MIN\_VALUE*;  
 for (Move move : moves) {  
 if (board.doMove(move)) {  
 int temp = min(board.clone(), 1);  
 if (temp > bestMoveValue) {  
 bestMove = move;  
 bestMoveValue = temp;  
 }  
 board.undoMove();  
 }  
 }  
 if (bestMove == null) {  
 Random random = new Random();  
 return moves.get(random.nextInt(moves.size()));  
 }  
 return bestMove;  
}

لیست حرکت های ممکن را در یک داده ساختار List ذخیره میکنیم

List<Move> moves = board.legalMoves();

تک تک حرکت های ممکن را در روش min-max بررسی مینماییم تا بهترین حرکت یعنی max انتخاب شود.

for (Move move : moves) {

اگر حرکت قابل انجام است آن را انجام بده:

if (board.doMove(move)) {

اکنون در حرکت انجام شده متد min را با عمق 1 اجرا کن

int temp = min(board.clone(), 1);

اگر این حرکت min کمتری دارد پس آن را به عنوان بهترین جرکت انتخاب کن

if (temp > bestMoveValue) {  
 bestMove = move;  
 bestMoveValue = temp;  
}  
board.undoMove();

به همین ترتیب در داخل تابع min به فراخوانی تابع max در عمق بعدی و پس از حرکت انجام شده می پردازیم

بدین ترتیب الگوریتم min-max اجرا می شود.

لازم به ذکر است با تغییر مقدار maxDepth میتوان حداکثر عمق جستجوی min-max را تغییر داد.

private final static int *maxDepth* = 3;