report for project 3

:

description of design of my classes

1. scaffold: I create a grid, which treat + - | as a col, so the extra cols and rows are larger than the scaffold’s, but I invert them every time when they are being used. Then I just assigned each spot to be the correct char value, and print out them when display is being called. I use two for loops, the loop of row is outer, because when displayed, the scaffold should be up to down showed. For make move and undo move, I use two stacks to save the last move’s col and color, so that I can get access to it later, and if I already erased one, I can continue erasing because the stack has records everything.
2. game: It’s really hard to check whether it is complete or not after one move has been made. I record every last move been made, and I then check whether this move will result in any of the four cases: horizontally connected to N, vertically connected to N, upper left diagonal connected to N, upper right diagonal connected to N. Some directions need to be checked in two sides, and sum the two return value up to evaluate. Take turn function is just about color, so I separate it to two cases. Play is a multiple take turns with one step to print out the result.
3. support: I use a rating function, which is really similar to the game’s complete function, but returns a positive big number or a negative big number, so that I can get to know whether the move just taken is good or bad. Also I create a recursive function, which returns a pair. I will say it later.
4. determineBestMove function: I first set a boolean that computer played to be true, so every time bool is true, the function will return the max number, just because computer player will always want the max points. Every time the function is called, I will create an array of points, which equals in size to the number of cols the scaffold has. Then for each col, if it is valid, I will do a move, and then check whether this move results in a ending condition. If it does, store the value returned by the rating function in the array’s place which is one on one to the col. If it doesn’t, I then recursive call this function. If the col isn’t valid, I will store a junk value( which depends on the boolean) in the array’s place which is one on one to the col. At the end of this function, I will, depends on the boolean, iterate all the points saved in the array, and find the max or min, and return it to it’s upper array, which is a recursive process until the best point is chosen out in the root array. Every time I get the best point, I will save it, and at the same time, save the col of that move.

pseudocode

completed in game is nearly the same to rate, I will just write rate:

if sequence is true

check downward

if N connected, set point to a positive big number minus depth taken

break out of the loop

else, reset number connected to 1

check horizontally

check left part

if N connected, set point to a positive big number minus depth taken

else, record the number of connected

check right part

add the number get from left with that of right

if N connected, set point to a positive big number minus depth taken

break out of the loop

else, reset number connected to 1

check diagonally, lower left to upper right

check left part

if N connected, set point to a positive big number minus depth taken

else, record the number of connected

check right part

add the number get from left with that of right

if N connected, set point to a positive big number minus depth taken

break out of the loop

else, reset number connected to 1

check diagonally, upper left to lower right

check left part

if N connected, set point to a positive big number minus depth taken

else, record the number of connected

check right part

add the number get from left with that of right

if N connected, set point to a positive big number minus depth taken

break out of the loop

else, reset number connected to 1

check special case

if game isn’t over, scaffold is full

set point to zero

if sequence is true

check downward

if N connected, set point to a negative big number minus depth taken

break out of the loop

else, reset number connected to 1

check horizontally

check left part

if N connected, set point to a negative big number minus depth taken

else, record the number of connected

check right part

add the number get from left with that of right

if N connected, set point to a negative big number minus depth taken

break out of the loop

else, reset number connected to 1

check diagonally, lower left to upper right

check left part

if N connected, set point to a negative big number minus depth taken

else, record the number of connected

check right part

add the number get from left with that of right

if N connected, set point to a negative big number minus depth taken

break out of the loop

else, reset number connected to 1

check diagonally, upper left to lower right

check left part

if N connected, set point to a negative big number minus depth taken

else, record the number of connected

check right part

add the number get from left with that of right

if N connected, set point to a negative big number minus depth taken

break out of the loop

else, reset number connected to 1

check special case

if game isn’t over, scaffold is full

set point to zero

for the determine best move function:

if the scaffold is vacant

iterate the left half cols

make move

rate that move recursively, and save the points to the array of that base

if sequence is true

set best to a junk value, -5000

else

set best to a junk value, 5000

iterate the storage

find extreme value and update column and point

undo the move

return the pair of col and point

check every col

if valid to put in a checker

make move

find the exact location with row and col position

if the game is over due to that move

find extreme value and update column and point

undo the move

return the pair of col and point

if the game isn’t over

if it’s col number is big

restrict searching depth to 5 to accelerate

change color and boolean

recursive call this function, store the point in the array

if it’s col number is not big

restrict searching depth to 8 to accelerate

change color and boolean

recursive call this function, store the point in the array

undo the move, reset the color and boolean

if not valid to put in a checker

store a junk value into the array

set best to a junk value first

iterate all the numbers saved in the array

find extreme value and update column and point

return the pair of best column number and best score

Difficulties of this project:

It’s really hard to deal with game with lare numbers of possible moves in a short time, I don’t know how to improve it. I try to use depth as a restriction, but it is risky, for it will result in less number of consideration, which could cause a bad decision.

Another thing is that, I think my complete function is too complex, there must be a way that can make it clearer and shorter, and maybe furthermore accelerate the speed.

test cases for this project

/////////////////———————————-test scaffold————————————//////////////////

// Scaffold a(3, 2);

// assert(a.cols() == 3 && a.levels() == 2); // **test cols() and levels()**

// assert(a.numberEmpty() == 6);

// assert(a.checkerAt(1, 1) == VACANT);

// assert(a.makeMove(1, 1) == true && a.checkerAt(1, 1) == BLACK && a.numberEmpty() == 5); // **test makemove() & numberEmpty()** **&** **checkerat()**

// assert(a.undoMove() == 1); // **test undo()**

// assert(a.makeMove(3, 1));

// assert(a.undoMove() == 3);

// assert(a.undoMove() == 0);

//

// /////////////////———————————test players———————————/////////////////////

// BadPlayer b("nachensmallberg");

// assert(b.name() == "nachensmallbergl"); / / **test get name()**

// assert(!b.isInteractive()); // **test isInteractive**

// Scaffold c(1, 1);

// assert(b.chooseMove(c, 1, 1) == 1);

// c.makeMove(1, 1);

// assert(b.chooseMove(c, 1, 1) == -1); // **test badPlayer() make move**

//

// HumanPlayer d("me");

// assert(d.name() == "me"); // **test get name()**

// assert(d.isInteractive()); // **test isInteractive**

// Scaffold e(1, 1);

// e.makeMove(1, 1);

// assert(d.chooseMove(e, 1, 1) == -1); // **test HumanPlayer() make move**

//

// ///////////////———————————-test Game class—————————/////////////////////

// HumanPlayer f("tony");

// HumanPlayer g("steven");

// int i = -5;

// Game h(1, 1, 1, &f, &g);

// assert(!h.completed(i)); // **test completed()**

// assert(h.takeTurn());

// assert(h.completed(i));

// //assert(i == 0); // **test completed()**

// assert(!h.takeTurn()); // **test takeTurn()**

**in order to test smartPlayer, I use it as a humanPlayer in the online game provide in the spec. It behaves good, but not super smart.**