INFO0012-2/3 : Second Project (Parallel programming)

Deadline December 18th, 2017

General information

- **Deadline: December 18, 2017, 23:59.**
- Project must be done by **teams of two students**.
- There will be a penalty for late submission.
- Questions will no longer be answered 24 hours before the deadline.
- English is strongly encouraged.
- Contact: `S.Hiard@uliege.be`, Office I.112 (B28).

1 Introduction and terminology

For this project, you are asked to write a program that displays bouncy squares using the SDL media library for managing the display. We will use the following terminology:

- **Board:** The environment in which the bouncy squares move. It is displayed as an $N \times N$ array of pixels as depicted in Figure 1.
- **Screen:** The device on which the board is displayed.
- **Pixel:** The minimum controllable element on the screen.
- **Grid:** The inner representation of the board. It is an integer array of N rows and N columns (pixels are indexed from left to right and from top to bottom)
- **Square:** A colored square of 16×16 pixels on the board.

2 Squares and their Movements

Time progresses in discrete ticks and squares move at each tick according to the following rules.

- A square always occupies 16×16 pixels on the board.

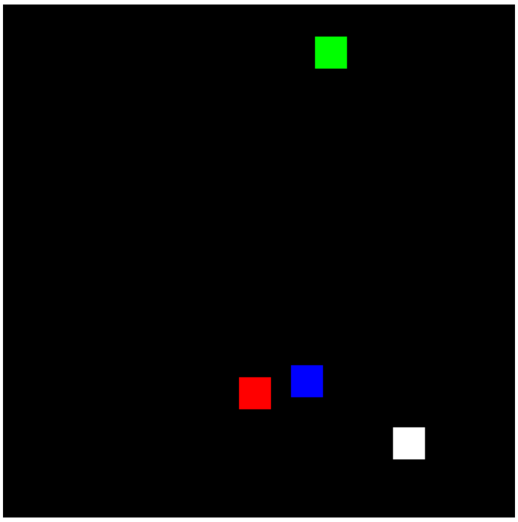

Figure 1: Graphical representation of the board

- Two squares cannot overlap.
- At each tick, a square can move by at most one pixel horizontally and one pixel vertically, provided that this movement does not cause two squares to overlap, or a square to move partially out of the board. A square thus moves in one among 8 possible directions[1].
- The internal representation of a square is a 5-tuple of integers containing (i) the X position of the upper left corner $\in [0, N-16[$, (ii) the Y position of the upper left corner $\in [0, N-16[$, (iii) the color of the square, (iv) the X velocity of the square $\in [-1, 1]$ and (v), the Y velocity of the square $\in [-1, 1]$.
- If a movement would lead to violating the first two rules, then the movement is not performed, and the direction of the square is changed as follows:
  - If the square is about to cross board boundaries, then its X (resp. Y, or both X and Y) velocity is reversed if crossing either a vertical (resp. horizontal, or a combination of horizontal and vertical) boundary of the board.
  - If two squares are about to overlap, then they simply swap their respective directions. If multiple squares collide, the one with the highest (and then leftmost) position swaps its direction with the lowest (and then rightmost) one.

[1] or rather 9, counting not moving at all as a direction

- If both situations occur simultaneously, the rule for crossing boundaries is applied before swapping directions.

3 Implementation

You are asked to write in C a parallel program that acts as described above.

- The number of squares should be a parameter of your program.
- Your program should provide the possibility of manually initializing (some or all of the) squares, the alternative being to randomly initialize a number of squares. The initialization, should not allow any overlap between squares.
- The board is displayed on the screen using the SDL library. An example fully functional single-process program using SDL is provided to help you.
- The X and Y coordinates of squares are stored in shared memory, but not their direction; a “master process” is used to handle the console inputs and SDL outputs. All processes have access to the data stored in shared memory.
- Access to the shared memory will be controlled with semaphores, if needed. You are not allowed to use active waits (i.e., repeatedly testing a condition in a loop).
- The user commands (including pressing <enter> to cleanly quit the program) are transmitted to the “master process” using terminal I/O.
- Each square is controlled by one and only one process, those processes being referred to as “workers”. Each worker is responsible for calculating the next position of its square. Only when all workers have computed the position for tick T can the master process update the display. When the display is performed, all workers can now compute the position at tick $T + 1$.
- Communication between the master process and the worker processes is handled using semaphores and shared memory.
- When a collision between two squares occur, their corresponding processes exchange their directions using blocking message queues.
- The master process can possibly be separated into several processes (e.g. one waiting for user input, and one waiting for worker completion for the current tick).

4 Submission procedure

- This project must be coded in C using the `System V IPCs` for the shared memory, the semaphores and the message queues. The display will be performed using the SDL library.

- You must write a report describing how you implemented your program, and in particular how the synchronization is performed. This can be done by representing the program using the simplified C syntax used in the problem-solving lessons. You may use functions such as `displayOnScreen()` or `computeNextPosition()` to simplify the pseudo-code in your report in order to focus on the inter-process communication.

- The project has to be done in teams of 2 students and completed before before December 18th at 23h59. The completed program and report (PDF only) will be included in a ZIP archive named `sXXXXXX_NAME1_sYYYYYY_NAME2.zip` where `sXXXXXX`, `NAME1`, `sYYYYYY`, and `NAME2`, are the student IDs and uppercase surnames of the team members.

  Submit your archive to the **Montefiore Submission Platform**[2], after having created an account if necessary. If you encounter any problem with the platform, let me know (S.Hiard@uliege.be). However problems that unexpectedly and mysteriously appear five minutes before the deadline will not be considered. **Do not send your work by E-mail; it will not be read**.

Good programming !

[2] http://submit.montefiore.ulg.ac.be/