Life

Julian van Doorn (s2518074)

December 8, 2019

1 Description / Explanation

In this assignment we had to recreate the game "n-on-a-row". The game will start by asking you how big the field must be and how many on a row you want to play. It then allows you to play against the computer, against another human or let the computer play against itself.

When you play you can calculate the remaining amount of games (takes very long — not recommended for board sizes above 3). You can undo your last turn or you can quit. If the computer plays against itself it will store the result in a file called *summary-**widthxheight**.dat* where width and height are the height of the board. This can easily be plotted using a program such as *gnuplot*.

To demonstrate this last feature we let the computer play n-on-a-row on a n-by-n-field for $3 \leq n \leq 10$. It gave the result as seen in Figure 1.

We notice that almost all games end in a tie. Which is logical as all moves are generated randomly.

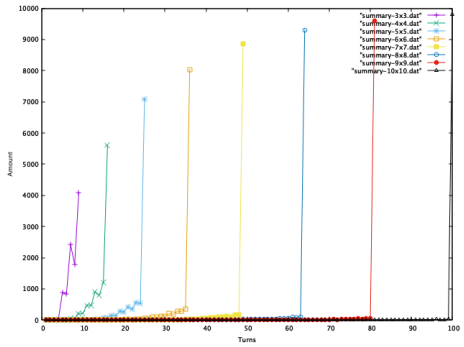

Figure 1: Results of n-on-a-row on a n-by-n-field for $3 \leq n \leq 10$, plotted using gnuplot.

2 Time

I had some issues designing additional functions but creating the pointer field went pretty smooth.

Time	Description
4h	Creating basic functionality
4h	Debugging and optimizing
2h	Writing report and documentation

3 Code

makefile

```
all: main.o board.o
    g++ -Wall -std=c++11 -o doorn4 board.o main.o
board.o: board.cpp board.h
    g++ -Wall -std=c++11 -c board.cpp
main.o: main.cpp board.h
    g++ -Wall -std=c++11 -c main.cpp
```

board.h

```
/*
 * board.h
 *
 * Headerfile of board.cpp.
 *
 * Last edited on: Sunday December 8th 2019
 */

class Field {
    /**
     * Represents a field on the board.
     */
    public:
    char value = ' '; // Value of the field
    Field *neighbours[8] = {nullptr}; // Its neighbours
};

class Action {
    /**
     * Represents an action so we can undo
     * it.
     */
    public:
    Action(int _x, int _y, Action *_previous);

    int x; // Its x coordinate
```

```
27      int y; // Its y coordinate
28      Action *previous = nullptr; // The action before it
29  };
30
31  class Board {
32      /**
33       * Represent the board.
34       */
35      private:
36      Field *start = nullptr; // Start of the pointer field (0,
            0)
37      Action *last_action = nullptr; // The last action
38
39      int width, height, amount; // Width, height and how many
            on a row
40      bool player1, player2; // Whether player 1/2 are played by
             a human
41
42      char player1char = 'B', player2char = 'W'; // Character
            for player 1/2
43
44      bool turn = false; // False means its player one's turn
45      int turns = 0; // The amount of turns we had in the
            current game
46
47      int *turns_keeper; // Keep track of how many games took a
            specific turn count
48      int p1_wins = 0, p2_wins = 0, ties = 0; // Player 1/2 wins
             and ties counters
49
50      /**
51       * Returns the amount of filled fields.
52       */
53      int filled();
54
55      /**
56       * Checks if the board is full.
57       */
58      bool full();
59
60      /**
61       * Calculates the score of field in  a given
62       * direction.
63       */
64      static int score(Field *target, int direction);
65
66      /**
67       * Check if the given field at x, y has
68       * a score high enough in any direction.
69       */
```

```
70      bool check(int x, int y);
71
72      /**
73       * Get instructions of the user and acts
74       * accordingly.
75       */
76      void user_controls(int &x, int &y, bool &q, bool &c);
77
78      /**
79       * Generates a random move for the computer.
80       */
81      void computer_controls(int &x, int &y);
82
83      /**
84       * Prints the field including an x
85       * and y axis with coordinates. Works
86       * perfectly up to 1000x1000 fields.
87       */
88      void print();
89
90      /**
91       * Prints the result of a game.
92       */
93      void print_result(bool won);
94
95      /**
96       * Set a specific field on the board
97       * to a value using the given x and
98       * y coordinate.
99       */
100     bool set(int x, int y, char value);
101
102     /**
103      * Gets a specific field on the board using
104      * the given x and y coordinate.
105      */
106     Field *get(int x, int y);
107
108     /**
109      * Save the last move.
110      */
111     void save(int x, int y);
112
113     /**
114      * Undo the last n-moves.
115      */
116     void undo(int times);
117
118     /**
119      * Clears the whole history.
```

```
120         */
121         void deconstruct_history();
122 
123         /**
124          * Calculate the remaining games.
125          */
126         long calculate();
127 
128         public:
129         /**
130          * Constructor of the Board class, should
131          * be self explanatory.
132          */
133         Board(int _height, int _width, int _amount, bool _player1,
                bool _player2);
134 
135         /**
136          * Constructs the pointer field of the board.
137          */
138         void construct();
139 
140         /**
141          * Prints a summary of the set of games and
142          * saves the results to a file.
143          */
144         void print_summary();
145 
146         /**
147          * Deconstruct the pointer field and history.
148          * Also clears some additional variables.
149          */
150         void deconstruct();
151 
152         /**
153          * The main function of the game, it
154          * handles all the logic and calls
155          * functions correspondingly.
156          */
157         void play();
158 
159         /**
160          * Cleans the board for a new game.
161          */
162         void clean();
163 };
```

board.cpp

```
1  /*
2   * board.cpp
```

```
 *
 * Last edited on: Sunday December 8th 2019
 */

#include <iostream>
#include <fstream>
#include "board.h"


Board::Board(int _height, int _width, int _amount, bool _player1, bool _player2) {
    height = _height;
    width = _width;
    amount = _amount;

    player1 = _player1;
    player2 = _player2;

    turns_keeper = new int[width * height];
}

void Board::construct() {
    Field *previous_row_start = nullptr; // Previous row
    Field *row_start = nullptr; // Current row
    Field *previous = nullptr; // Previous field
    Field *current = nullptr; // Current field

    for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
            current = new Field; // Create a new field

            if (col == 0) {
                row_start = current;

                if (row == 0) {
                    start = current;
                }
            } else {
                // Connect us to our left neighbour
                current->neighbours[3] = previous;
                current->neighbours[3]->neighbours[4] = current;
            }

            if (row > 0) {
                Field *above = previous_row_start; // The field above us
                for (int i = 0; i < col; i++) {
                    // Take the i-th element in the row above
                    above = above->neighbours[4];
```

```
50                  }
51
52                  // Connect us to our upper neighbours
53                  current->neighbours[0] = above->neighbours[3];
54                  current->neighbours[1] = above;
55                  current->neighbours[2] = above->neighbours[4];
56
57                  // Connect our upper neighbours to us and our
                        direct (left/right) neighbours
58                  above->neighbours[5] = current->neighbours[3];
59                  above->neighbours[6] = current;
60                  above->neighbours[7] = current->neighbours[4];
61
62              }
63              previous = current;
64          }
65          previous_row_start = row_start;
66      }
67  }
68
69  Field *Board::get(int x, int y) {
70      Field *target = start; // The target field
71
72      if (x > width - 1 || y > height - 1 || x < 0 || y < 0) {
73          return nullptr; // Field does not exist
74      }
75
76      for (int i = 0; i < y; i++) {
77          // Move down
78          target = target->neighbours[6];
79      }
80      for (int i = 0; i < x; i++) {
81          // Move to the right
82          target = target->neighbours[4];
83      }
84
85      return target;
86  }
87
88  bool Board::set(int x, int y, char value) {
89      Field *target = get(x, y); // Get the target
90      if (target == nullptr || (target->value != ' ' && value !=
            ' ')) {
91          return false; // Failed to update
92      }
93
94      target->value = value;
95      return true; // Successfully updated
96  }
97
```

```
98  void Board::print() {
99      Field *row = start;
100
101     std::string s1 = "\n", s2 = "\n", s3 = "\n";
102
103     // Creates strings for our x coordinates.
104     for (int i = width - 1; i >= 0; i--) {
105         std::string si = std::to_string(i);
106
107         if (i > 99) {
108             s1.insert(0, 1, si[2]);
109             s2.insert(0, 1, si[1]);
110             s3.insert(0, 1, si[0]);
111         } else if (i > 9) {
112             s1.insert(0, 1, ' ');
113             s2.insert(0, 1, si[1]);
114             s3.insert(0, 1, si[0]);
115         } else {
116             s1.insert(0, 1, ' ');
117             s2.insert(0, 1, ' ');
118             s3.insert(0, 1, si[0]);
119         }
120
121         // Spacing between the numbers on x axis
122         s1.insert(0, 1, ' ');
123         s2.insert(0, 1, ' ');
124         s3.insert(0, 1, ' ');
125     }
126
127     // Print the x axis
128     std::cout << "  " << s1 << "  " << s2 << "  " << s3;
129
130     // Print the y axis and all the field values
131     int r = 0;
132     while (row != nullptr) {
133         Field *col = row;
134
135         printf("%3d", r); // y axis
136         while (col != nullptr) {
137             std::cout << col->value << ' '; // Print field
138             col = col->neighbours[4]; // Go to the next
139         }
140         std::cout << std::endl; // Newline
141         row = row->neighbours[6]; // Go to next row
142         r++; // Increment for y axis
143     }
144 }
145
146 int Board::filled() {
147     return turns;
```

```
148 }
149
150 bool Board::full() {
151     return filled() == width * height;
152 }
153
154 int Board::score(Field *target, int direction) {
155     int score = 0; // The score in the given direction
156
157     Field *next = target->neighbours[direction];
158     while (next != nullptr && next->value == target->value) {
159         score++; // Increment score
160         next = next->neighbours[direction]; // Go to the next
                field
161     }
162     return score;
163 }
164
165 bool Board::check(int x, int y) {
166     Field *target = get(x, y);
167     if (target == nullptr) {
168         return false;
169     }
170
171     return score(target, 0) + score(target, 7) + 1 >= amount
            ||
172            score(target, 1) + score(target, 6) + 1 >= amount
                ||
173            score(target, 2) + score(target, 5) + 1 >= amount
                ||
174            score(target, 3) + score(target, 4) + 1 >= amount;
175
176 }
177
178 void Board::user_controls(int &x, int &y, bool &q, bool &c) {
179     std::cout << "Enter x-coordinate or negative number for
            options: ";
180     std::cin >> x;
181
182     if (x < 0) {
183         // Give menu options
184         std::cout << "1) Quit 2) Back 3) Undo 4) calculate: ";
185         std::cin >> y;
186
187         c = true; // Tells the game we had a menu (avoid
                calling set())
188         switch (y) {
189             case 1:
190                 q = true; // Quit the game
191                 break;
```

```
192             case 2:
193                 break; // Continue with the game
194             case 3:
195                 undo(2); // Undo twice (to their previous turn
                        )
196                 break;
197             case 4:
198                 std::cout << calculate() << " possible games
                        from this point.";
199                 break;
200             default:
201                 break;
202         }
203     } else {
204         std::cout << "Enter y-coordinate: ";
205         std::cin >> y;
206     }
207 }
208
209 void Board::computer_controls(int &x, int &y) {
210     while (true) {
211         y = rand() % height;
212         x = rand() % width;
213
214         if (get(x, y)->value == ' ') {
215             return;
216         }
217     }
218 }
219
220 void Board::print_result(bool won) {
221     if (!won) {
222         std::cout << "There was a tie after " << turns << "
                turns." << std::endl;
223     } else if (!turn) {
224         std::cout << "Player one has won after " << turns << "
                turns." << std::endl;
225     } else {
226         std::cout << "Player two has won after " << turns << "
                turns." << std::endl;
227     }
228 }
229
230 void Board::print_summary() {
231     std::ofstream data_file;
232     std::string file_name = "summary-" + std::to_string(width)
            + "x" + std::to_string(height) + ".dat";
233
234     data_file.open(file_name, std::ios::trunc);
235
```

```
236     std::cout << std::endl;
237     std::cout << "Player one has won " << p1_wins << " times
            and player two " << p2_wins << " times." << std::endl;
238     std::cout << "There were " << ties << " ties." << std::
            endl;
239
240     std::cout << "Turn statistics: " << std::endl << "Turns :
            amount" << std::endl;
241     data_file << "# Turns Amount" << "\n";
242     for (int i = 0; i < width * height; i++) {
243         // Print the amount of games that took n turns
244         printf("%5d : %d \n", i + 1, turns_keeper[i]);
245         data_file << i + 1 << " " << turns_keeper[i] << "\n";
246     }
247
248     data_file.close();
249 }
250
251 void Board::play() {
252     int x, y; // Coordinates
253     bool q = false, c = false; // Quit/continue flags
254
255     while (true) {
256         if ((player1 && !turn) || (player2 && turn)) {
257             print();
258
259             user_controls(x, y, q, c);
260             if (q) {
261                 return; // Quit
262             } else if (c) {
263                 c = false;
264                 continue; // Avoid failing an action
265             }
266         } else {
267             computer_controls(x, y);
268         }
269
270         if (set(x, y, !turn ? player1char : player2char)) {
271             turns++;
272             save(x, y);
273
274             if (check(x, y)) {
275                 print_result(true);
276                 turns_keeper[turns - 1] += 1; // Increment
                        turn
277                 !turn ? p1_wins++ : p2_wins++; // Increment
                        wins
278                 return;
279             } else if (full()) {
280                 print_result(false);
```

```
281                 turns_keeper[turns - 1]++; // Increment turn
282                 ties++; // Increment ties
283                 return;
284             }
285 
286             turn = !turn;
287         } else {
288             std::cout << "Failed to do action.";
289         }
290     }
291 }
292 
293 void Board::deconstruct_history() {
294     Action *p = last_action;
295     Action *n;
296     while (p != nullptr) {
297         n = p->previous;
298         delete (p);
299         p = n;
300     }
301 
302     last_action = nullptr;
303 }
304 
305 void Board::clean() {
306     turns = 0;
307     turn = false; // Player 1 turn again
308 
309     deconstruct_history();
310 
311     for (int x = 0; x < width; x++) {
312         for (int y = 0; y < height; y++) {
313             set(x, y, ' '); // Clear the board
314         }
315     }
316 }
317 
318 void Board::deconstruct() {
319     deconstruct_history();
320 
321     for (int y = height - 1; y >= 0; y--) {
322         for (int x = width - 1; x >= 0; x--) {
323             delete get(x, y);
324         }
325     }
326 
327     turns_keeper = nullptr;
328     p1_wins = 0, p2_wins = 0, ties = 0;
329 
330     start = nullptr;
```

```
331 }
332
333 void Board::save(int x, int y) {
334     last_action = new Action(x, y, last_action);
335 }
336
337 void Board::undo(int times) {
338     for (int i = 0; i < times; i++) {
339         Action *p = last_action;
340         if (p == nullptr) {
341             return;
342         }
343         set(p->x, p->y, ' ');
344         turns--; // We undid a turn so decrement
345         turn = !turn;
346
347         last_action = p->previous;
348         delete p;
349     }
350 }
351
352 long Board::calculate() {
353     long count = 0;
354
355     for (int x = 0; x < width; x++) {
356         for (int y = 0; y < height; y++) {
357             Field *p = get(x, y);
358
359             if (p->value == ' ') {
360                 set(x, y, !turn ? player1char : player2char);
361                 save(x, y);
362                 turns += 1;
363                 turn = !turn;
364
365                 if (full() || check(x, y)) {
366                     // We reached a final state
367                     undo(1);
368                     count += 1;
369                 } else {
370                     // Recursive call
371                     count += calculate();
372                     undo(1);
373                 }
374             }
375         }
376     }
377
378     return count;
379 }
380
```

```
381 Action::Action(int _x, int _y, Action *_previous) {
382     /**
383      * Constructor of the Action class, should be
384      * self explanatory.
385      */
386     x = _x;
387     y = _y;
388     previous = _previous;
389 }
```

main.cpp

```
1  /*
2   * main.cpp
3   *
4   * Assignment 4: n-on-a-row
5   * Author: Julian van Doorn (2518074)
6   *
7   * This is a game called n-on-a-row. It allows you to play
        yourself
8   * or simulate games. It also provides additional
        functionality as
9   * described in the task.
10  *
11  * Last edited on: Sunday December 8th 2019
12  * Compiled using cmake version 3.15.3 on macOS 10.15
13  *
14  */
15
16 #include <iostream>
17 #include "board.h"
18
19 bool is_affirmative(const std::string &value) {
20     /**
21      * Checks if an input is affirmative.
22      */
23     return value == "Y" ||
24            value == "y";
25 }
26
27 void print_description() {
28     /**
29      * Print a description of the game.
30      */
31     std::cout << "
          ==================================================" <<
          std::endl
32           << "|n-on-a-row
                                                  |" << std
                ::endl
```

```
33                << "|Author: Julian van Doorn (s2518074)
                                |" << std::endl
34                << "|The program will ask you for a couple
                      settings|" << std::endl
35                << "|and then lets you play or simulate the game
                      .   |" << std::endl
36                << "|

                      |" << std::endl
37                << "|Date: 8 December 2019
                                                |" << std::endl
38                << "
                      ================================================
                      " << std::endl;
39  }
40
41  void query_options(int &height, int &width, int &amount, bool
        &player1, bool &player2, int &count) {
42      /**
43       * Query the user for the game settings.
44       */
45      std::string holder;
46
47      std::cout << "What will be the height of the board: ";
48      std::cin >> height;
49      std::cout << "What will be the width of the board: ";
50      std::cin >> width;
51      std::cout << "How many on a row will we play: ";
52      std::cin >> amount;
53      std::cout << "Do you want a human to play player 1 (black)
            (Y/N): ";
54      std::cin >> holder;
55      player1 = is_affirmative(holder);
56      std::cout << "Do you want a human to play player 2 (white)
            (Y/N): ";
57      std::cin >> holder;
58      player2 = is_affirmative(holder);
59      std::cout << "How many games do you want to play: ";
60      std::cin >> count;
61  }
62
63  int main() {
64      /**
65       * Main function.
66       */
67      srand(time(nullptr));
68
69      print_description();
70
71      int height, width, amount, count;
```

```
72      bool player1, player2;
73
74      query_options(height, width, amount, player1, player2,
            count);
75
76      Board board = Board(height, width, amount, player1,
            player2);
77      board.construct();
78
79      for (int i = 0; i < count; i++) {
80          board.play();
81          board.clean();
82      }
83
84      if (!player1 && !player2) {
85          board.print_summary();
86      }
87
88      board.deconstruct();
89
90      return 0;
91  }
```