Introduction to Programming

— Practical #8 —

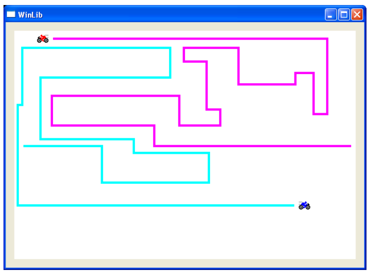

FIGURE 1 – Game of Tron.

1 Tron

In this practical, we are going to program the game of TRON. It is game with 2 players in which each controls a mobile that moves with constant speed and leaves behind a trace. The first player that crashes on its trace or the adversary's trace (or exits the window) loses the game. This practical is rather ambitious and is a mini-project. It will take us several sessions.

1.1 Snake

We will proceed in two steps. First program a game of Snake with a single player. In this game, the single player controls a snake that grows little by little (of one element each x round, with the convention that the total length is bounded by a constant $nmax$ elements).

The starting project has two files, `utils.h` and `utils.cpp`, that contain a structure point (that we may enrich with useful methods later) and a function meant to get the keys pressed by the players.

We will build an object `Snake` with adequate methods, and a function `game_1p` exploiting the capacities of the Snake to reproduce the desired behavior. First, we won't care about collisions (with the border or with itself), and only do that in a second time. Your work is split into 6 steps.

1. *(on paper)* Define an interface for class `Snake` (list all necessary functionalities).
2. *(on paper)* Think about implementation of the class `Snake` : how to store the data ? How to program the different methods ? (read first some remarks from next paragraph).
3. In a file `snake.h`, write the declaration of the class `Snake` : its members, its methods, what is public, what is not.
4. *Submit the result of your thoughts to the instructor to validate your choices.*
5. Implement the class `Snake` (that is, program the methods that you declared).
6. Program the function `game_1p` using a `Snake`.

Remark : the file `utils.h` defines :

1. 4 integers `left`, `down`, `up`, `right` such that :
   (a) the function $x \to (x+1)\%4$ transforms left to down, down to right, right to up and up to left ; it corresponds to a quarter turn in trigonometric sense.

   (b) the function $x \to (x+3)\%4$ makes the same but in clockwise direction. [1]

2. an array of 4 points `dir` such that, after definition of a function computing the sum of two points, the function $p \to p + \texttt{dir}[d]$ returns :
   (a) for $d =$ `left` the point corresponding to a shift of p of 1 unit left.
   (b) for $d =$ `up` the point corresponding to a shift of p of 1 unit up.
   (c) for $d =$ `right` the point corresponding to a shift of p of 1 unit right.
   (d) for $d =$ `down` the point corresponding to a shift of p of 1 unit down.

1.2 Tron

From the game of Snake, it will be easy to implement the game of Tron.

1. Two players.
   Inspired by function `game_1p`, create a function `game_2p` with two players. We will use for this player the keys `S`, `X`, `D` and `F`. [2] The function `keyboard()` will return the integers `int('S')`, `int('X')`, `int('D')` and `int('F')`. Remark : we treat only one key per round, hence a single call to function `keyboard()` per round, otherwise the snakes are in concurrency for the keyboard.
2. Ultimate tuning.
   (a) Handle the collision of the snakes.
   (b) The principle of Tron is that the trace of mobiles remains. To implement that, we just need to extend the snake at each round.

1.3 Graphics

Small bonus to make our game more attractive : we will see how to handle graphics instead of the uniform rectangles we had until now. The objective is to replace the heading square by an image that we will move at each step.

We are going to use the `NativeBitmap` of Imagine++, that are images that are faster to draw on screen then regular images. To put an image in a `NativeBitmap` we proceed like follows :

```
// Integers passed by reference when loading the image
// so as to store the width and height of the image
int w,h;
// Image loading
byte* rgb;
loadColorImage(srcPath("image_name.bmp"),rgb,w,h);
// Declaration of a NativeBitmap
NativeBitmap my_native_bitmap (w,h);
// Put the image in the NativeBitmap
my_native_bitmap.setColorImage(0,0,rgb,w,h);
delete [] rgb; // We don't need the image anymore
```

The display of a `NativeBitmap` on screen can be done using the function of Imagine++ :

```
void putNativeBitmap(int x, int y, NativeBitmap nb)
```

1. Replace in the snake the display of the head by a bitmap. You can use the images `moto_blue.bmp` and `moto_red.bmp` in the archive.
2. Use the image `explosion.bmp` to show the death of a player.

1.4 Make the code great

To have a clean code of which you may be proud, heed the following guidelines :
— The class `Snake` has one/several constructor(s).
— The class `Snake` has a destructor only if necessary. Hint : did some dynamic allocation occur ?
— No useless copy of a `Snake` when passing as argument of a function. Hint : are objects of type `Snake` passed by reference ? See the course about copy constructor to understand why it matters.

1. In mathematics, $(x+3)\%4 = (x-1)\%4$, but for C++ $-1\%4 = -1$, which is wrong for our purpose.
2. The keys `A`, `Z`, `Q` and `W` are not in the right configuration for a Qwerty keyboard, thus avoid them.

— Only methods used by the exterior hare public. All others must be private.
— The methods that do not change the Snake are const.
— The indentation is correct. Do we really have to repeat this obvious guideline ?
— The bitmaps are read with `srcPath` and the program stops if one image file is not found.