Advanced Robot Programming Labs
C++ Programming

Lab 1: Open projects

1 Content of this lab

The goal of the first C++ labs are to learn C++ by building your own project. Several small projects are listed below and you are encouraged to try and build several of them. Many of them can be done without using evolved concepts of C++.

You are encouraged to use functions, classes or even templates if you think it will lead to a better code that is easier to understand and to write. You are also encouraged to separate your code in several files. If you are not sure about classes or functions, you can always change it later and compare the same program with different code designs.

You may look for mathematical solutions, problem formulation or problem variations on Internet, but of course the actual code should come from yourselves.

1.1 How to compile

We use CMake tools to compile our projects. The content of the CMakeLists.txt file should follow the structure of your code directory (or preferably sub-directory): The structure of your code directory (or sub-directory should be like:

```
MyProject
├── include
│   ├── a_class.h
│   ├── some_functions.h
│   └── a_basic_header.h
├── src
│   ├── main.cpp
│   ├── a_class.cpp
│   └── some_functions.cpp
└── CMakeLists.txt
```

```
cmake_minimum_required (VERSION 2.6)
project (MYPROJECT)

add_compile_options(-std=c++11)

include_directories(include)

add_executable(mygame
  src/main.cpp include/a_basic_header.h
  include/a_class.h a_src/class.cpp
  include/some_function.h src/some_function.cpp)
```

The CMake file is used to generate a Makefile. The steps are:

1. Create a build directory: `mkdir build`
2. Go inside: `cd build`
3. Call CMake: `cmake ..`
4. Call Make: `make`
5. Then launch your program: `./mygame`

Centrale Nantes

2 Single player games

In the single player games, the opponent is the computer which plays with a basic or advanced AI depending on your programming motivation (and game AI possibilities...).

2.1 Find the number

Here both the players start by giving a number between 1 and 100 to the computer. Then they try to find the number of the other player in a minimum number of turns.

- This game involves: Input/output
- AI tips: Random numbers depend on seed

2.2 Rock-paper-scissors

The player is asked to choose between rock, paper or scissors. The computer also does a choice and the winner depends on the two choices:

- Rock beats Scissors
- Paper beats Rock
- Scissors beats Rock
- Draw in the case of equal choices
- This game involves: Input/output, Random numbers
- AI tips: none, this game is completely random

3 Turn-based text games

In these two-player games, the players play one at a time. The goal is usually to reach the winning situation before the other. They can also be programmed to play against an artificial intelligence. This time the AI can be much more complex than for the first games.

3.1 21 sticks game

In this game, a given umber of sticks (typically 21) are displayed. Each player has to pick up 1 to 3 sticks, then the other player can pick up his sticks. The goal is not to have to pick the last stick.

- This game involves: Input/output
- AI tips: If you can start correctly you may never lose.

3.2 Battleship

In this game each player has a number of boats that are placed on a 10×10 grid. The goal is to sink all the opponent's boats. Each player begins by placing their boats according to the following:

- One Aircraft carrier [A] of length 5
- One Cruiser [C] of length 4
- One Destroyer [D] of length 3
- One Submarine [S] of length 3
- One Minesweeper [M] of length 2

The player grids may be entered manually or with a text file (check grid consistency), or randomly generated. Each player then enters coordinates of the targeted grid cell. A boat can be hit, sunk or it may be a miss.

The current grids should be displayed with:

- A dot for a non-targeted cell
- A cross for missed shots
- The boat letter for hit or sunk boats

	0	1	2	3	4	5	6	7	8	9
0	x	.	.	.	.	.	.	.	.	.
1	.	.	.	.	C	C	x	.	.	.
2	.	x	.	.	.	.	.	M	.	.
3	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.
5	.	.	x	.	A	A	A	A	A	.
6	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	x	.	.
9	.	.	.	.	.	.	.	.	.	.

- This game involves: Input/output, Vectors, File input, Data representation
- AI tips: Do not target the same cell twice, finish what you started

3.3 4 in a row

This game is played on a 6×7 grid. Each player has to drop a token in one of the column, that will fall to the bottom or above potential existing tokens in this column. The goal is to be the first to align 4 tokens, either vertically, horizontally or diagonally.

- This game involves: Input/output, Vectors, Data representation

Centrale Nantes

- AI tips: Do not let the other player align 4 token, or try to guess best possible choices for the next 1, 2, 10... turns.

.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	o	.	x	.	.
.	o	x	x	o	o	.

4 Algorithms

In this section, we program the computer to solve a given problem through an algorithm. Similarly to a cooking recipe, an algorithm is a sequence of basic steps (like finding the minimum value in a vector) that lead to an overall more complex behavior.

The problem data may be defined in a file or be randomly generated. The use of the STL and of the `math.h` functions will be of good use.

4.1 Tower of Hanoi

This one is a classical mathematical puzzle. A given number of disks of different sizes are placed on three rods. The only constraint is that a bigger disk cannot be above a smaller one.

The goal is to move the disks one at a time (only the top disk of a rod may be moved to another rod) in order to have them all on a single rod. The Wikipedia page will give you strong hints to solve this problem.

The starting position of the disks may be randomly generated or written in a file. The goal is of course to use a minimum number of moves.

- This involves: File input, output, Vectors, Data representation, Recursive programming

4.2 Robust fitting with Hough lines

This method will be taught in Computer Vision. We assume we have a number of (x, y) points and want to find the line or lines that pass through them. A method was proposed by Hough and consists in finding all possible lines that pass through each point, and then count the similar lines. The ones that have the highest number of counts (or votes) are considered to be actual lines in the image. Do to so, we write a line equation as:

$$x \cos\theta + y \sin\theta = \rho$$

We then consider a finite set of possible θ and ρ, typically:

- θ can go from $-\pi/2$ to $+\pi/2$ with 100 intermediary values
- ρ can go from 0 to half the diagonal of the image with 100 intermediary values

We thus have a 100×100 grid of all potential (ρ, θ) corresponding to 10000 lines.

In practice, at the beginning of the program you can define X as [-10, .., 10] and chose one, then two values for (a, b). Then Y is computed as $Y = aX + b$ and you have your set of X and Y coordinates. You can also add some random numbers to make it more difficult to the algorithm.

The algorithm is given below.

Data: X and Y, min. votes n
Result: (ρ, θ) for best lines
init ρ_v and θ_v vectors from desired range and step;
RT←zero matrix of dim. 100×100;
for *i in range(100)* **do**
  for *(x,y) in (X,Y)* **do**
    $\rho \leftarrow x \cos\theta_i + y \sin\theta_i$;
    j←index of ρ_v nearest to ρ;
    RT[j,i]+= 1;
  end
end
(ρ, θ) are the values where RT[i,j] > n;

Algorithm 1: Hough line detector

- This involves: Output, Vectors, Sort and Find maximum (STL)

4.3 Path planning with A*

The A* algorithm is a popular path planning method that can easily find the shortest path from a starting to a desired position in a graph. A graph is a set of nodes (positions or situations) linked by elementary steps (for example, going from (0,0) to (0,1)). It may be a classical 2D or 3D space, but also any kind of graph where we want to find the shortest path.

- This involves: File input, output, Vectors, Data representation, Sort and find minimum (STL), Classes, Inheritance, Templates or References

4.3.1 General A* algorithm

The algorithm is based on two functions:

- The travel function $g(i)$ expresses the distance between the starting position and the evaluated one.
- The heuristic function $h(i)$ tries to estimate the cost from the current position to the desired one.

The total cost function is the sum of g and h. It is also using two sets of nodes:

- The closed set regroups all nodes that have been evaluated (starts at empty)
- The open set regroups all nodes that are to be evaluated (starts with the initial position)

```
Data: start and goal positions
Result: Sequence of nodes that define the shortest path
closedSet← [];
openSet← [start];
inside ← true;
start.g← 0;
start.compute_h(goal);
start.compute_f();
while openSet not empty do
    candidate ← node in openSet with lowest f score;
    if candidate == goal then
        return candidate;
    end
    openSet.remove(candidate);
    closedSet.add(candidate);
    for each neighboor of candidate do
        union←openSet + closedSet;
        if neighboor is in union then
            twin←neighboor from union set ;
            if twin.g > neighboor.g then
                twin.set_parent(candidate);
                twin.compute_f();
                openSet.add(twin);
            end
        else
            neighboor.compute_h(goal);
            neighboor.compute_f;
            openSet.add(neighboor);
        end
    end
end
```

Algorithm 2: A* algorithm

As we see, the algorithm does not depend on the type of the nodes, as long as the following can be defined:

- The h function from the node to the goal
- The *neighboor* function that gives the nearest nodes to the current node (`vector`?)
- The == operator that checks whether two nodes are the same

This is perfect for classes as the actual algorithm can be written in a generic way. I suggest you write it for a given class to test it without using templates.

4.3.2 Classical 2D path planning

This first A* problem consists of a 20×20 grid where a robot must go from $(0, 0)$ to $(19, 19)$. Obstacles exist of course in the grid. They can be defined randomly or in a text file that the program will read. In this case:

- The h function is the Euclidean distance to the goal.
- The *neighboor* function should give all the nodes reachable from a ± 1 move on x or y.
- The == operator corresponds to having equal x and y.

4.3.3 The 8-puzzle or 15-puzzle problem

In this problem, we have a 3×3 grid with cells numbered from 1 to 8[1], and an empty cell. The goal is to reach the following configuration:

1	2	3
4	5	6
7	8	.

We go from one position to another one by sliding one cell in the empty cell. For example, the two configurations that may lead to the desired one are:

1	2	3
4	5	6
7	.	8

1	2	3
4	5	.
7	8	6

In this case:

- The h function can be the Manhattan distance, that is:

$$h(x, y) = \sum_{i=1}^{8} (|x_i - x_i^*| + |y_i - y_i^*|)$$

  where (x_i, y_i) is the position of cell i in the grid and (x_i^*, y_i^*) is the position of cell i in the desired grid.
- The *neighboor* function should give all the grids reachables from the current one.
- The == operator corresponds to having the cells at the same positions.

Note that not all starting grids can reach the desired one. The best way to build a starting position is to start from the desired one and randomly move cells for 30 to 40 steps. The algorithm will usually find a much shorter path to come back.

5 Genetic algorithms

[1] can also be done on a 4×4 grid with numbers 1..15

Centrale Nantes