Project 3 Report

1. Scaffold class

Scaffold class has 4 private data members. Two int members m\_ncol and m\_nlev indicate numbers of columns and levels of the scaffold. A dynamically allocated vector<int> is used to store dropped checkers. There are m\_col vector pointers and each pointer points to vector representing a column. When a player drops a checker to a certain column, a color is pushed into the vector of that column. A stack<int> member m\_step records the order of the columns that the checker is drops into.

int m\_ncol, m\_nlev;

vector<int> \*m\_col;

stack<int> m\_step;

Other than the provided member functions, I also declare copy constructor, assignment operator and destructor for the Scaffold class. In order to implement assignment operator, I add a swap() function. Here are pseudocodes for some non-trivial functions.

int numberEmpty() const

multiply column number and level number to calculate the total size of the scaffold

count elements of each column vector and get the sum

total size minus sum of checkers to get the number of empty

int checkerAt(int column, int level) const

if column or level is out of range

output error and exit

if level is larger than column-1 vector’s size

return vacant

else

return value of that position

void display() const

repeatedly

from column 1, level m\_nlev, check out the color of each position

if red, output as |R

if black, output as |B

if vacant, output as |blank space

after each row, output | endl

output m\_ncol -+ and one +

bool makeMove(int column, int color)

check if column is in range, color is black or red

if true

push color to vector of column-1

push column to m\_step

return true

return false

int ScaffoldImpl::undoMove()

check if m\_step empty

if empty

return 0

if not empty

get the top of m\_step i and pop it our

pop the top of vector of i-1

return i

1. HumanPlayer class

int HumanPlayerImpl::chooseMove(const Scaffold& s, int N, int color)

check the number of empty of scaffold

if number > 0

prompt human player to choose column to move until get valid column

return column

else

return -1

1. BadPlayer class

A badplayer chooses leftmost non-full column to move

int BadPlayerImpl::chooseMove(const Scaffold& s, int N, int color)

check the number of empty of scaffold

if number > 0

repeatedly

from the leftmost column check if has vacant position

if has

drop checker

return column

else

return -1

1. SmartPlayer

In order to implement the choosemove() function, I add three non-member functions. rating function indicates the state of scaffold. After the player make move, it is not possible that this player will lose. The only outcomes that a player make a move is winning, tiegame and unfinished. If someone wins, it returns 1000-level. If the game draws, it returns 0. If the game is unfinished, it return -1. We set level here is because we wish to win early and delay lose. Therefore, early move has larger weight. Switchcolor function switches color between red and black. Determine function is basic procedures of minimax algorithm.

For implementation of choosemove function, I use minimax algorithm by determine function. At first, I create a map to record the move and its corresponding rate. For each possible move player can make, make the move. Then we call rating function to check the outcome of this move and get the rate. If the rate is not -1, which means the player wins or the game draws, we map this move and the rate, undo the move and return the column of this move. Otherwise, we switch the color, increment level and call determine function recursively to get the best move the component can move. Then we undo the move. It is noteworthy that either the smart player or its component will choose the move with highest rate. When doing recursion, we don’t need to consider whether this move is made by the player or by its component. So we always choose the move with the highest move. When the loop is over, we get the last elements in the map, which has the greatest value of rate and return its column.

For judging if the player wins, I declare winHorizonally, winVertically and winDiagonally and a win function to combine their results.

bool winHorizontally(Scaffold &s, int N, int &color)

for every row of scaffold

from the first column, count the consecutive same color checkers

if count == N

return true

return false

bool winVertically(Scaffold &s, int N, int &color)

for every column of scaffold

from the first row, count the consecutive same color checkers

if count == N

return true

return false

bool winDiagonally(Scaffold &s, int N, int &color)

for every diagnonal of scaffold

count the consecutive same color checkers

if count == N

return true

return false

int rating(const Scaffold& s, int g, int level)

copy s to ss

if win

return 1000 – level

else

if tie

return 0

return -1

int determine(Scaffold& s, int N, int color, int &rate, int level)

set map between column and rate

for each possible column to move

make the move

rate the move

if rate != -1

record column and rate

undo the move

return column

switch color

call determine(s, N, color, r, level+1)

record the column and rate

switch color

undo move

find the max rate in the map

return the column of max rate

int SmartPlayerImpl::chooseMove(const Scaffold& s, int N, int color)

set r to -1

copy s to ss

return determine function

1. Game class

Game class has 5 data members:

Scaffold \*m\_s;

Player \*m\_red, \*m\_black;

int goal;

int turn;

Goal is the number of consecutive checkers to win the game, turn indicate the color of current move.

bool GameImpl::completed(int& winner) const

if win

return true

else

if tie

return true

return false

bool GameImpl::takeTurn()

if win

return false

else

if current color is red

make move

set color to black

if current color is black

make move

set color to red

display scaffold

return true

void GameImpl::play()

output operation hint

repeatedly

switch turn and make move

if game is completed

check outcome and output it

break

1. Test case

/\*\*\*\*\*\*\*\*\*\*test scaffold class\*\*\*\*\*\*\*\*\*\*\*/

#include "provided.h"

#include <iostream>

#include <cassert>

using namespace std;

void doScaffoldTests()

{

Scaffold s(3, 2);

assert(s.cols() == 3 && s.levels() == 2 &&

s.numberEmpty() == 6);

assert(s.makeMove(1, RED));

assert(s.makeMove(1, BLACK));

assert(!s.makeMove(1, RED));

assert(s.numberEmpty() == 4);

s.display();

assert(s.checkerAt(1, 1) == RED && s.checkerAt(1, 2) == BLACK);

assert(s.checkerAt(2, 1) == VACANT);

assert(s.undoMove());

assert(s.checkerAt(1, 2) == VACANT);

assert(s.undoMove());

assert(s.checkerAt(1, 1) == VACANT);

s.display();

}

int main()

{

doScaffoldTests();

cout << "Passed all tests" << endl;

}

/\*\*\*\*\*\*\*\*\*\*test player class\*\*\*\*\*\*\*\*\*\*\*/

#include "provided.h"

#include <iostream>

#include <cassert>

using namespace std;

void doPlayerTests()

{

HumanPlayer hp("Marge");

assert(hp.name() == "Marge" && hp.isInteractive());

BadPlayer bp("Homer");

assert(bp.name() == "Homer" && !bp.isInteractive());

SmartPlayer sp("Lisa");

assert(sp.name() == "Lisa" && !sp.isInteractive());

Scaffold s(3, 2);

s.makeMove(1, RED);

s.makeMove(1, BLACK);

cout << "=========" << endl;

int n = hp.chooseMove(s, 3, RED);

cout << "=========" << endl;

assert(n == 2 || n == 3);

n = bp.chooseMove(s, 3, RED);

assert(n == 2 || n == 3);

n = sp.chooseMove(s, 3, RED);

assert(n == 2 || n == 3);

}

int main()

{

doPlayerTests();

cout << "Passed all tests" << endl;

}

/\*\*\*\*\*\*\*\*\*\*test game class\*\*\*\*\*\*\*\*\*\*\*/

#include "provided.h"

#include <iostream>

#include <cassert>

using namespace std;

void doGameTests()

{

BadPlayer bp1("Bart");

BadPlayer bp2("Homer");

Game g(2, 2, 2, &bp1, &bp2);

int winner;

assert(!g.completed(winner));

g.takeTurn(); // Red's first move

assert(!g.completed(winner) &&

(g.checkerAt(1, 1) == RED || g.checkerAt(2, 1) == RED));

g.takeTurn(); // Black's first move

assert(!g.completed(winner));

g.takeTurn(); // Red's second move; Red must win

assert(g.completed(winner) && winner == RED);

}

int main()

{

doGameTests();

cout << "Passed all tests" << endl;

}

/\*\*\*\*\*\*\*\*\*\*test player choosemove function\*\*\*\*\*\*\*\*\*\*\*/

#include "provided.h"

#include <iostream>

#include <cassert>

using namespace std;

void dochoosemoveTests()

{

BadPlayer bp("Homer");

HumanPlayer hp("Marge");

Game g(4, 3, 3, &bp, &hp);

g.play();

}

int main()

{

dochoosemoveTests();

cout << "Passed all tests" << endl;

}

#include "provided.h"

#include <iostream>

#include <cassert>

using namespace std;

void dochoosemoveTests()

{

SmartPlayer bp("Homer");

SmartPlayer hp("Marge");

Game g(4, 3, 3, &bp, &hp);

g.play();

}

int main()

{

dochoosemoveTests();

cout << "Passed all tests" << endl;

}