Hogeschool van Amsterdam

Game Technology

Titelblad

Studentnaam	**:**	**Jan-Willem Jozic**
Studentnummer	**:**	**500623980**
Plaats	**:**	**Hoofddorp**
Datum van uitgave	**:**	**17-2-2016**
Opleiding instituut	**:**	**Hogeschool van Amsterdam**

Introduction

This report documents the solution applied in the C++ project: Knight's Tour

It will walk you, the reader, through the code and motivates, how each requirement/exemplar is met and why such choices were made.

This program is the result of following tutorials on how to complete the knights tour.

Below are the requirements and exemplars listed for this project.

The entire code should already display that this project was written in the programming language C++

Requirements

- ☐ Programming language has to be C++.
- ☐ The program makes use of pointers and references.
- ☐ There is a visual representation. *(hint: see Eight Queens solution by Bill Weinman)*
- ☐ The board is at least 5x5 in size.
- ☐ The knight has to move according to its movement rules.
- ☐ The knight can only visit each square once.

Exemplar

- ☐ Board size can be set by the user *(NxN size).*
- ☐ The user can decide where the knight starts.
- ☐ Uses a more efficient solution than brute force *(explain in report required)*

These requirements are directly copied from the KnightsTour.pdf that was handed out.

Code review

This is the constructor that contains all the possible moves the knight can make.

```
Tour()
       {
              moves[0] = make_pair(2, 1);
              moves[1] = make_pair(1, 2);
              moves[2] = make_pair(-1, 2);
              moves[3] = make_pair(-2, 1);
              moves[4] = make_pair(-2, -1);
              moves[5] = make_pair(-1, -2);
              moves[6] = make_pair(1, -2);
              moves[7] = make_pair(2, -1);
}
```

This function sets up an array with all the possible moves for use later,

```
array<int, 8> sortMoves(int x, int y) //const
       {
              array<tuple<int, int>, 8> counts;
              for (int i = 0; i < 8; ++i)
              {
                     int dx = get<0>(moves[i]);
                     int dy = get<1>(moves[i]);

                     int c = 0;
                     for (int j = 0; j < 8; ++j)
                     {
                            int x2 = x + dx + get<0>(moves[j]);
                            int y2 = y + dy + get<1>(moves[j]);

                            if (x2 < 0 || x2 >= N || y2 < 0 || y2 >= N)
                                   continue;
                            if (data[y2][x2] != 0)
                                   continue;

                            c++;
                     }

                     counts[i] = make_tuple(c, i);
              }

              random_shuffle(counts.begin(), counts.end());

              sort(counts.begin(), counts.end());

              array<int, 8> out;
              for (int i = 0; i < 8; ++i)
                     out[i] = get<1>(counts[i]);
              return out;
}
```

This is the function that will solve the knights tour, it starts by running through the two arrays as the Horizontal a-h and the Vertical arrays 1-5.

After that it will run down the knights tour using the types of moves earlier defined.

If it doesn't find a working jump, it will backtrack and try again until the solution is found.

```
void solve(string start)
       {

              for (int v = 0; v < N; ++v)
              for (int u = 0; u < N; ++u)
                     data[v][u] = 0;

              int x0 = start[0] - 'a';
              int y0 = start[1] - '1';
              data[y0][x0] = 1;

              array<tuple<int, int, int, array<int, 8>>, N*N> order;
              order[0] = make_tuple(x0, y0, 0, sortMoves(x0, y0));

              int n = 0;
              while (n < N*N - 1)
              {
                     int x = get<0>(order[n]);
                     int y = get<1>(order[n]);

                     bool ok = false;
                     for (int i = get<2>(order[n]); i < 8; ++i)
                     {
                            int dx = moves[get<3>(order[n])[i]].first;
                            int dy = moves[get<3>(order[n])[i]].second;

                            if (x + dx < 0 || x + dx >= N || y + dy < 0 || y + dy >= N)
                                   continue;
                            if (data[y + dy][x + dx] != 0)
                                   continue;

                            ++n;
                            get<2>(order[n]) = i + 1;
                            data[y + dy][x + dx] = n + 1;
                            order[n] = make_tuple(x + dx, y + dy, 0, sortMoves(x + dx, y
+ dy));

                            ok = true;
                            break;
                     }

                     if (!ok) // Failed. Backtrack.
                     {
                            data[y][x] = 0;
                            --n;
                     }
              }
}
```

This is the operator that will print the knights tour output by placing each number at the correct position for both arrays

```
template<int N>
ostream& operator<<(ostream &out, const Tour<N> &b)
{
	for (int v = 0; v < N; ++v)
	{
		for (int u = 0; u < N; ++u)
		{
			if (u != 0) out << ",";
			{
				out << "[";
				out << setw(3) << b.data[v][u];
				out << "]";
			}
		}
		out << endl;
	}
	return out;
}
```

This is the main function, it starts out with explaining the knights tour to the user, between the printed messages. I've put sleep timers so that the user isn't being overwhelmed right away with text. The use of the sleep function was also so that the user can't write messages in the command prompt in between.

At the end the user can write down the position of where the knights tour should start,

After this they will get an output of the knights tour. If they press the escape button after this the program will shut down.

Because the user can decide where the knight will start its tour, the Exemplar has been met.

```
int main(int argc, char **argv)
{
	string location;
	cout << '\n' << "Welcome to the Knights Tour, " << '\n';
	Sleep(1000);

	cout << '\n' << "The board size will be " << 8 << " by " << 8 << " Now, select a 
starting position." << '\n' << "The starting position is defined like a real 
chessboard" << '\n';
	Sleep(500);
	cout << "Assign the start location by its starting Width," << '\n' << "going a 
or b or c up to h" << '\n' << '\n';
	Sleep(500);
	cout << "Then follow this up by deciding the Height" << '\n' << "Using a number 
between 1 and 8" << '\n' << '\n';
	Sleep(500);
	cout << "The end result should be something like: \'a1\' or \'c4\'" << '\n' << 
'\n'
		 << "After this, press Enter to let the knight run his tour" << '\n';
	cout << "Remember, you start from the top left," << '\n' << "a-h is for the 
Horizontal position," << '\n' << "1-8 is Vertical" << '\n' << '\n';

	cin >> location;



	Tour<8> board;
	board.solve(location);
	cout << board << endl;

	cout << '\n' << "The knights tour is over. Press the ESC button to exit the 
program";

	do {

	} while (GetAsyncKeyState(VK_ESCAPE) == 0);

			return 0;
}
```

Result

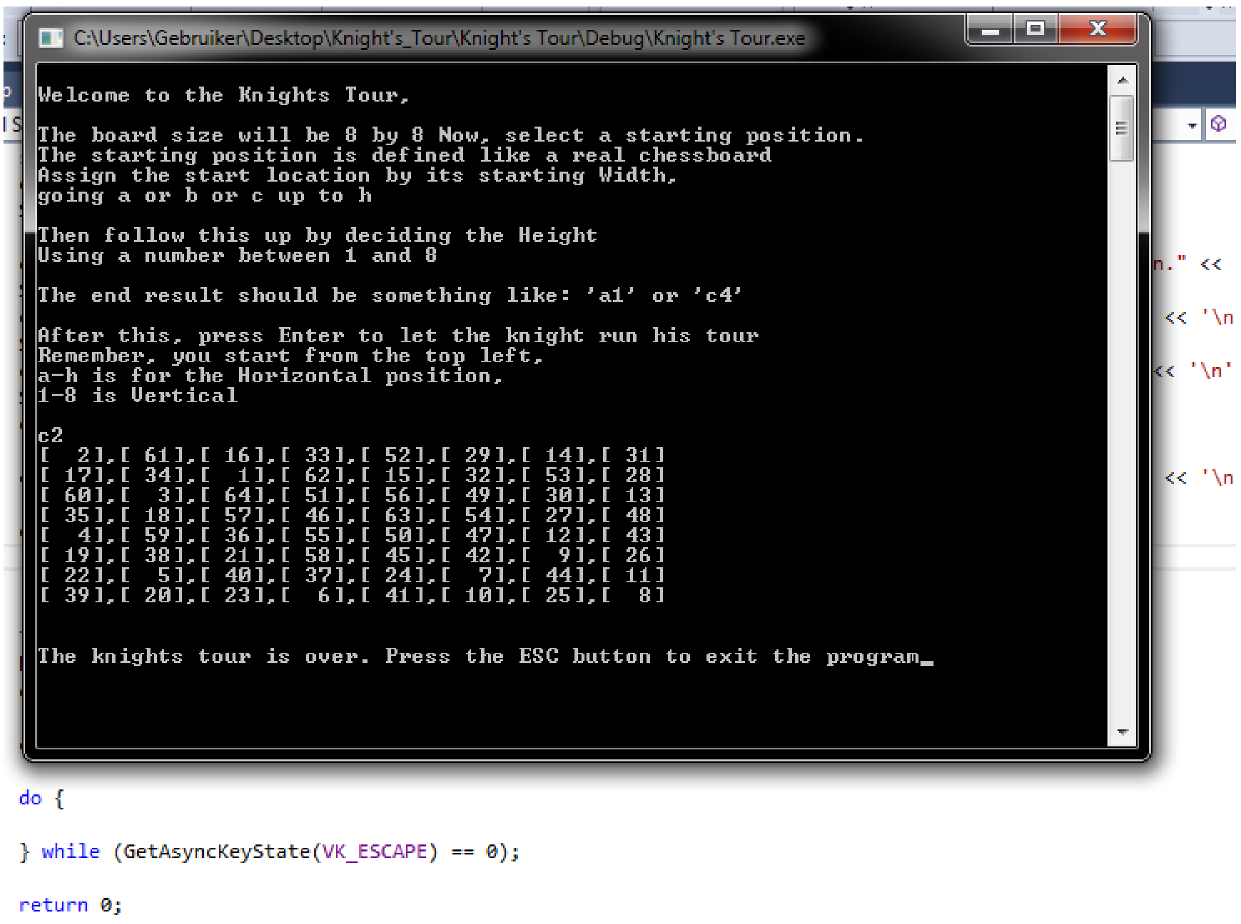
Here is a screen capture of the knights tour program,

As shown there is a visible grid that displays each step in its number, starting Step number one at c2 as was put in.

Then moving to a1, followed by b3 and so forth until its reached the end.

Thus fulfilling the requirement for a visual display of the knights tour and that it has a size bigger than 5x5, in this case 8x8.

The knight only moves in L shaped patterns and arrives on each location once.


C:\Users\Gebruiker\Desktop\Knight's_Tour\Knight's Tour\Debug\Knight's Tour.exe

```
Welcome to the Knights Tour,

The board size will be 8 by 8 Now, select a starting position.
The starting position is defined like a real chessboard
Assign the start location by its starting Width,
going a or b or c up to h

Then follow this up by deciding the Height
Using a number between 1 and 8

The end result should be something like: 'a1' or 'c4'

After this, press Enter to let the knight run his tour
Remember, you start from the top left,
a-h is for the Horizontal position,
1-8 is Vertical

c2
[  2],[ 61],[ 16],[ 33],[ 52],[ 29],[ 14],[ 31]
[ 17],[ 34],[  1],[ 62],[ 15],[ 32],[ 53],[ 28]
[ 60],[  3],[ 64],[ 51],[ 56],[ 49],[ 30],[ 13]
[ 35],[ 18],[ 57],[ 46],[ 63],[ 54],[ 27],[ 48]
[  4],[ 59],[ 36],[ 55],[ 50],[ 47],[ 12],[ 43]
[ 19],[ 38],[ 21],[ 58],[ 45],[ 42],[  9],[ 26]
[ 22],[  5],[ 40],[ 37],[ 24],[  7],[ 44],[ 11]
[ 39],[ 20],[ 23],[  6],[ 41],[ 10],[ 25],[  8]


The knights tour is over. Press the ESC button to exit the program_
```

```
do {

} while (GetAsyncKeyState(VK_ESCAPE) == 0);

return 0;
```

That should complete the knights tour with all written requirements met.