





cs380su21-meta.sg

[Dashboard](#) / [My courses](#) / [cs380su21-meta.sg](#) / [5 July - 11 July](#) / [Assignment 9 \(Adversarial Search\)](#)

 [Description](#)


 [Submission](#)

 [Edit](#)

 Submission view

Grade

Reviewed on Thursday, 15 July 2021, 4:47 PM by Automatic grade
grade: 100.00 / 100.00

Assessment report  [-]
[\[+\]](#) **Summary of tests**

Submitted on Thursday, 15 July 2021, 4:47 PM ([Download](#))

functions.cpp

```
1   /*!*****  
2  \file functions.cpp  
3  \author Vadim Surov, Goh Wei Zhe  
4  \par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu  
5  \par Course: CS380  
6  \par Section: B  
7  \par Programming Assignment 9  
8  \date 07-14-2021  
9  \brief  
10 This file has declarations and definitions that are required for submission  
11 *****/  
12  
13 #include "functions.h"  
14  
15 namespace AI  
16  {  
17  
18  
19 } // end namespace
```

functions.h

```

1  /*!*****
2  \file functions.h
3  \author Vadim Surov, Goh Wei Zhe
4  \par DP email: vsurov@digipen.edu, weizhe.goh@digipen.edu
5  \par Course: CS380
6  \par Section: B
7  \par Programming Assignment 9
8  \date 07-14-2021
9  \brief
10 This file has declarations and definitions that are required for submission
11 *****/
12
13 #ifndef FUNCTIONS_H
14 #define FUNCTIONS_H
15
16 #include <iostream>
17 #include <vector>
18 #include <cstring> // memcpy
19 #include <limits.h>
20
21 #include "data.h"
22
23 #define UNUSED(x) (void)x;
24
25 // Class that defines the game specific data and code
26 class Grid
27 {
28     static const int width = 3;
29     static const int height = 3;
30
31     char squares[width * height];
32
33 public:
34     // The game marks/pieces
35     static const char x = 'x';
36     static const char o = 'o';
37     static const char _ = ' ';
38
39     Grid(char* squares = nullptr)
40     : squares{ _, _, _, _, _, _, _, _, _ }
41     {
42         if (squares)
43             for (int i = 0; i < height * width; ++i)
44                 this->squares[i] = squares[i];
45     }
46
47     Grid(const Grid& rhs)
48     {
49         this->operator=(rhs);
50     }
51
52     void operator=(const Grid& rhs)
53     {
54         std::memcpy(squares, rhs.squares, height * width * sizeof(char));
55     }
56
57     void set(int i, char c)
58     {
59         squares[i] = c;
60     }
61
62     void clear(int i)
63     {
64         squares[i] = _;
65     }
66
67     char* getSquares()
68     {
69         return squares;
70     }
71
72     /*!*****
73     \brief
74     Function loops through a vector array to search through all empty squares
75     in the grid and returns a vector of indices.
76
77     For example, grid [' ',' ','o',' ','x',' ',' ',' ',' '], the function will
78     return [0,1,3,5,6,7,8]
79
80     \return
81     Returns a vector of the empty squares indexes.
82     *****/
83     std::vector<int> emptyIndices() const
84     {
85         std::vector<int> array = {};
86
87         for (int i = 0; i < width * height; ++i)
88         {
89             if (this->squares[i] == ' ')
90                 array.push_back(i);
91         }
92
93         return array;
94     }
95
96     /*!*****
97     \brief
98     Function to determine player's Tic Tac Toe winning condition.
99
100    \param player
101    Player's game mark / pieces
102
103    \return
104    Returns true if the grid has a winning configuration for the player, else
105    return false.
106    *****/
107    bool winning(char player)
108    {

```

```

109
110     return (this->squares[0] == player && this->squares[1] == player &&
111             this->squares[2] == player) ||
112
113             (this->squares[3] == player && this->squares[4] == player &&
114             this->squares[5] == player) ||
115
116             (this->squares[6] == player && this->squares[7] == player &&
117             this->squares[8] == player) ||
118
119             (this->squares[0] == player && this->squares[3] == player &&
120             this->squares[6] == player) ||
121
122             (this->squares[1] == player && this->squares[4] == player &&
123             this->squares[7] == player) ||
124
125             (this->squares[2] == player && this->squares[5] == player &&
126             this->squares[8] == player) ||
127
128             (this->squares[0] == player && this->squares[4] == player &&
129             this->squares[8] == player) ||
130
131             (this->squares[2] == player && this->squares[4] == player &&
132             this->squares[6] == player);
133 }
134
135 /*!*****
136 \brief
137 An overloading insertion operator function that takes and return a stream
138 object.
139
140 \param os
141 Output stream to perform output.
142
143 \param rhs
144 Right hand side object.
145
146 \return
147 Returns the ostream through ostream.
148 *****/
149 friend std::ostream& operator<<(std::ostream& os, const Grid& rhs)
150 {
151     for (int j = 0; j < height; ++j)
152     {
153         if (j == 0)
154             os << "[";
155         else
156             os << " ";
157
158         for (int i = 0; i < width; ++i)
159         {
160             os << rhs.squares[j * width + i];
161
162             if (i == width - 1 && j == height - 1)
163                 os << "]";
164             else
165                 os << ",";
166         }
167
168         if (j != height - 1)
169             os << std::endl;
170     }
171
172     return os;
173 }
174 };
175
176 namespace AI
177 {
178     // A node of the game tree
179     template<typename T>
180     class Move
181     {
182     public:
183         T grid;          // Result of a move: new state of the game grid
184         int score;        // Score of the move
185         std::vector<Move*> next; // All possible next moves
186
187         // Index of the first move in member next that has the best score
188         int bestMove;
189         int spotIndex;    // Index of the move's spot (used for a visualization)
190
191         Move(T grid = {}, int score = 0,
192             std::vector<Move*> next = new std::vector<Move*>{},
193             int bestMove = -1)
194             : grid{ grid }, score{ score }, next{ next }, bestMove{ bestMove },
195             spotIndex{ -1 }{}
196
197         /*!*****
198         \brief
199         Destructor for class Move
200         *****/
201         ~Move()
202         {
203             //Delete each move object in vector next
204             for (size_t i = 0; i < next->size(); ++i)
205             {
206                 delete (*next)[i];
207             }
208
209             //Remove vector elements, make vector size to 0.
210             next->clear();
211
212             //Delete pointer to vector
213             delete next;
214         }
215
216         /*!*****

```

```

217 \brief
218 Function to move object to specific index location
219
220 \param i
221 Index position to be moved to
222
223 \return
224 Returns a reference to Move object
225 *****/
226 Move& at(int i)
227 {
228     return *((*next)[i]);
229 }
230
231 int getScore() const
232 {
233     return score;
234 }
235
236 void setSpotIndex(int i)
237 {
238     spotIndex = i;
239 }
240
241 friend std::ostream& operator<<(std::ostream& os, const Move& rhs)
242 {
243     os << rhs.grid << std::endl;
244     os << rhs.score << std::endl;
245     os << rhs.next->size() << std::endl;
246     os << rhs.bestMove << std::endl;
247     return os;
248 }
249 };
250
251 /***/
252 \brief
253 Function to find the best move for maximizer. For initial call, set player
254 parameter as maximizer. However, found solution (sequence of moves) may not
255 be necessary the shortest.
256
257 \param grid
258 Current state of the Tic Tac Toe board.
259
260 \param player
261 Player's game mark / pieces
262
263 \param maximizer
264 Maximizer's game mark / pieces
265
266 \param minimizer
267 Minimizer's game mark / pieces
268
269 \return
270 Returns a pointer to Move object
271 *****/
272 template<typename T>
273 Move<T>* minimax(T grid, char player, char maximizer, char minimizer)
274 {
275     if (grid.winning(minimizer))
276         return new Move<T>(grid, -10);
277     else if (grid.winning(maximizer))
278         return new Move<T>(grid, 10);
279
280     std::vector<int> availSpots = grid.emptyIndices();
281
282     if (availSpots.size() == 0)
283         return new Move<T>(grid, 0);
284
285     std::vector<Move<T>*> next_ = new std::vector<Move<T>*>;
286
287     for (size_t i = 0; i < availSpots.size(); ++i)
288     {
289         grid.set(availSpots[i], player);
290
291         AI::Move<T>* move = minimax(grid,
292                                     (player == maximizer) ? minimizer : maximizer,
293                                     maximizer,
294                                     minimizer);
295
296         move->setSpotIndex(availSpots[i]);
297         next_->push_back(move);
298         grid.clear(availSpots[i]);
299     }
300
301     int bestScore_ = 0;
302     int bestMove_ = -1;
303
304     if (player == maximizer)
305     {
306         bestScore_ = -INT_MAX;
307
308         for (size_t i = 0; i < next_->size(); ++i)
309         {
310             if ((*next_)[i]->getScore() > bestScore_)
311             {
312                 bestScore_ = (*next_)[i]->getScore();
313                 bestMove_ = i;
314             }
315         }
316     }
317     else
318     {
319         bestScore_ = INT_MAX;
320
321         for (size_t i = 0; i < next_->size(); ++i)
322         {
323             if ((*next_)[i]->getScore() < bestScore_)
324                 bestMove_ = i;
325         }
326     }
327     return next_[bestMove_];
328 }

```

7/15/2021

cs380su21-meta.sg Assignment 9 (Adversarial Search) Submission view

325
326
327
328
329
330
331
332
333
334
335
336
337

```
bestScore_ = (*next_)[i]->getScore();
bestMove_ = i;
}
}
}
// Return the move
return new Move<T>(grid, bestScore_, next_, bestMove_);
}
} // end namespace
#endif
```

VPL

◀ Showcase: Tic-Tac-Toe Game

Jump to...

Minimax. Example of a quiz question ▶

You are logged in as [Wei Zhe GOH](#) ([Log out](#))
[cs380su21-meta.sg](#)
[Data retention summary](#).
[Get the mobile app](#)

https://distance3.sg.digipen.edu/2021sg-summer/mod/vpl/forms/submissionview.php?id=13930&userid=119

5/5