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

Deadline December 19th, 2016

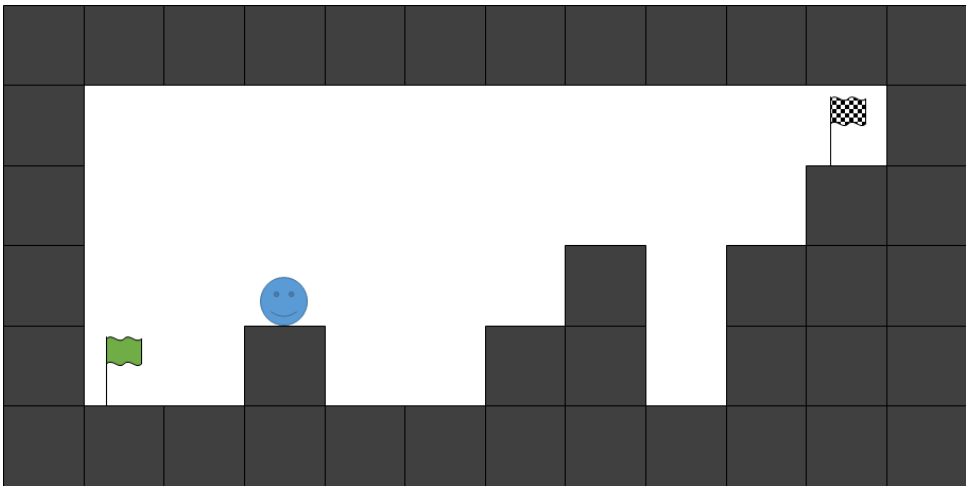

Figure 1: Graphical representation of the environment

For this project, you are asked to develop a program that will use a genetic algorithm to produce a creature perfectly fitted to a given environment. We will use the following terminology:

- **Grid:** The representation of the environment. It is a rectangle of $M \times N$ tiles as depicted in Figure 1.
- **Tile:** A space on the grid that can hold a creature, an obstacle, or nothing.
- **Creature:** Can only live inside the grid and moves over time from tile to tile.
- **Obstacle:** An occupied tile on the grid that creatures cannot pass through. Never moves over time.

1 Genetic algorithm

The goal of the genetic algorithm is to produce a creature perfectly fitted to the environment, by letting creatures evolve in successive generations and only allowing the best fitted to survive and reproduce.

The algorithm works in several steps :

1. Generate a population of C individuals at random.
2. Evaluate the performance of each individual and obtain its score.
3. Sort the individuals based on their score.
4. Delete $p\%$ of the population where the least successful individuals gets deleted first.
5. Replicate some or all of the remaining individuals to recover a population of C individuals, and perform some random mutations on the individuals.
6. Stop if the fittest individual meets the given requirement, or go back to step #2.

2 Creatures

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

- A creature always occupies a single tile of the environment.
- At each tick, a creature can move to any of the eight adjacent tiles, provided that it is not occupied by an obstacle. If after the move, there is no obstacle right below the creature (i.e. it is not resting on an obstacle), at the next tick, it will move with the same horizontal velocity (0, 1, or -1 tiles/tick) and a vertical velocity that is increased by -1 tiles/tick, simulating gravity. If after the move, it is resting on an obstacle, it chooses its next move. See Figure 2 for examples of moves.
- A creature's moves are determined by its internal program (this program can be viewed as its genes). This program is a sequence of T move directions.
- All creatures always starts at the same position in the grid.

- A creature's fitness is evaluated by letting it go through the T moves in its program. Note that this could take more than T ticks since some moves can take more than 1 tick, if the creature does not immediately reach a tile in which it is supported by an obstacle.
- Each grid contains a "goal tile" that the creature is trying to reach. After it has gone through all its moves, the distance from the creature to the goal tile is used as score (a lower score is thus a better outcome). Once a creature that reaches the goal has evolved, the algorithm stops.
- At each new generation, each of the T moves of a creature has an $m\%$ chance of mutating into a new move.

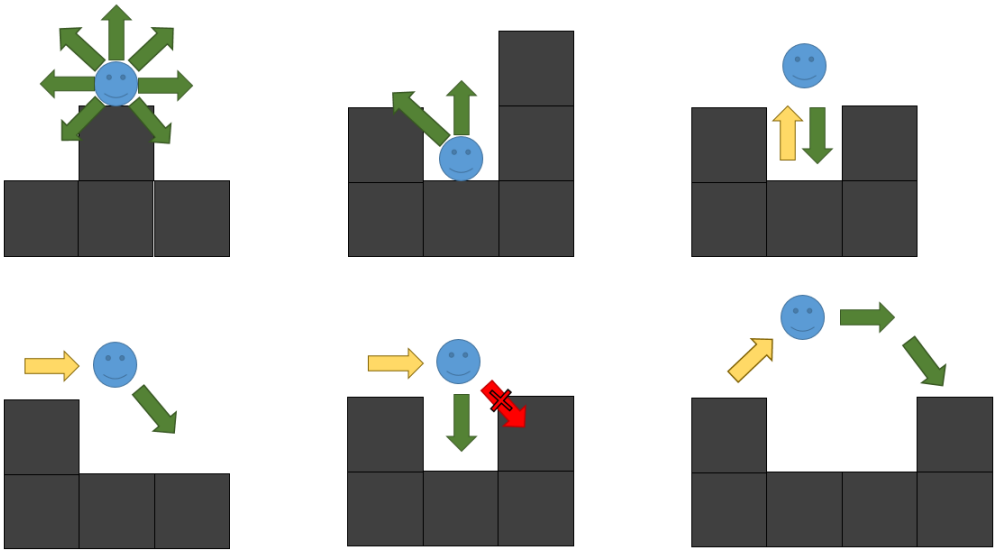

Figure 2: Possible moves and their outcome. Yellow arrow indicates the previous move.

3 Commands

Commands are taken from the console (keyboard) and are of three types. The user can:

1. Request the next generation (or create a random one the first time);
2. Request G additional generations in a row;
3. Display the score of the best creature so far, and visualize its performance, using 1 second as tick duration. This can be done at any time (i.e. even during the generation process).

4 Implementation

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

- The height $M \in [3, 5]$ and the width $N \in [5, 10]$ of the grid, the number $P \in [1, 10]$ of processes, the number $C \in [100, 1000]$ of creatures, the deletion percentage $p \in [20, 80]$, the mutation rate $m \in [1, 10]$, and the number $T \in [5, 20]$ of moves in a creature program should be parameters of your program.
- If provided, an optional 8th parameter is the location of a file containing the position of the starting tile, the goal tile, and the obstacles. Positions are represented by their (X,Y) coordinates and separated by a CRLF. For instance, a file containing "0 0\r\n3 1\r\n2 0\r\n3 0\r\n" indicates an environment where the creature starts at (0,0), must go to (3,1) and contains two obstacles, at (2,0) and (3,0). If this parameter is not provided, a grid of size (N, M) is generated at random.
- The grid is displayed on the console using characters to represent the grid and the creature.
- The sequence of moves (program) of the creatures and their score are stored in shared memory; a "master process" is used to handle the console inputs and outputs, as well as sorting, killing and breeding the creatures. All processes have access to the data stored in the 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 (generation request or best creature visualization) are transmitted to the "master process" using terminal I/O.
- Worker processes are used for testing and evaluating creatures. A creature is only evaluated once per generation, but the evaluation procedure can start even if the set of all C creatures for the current generation are not all created yet.
- Communication between the master process and the worker processes is handled 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 messages from the message queue).

5 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 on the console.
- You must write a report describing how you implemented your program, and in particular how the synchronization is performed.
- The project has to be done in teams of 2 students and completed before before December 19th 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**[1], after having created an account if necessary. If you encounter any problem with the platform, let me know (S.Hiard@ulg.ac.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...

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