Re-exam

DM857 Introduction to Programming
DS830 Introduction to Programming

Simulating a Population of Cells

During this project, you will implement a program to simulate a simple model of a population of unicellular microorganisms (cells for short) and its evolution over time under selective environmental pressure. The program progresses through three main stages: configuration, simulation, and reporting. Each stage is described below.

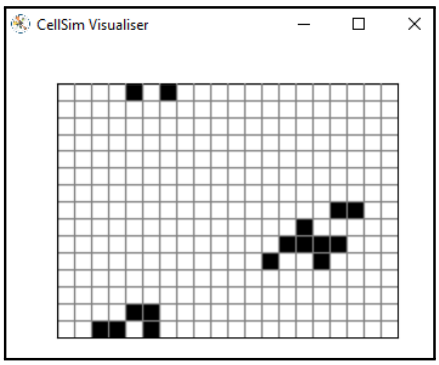

Figure 1: Visualization window.

Configuration

Running the simulation requires the following configuration parameters:

1. Grid dimensions (number of rows and columns each of them must be ≥ 3).
2. Initial cell population (≥ 1).
3. Age limit in ticks (≥ 1).
4. Division limit (number of successful divisions (≥ 1)).
5. Division success probability (between 0 and 1).
6. Division cooldown (minimum time between consecutive divisions of a cell (≥ 1)).
7. Simulation time limit, in ticks (≥ 1).
8. Whether to use a visual display of the simulation status (see the visualizer module) (True or False).

When the program starts, these parameters are set to their default values (listed in Table 1).

Then, the user can review and modify the configuration interactively, proceed to the simulation phase, or exit the program. The program must allow the user to adjust any parameters or run the simulation with the default settings.

Parameter	Default Value
Grid rows	15
Grid columns	25
Initial population	2
Age limit	10
Division limit	2
Division probability	0.2
Division cooldown	2
Time limit	100
Visualisation	Enabled

Table 1: Default simulation parameters.

Grid Configuration

For this simulation we will describe a rectangular space in which the cells can live. The space is divided into a grid of patches (elementary squares) with at least three rows and columns. Each patch can hold at most one cell. The space is toroidal, meaning that the rows and columns wrap around, similar to a snake game.

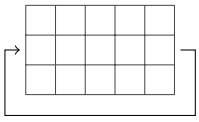

Figure 2: Geometrical setup of the space; each white square identifies a patch.

This geometry ensures that each patch in the grid has exactly eight adjacent patches (those reachable by moving one row and/or column). These adjacent patches form the patch's neighbourhood. The figure below illustrates a 5×5 grid, highlighting in light gray the neighbourhood of the patch in dark gray at (0,0).

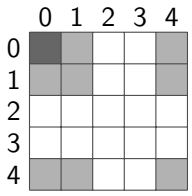

Figure 3: Example of neighbourhood (in gray) for the patch (in black) located at $(0, 0)$.

Each patch can contain at most one cell, and each cell occupies exactly one patch. Cells are initially placed randomly within the grid. The simulation proceeds in discrete time steps (ticks), updating the population by making cells age, divide, and die based on certain conditions.

- A cell can divide if:
  - At least one neighbouring patch is empty.

- It has not exceeded the division limit.
- The cooldown interval since the last division has passed.

If these conditions are met, the cell divides with a given probability, placing a single new cell (with full division counter) in a randomly selected empty neighbouring patch, while decreasing its division counter by one unit. The figure below illustrates this process.

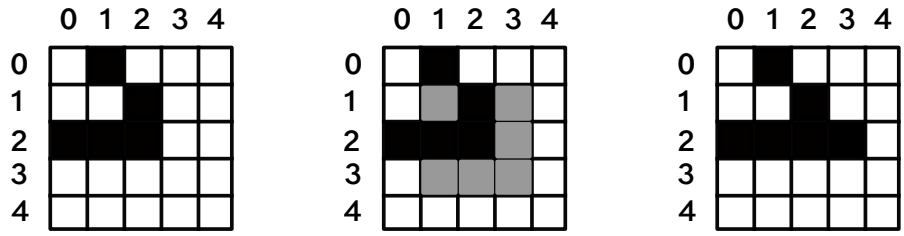

Figure 4: Example of the evolution of cell (2,2): The left panel shows its initial state, the center panel highlights potential neighboring cells for evolution in gray, and the right panel displays a randomly selected neighbor for evolution.

- When a cell reaches its age or division limit, it dies.
- If a cell finds no empty neighbouring patches, the oldest cell in the 3×3 region is randomly selected to die due to overcrowding.

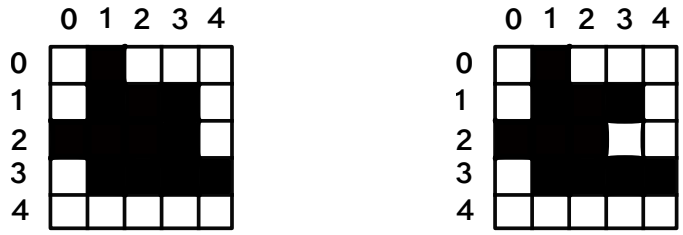

Figure 5: Example of death by overcrowding of cell (2,2): The left panel shows its initial state, and the right panel displays a randomly selected oldest cell that dies due to overcrowding.

- The simulation continues until the time limit is reached or all cells are dead.

The program requires several random operations. To implement these, use Python's built-in `random` module, particularly the functions `random`, `randint`, `choice`, `choices`, and `sample`.

1 Reporting

After the simulation ends, the program will display the following statistics before terminating:

1. Time covered by the simulation (in ticks).
2. Total number of cells created.
3. Total number of deaths.
4. Breakdown of causes of death (age limit, division limit, overcrowding).

A single cell may have multiple causes of death (e.g., reaching both age and division limits in the same tick).

2 Provided Material

You are provided with two modules: `models` and `visualizer`, along with documentation (available on itsLearning). The `models` module defines grid patches and cells, while the `visualizer` module enables graphical visualization.

The `models` module defines the `Patch` and `Cell` classes. Patches represent grid locations, while cells inhabit patches. Cells are created by invoking their constructor and assigning them to a patch via `put_cell`.

The `visualizer` module requires the `matplotlib` library. Upon instantiation, the `Visualizer` class requires a list of cells with defined coordinates. Calling `update` refreshes the figure, displaying the simulation's current state.

3 Hand-in

Files

Submit a zip file containing:

- A folder named `code` with your implementation, including a module named `cel_sim`.
- A PDF report titled

Setup

Ensure your submission follows these requirements:

- The main module must be named `cel_sim`.
- Do not modify the provided modules.
- Follow Python coding conventions and document your code.
- Only use standard Python libraries (`random`, `numpy`, `matplotlib`).
- The report must be in English, formatted for black-and-white printing, and limited to five pages (excluding the appendix).