University of Cape Town

Department of Computer Science

CSC1018F: Python Project

Week 1

Introduction

This year's project is going to be about Cellular Automata. For those of you who are unfamiliar with Cellular Automata, they are a simple way of modelling certain real world situations. Typically, they involve breaking your world up into a collection of cells in a grid. Cells are chosen to have different initial properties and at each step of the simulation rules are on enacted on each cell.

For instance, in the Langton's Ant Cellular Automata, all the cells are chosen to be either white or black and a single cell is chosen to contain a wandering ant. At each step of the simulation the ant moves, changes direction and flips the tile's colour depending on the colour of the tile it is currently on. This toy example has great importance to the Computer Science as it has can be used as a rudimentary way of computing. For those interested there are lots of videos on Langton's Ant on Youtube, the video at `http://www.youtube.com/watch?v=1X-gtr4pEBU` is pretty good. Cellular Automata have various other uses as well, for instance in simulating the growth of cancer in the body, or solving differential equations.

For the next three weeks you will be working on a small project. Your project will be to create a Cellular Automata called Lion's and Antelopes. The project will be broken up into three parts each being one week in duration. You will be required to submit your work at the end of each week so that it can be evaluated and feedback can be given. Do not get behind on the work, as this will only mean more work later and less marks being rewarded. Remember to code in a neat, modular fashion as each week's work will build upon your previous work.

The first week will require you to create the Cellular Automata system. This will involve implementing the basic rules for simulation. For the second, you will create a graphical interface for the Cellular Automata, so that it can be easily visualised. Finally, in the last week you will be required to modify your program so that it can save and load the output from the cellular automata into a file. So that it can be easily viewed later.

Lions and Antelopes

The cells in the Lions and Antelopes world can be one of three different types:

- Lions

Figure 1: Baby Lion

Figure 2: Adult Lion

  - Lions are predators that love to eat Antelope, in fact, that's all they eat. There are four rules that determine what happens to a lion cell for a given round of the simulation.
    1. Lions live for 20 generations, if a lion has lived in a cell for more than 20 generations it dies, leaving the square empty.
    2. Lions also have a random 1 in 32 chance of dying of natural causes, leaving the square empty.
    3. If a lion is surrounded by 6 or more lions and no antelope it dies of hunger, leaving the square empty.

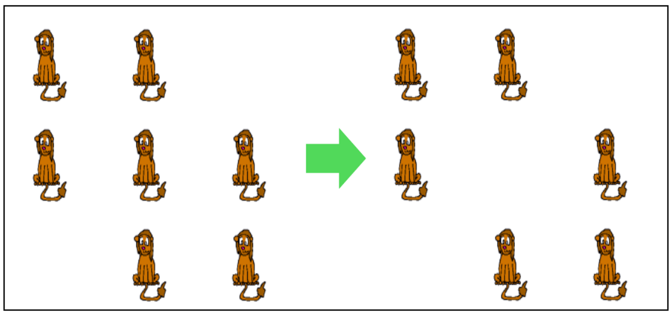
Figure 3: The lion is surrounded by 6 lions meaning that it dies and is replaced by an empty cell in the next generation.

    4. If the lion survives, its age is incremented.

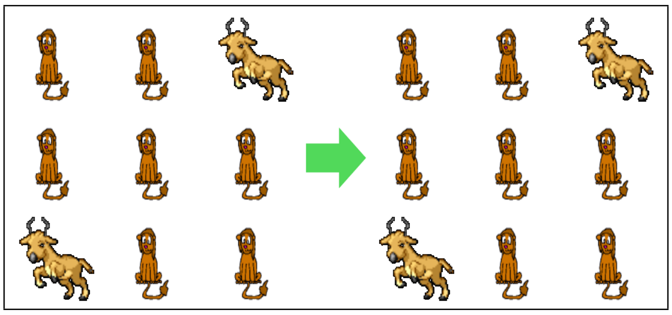

Figure 4: The lion in the center cell is not unlucky enough to die of natural causes, and although he is surrounded by 6 lions there are also 2 antelope. This means that the lion lives to the next generation.

- Antelopes

Figure 5: Baby Antelope

Figure 6: Adult Antelope

  - Antelopes are animals which exist to be eaten by the lions, they survive on grass which is plentiful where they live. There are four rules for cells that contains a fish.
    1. Antelope live for 10 generations, if an antelope has lived in a cell for more than 10 generations it dies, leaving the square empty.
    2. If the antelope is surrounded by 5 or more lions it gets eaten, leaving the cell empty.

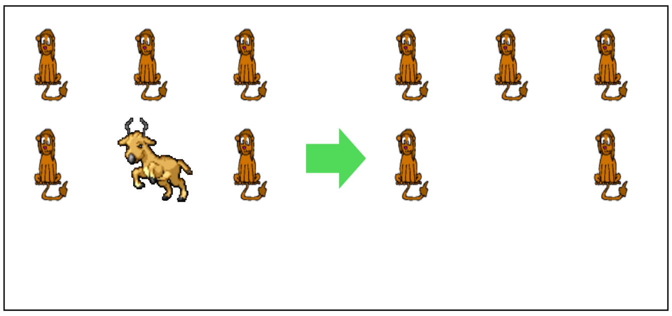

Figure 7: The antelope is surrounded by 5 lions and is eaten on the next turn, leaving the cell empty.

    3. If the antelope is surrounded by 8 other antelope it dies of starvation, leaving the cell empty.
    4. If the antelope survives, its ages is incremented.

- Empty

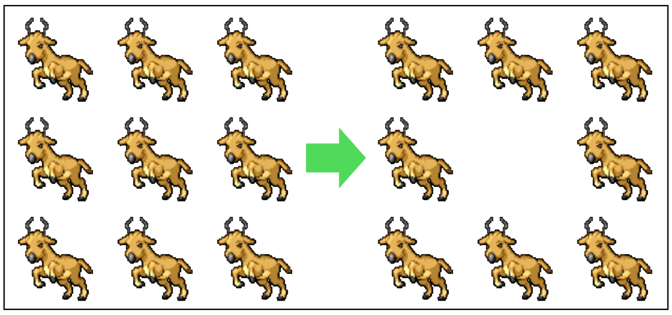

Figure 8: The antelope in the middle is completely surrounded by other antelope, according to the rules this antelope starves and is replaced by an empty cell in the next turn.

- Empty cells are cells which can be filled with either lions or antelopes if the correct conditions are met. There are two rules for empty cells which determine which animal is created at the cell.

  1. If there are at least 4 neighbouring lions **AND** at least 3 of those lions are at least 3 generations old, the breeding age for lions, **AND** there at most 3 neighbour antelopes then the cell is replaced with a newly born lion with an age of 1.

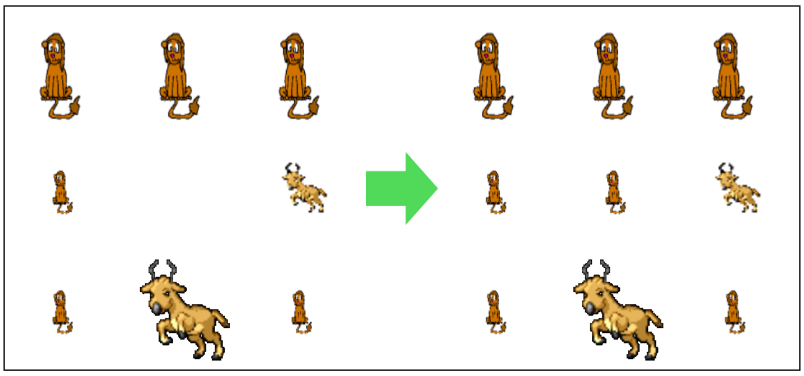

Figure 9: Here we can see a new lion being born. The empty cell in the center has become a new baby lion.

  2. If there are at least 4 neighbouring antelopes **AND** at least 3 of those antelopes are at least 2 generations old, the breeding age for antelopes **AND** there at most 3 neighbour lions then the cell is replaced with a newly born antelope with an age of 1.

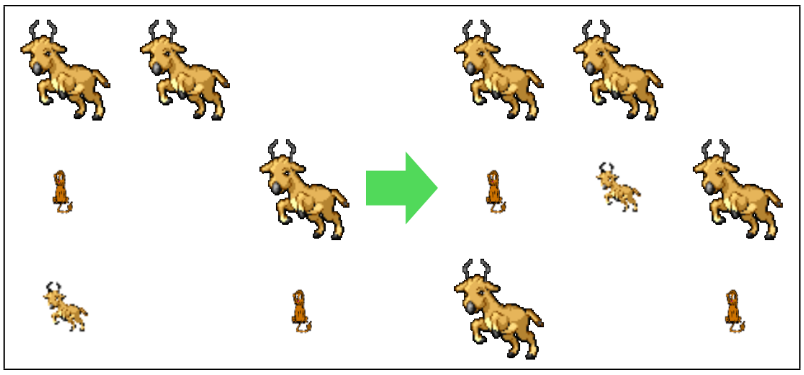

Figure 10: Here we can see a new antelope being born. The empty cell in the center has become a new baby antelope. In addition, the baby antelope in the bottom left cell has matured into breeding age.

Requirements

Write a program that will simulate the Lions and Antelope Cellular Automata. Your program must do following:

- Read in the width and height of the grid as arguments to the program. The program should also be able to take an optional number indicating the number of turns to simulate the Cellular Automata. If this number is not input then the program should run indefinitely.
- All of the rules should be implemented. You are allowed to experiment and add new rules, however, there should be a setting to run with only the original rules. Your program should be implemented efficiently, that is, it should be able to run at least 1 generation per second at a grid size of 100×100.
- Finally, there should be some output describing the state of the grid. This is vital for the tutor to mark the correctness of your work. Any form of text-based output for the grid will be sufficient. For example, outputting an E for an empty cell, an L for a cell containing and lion and an A for a cell containing an antelope.

Finally, it is important to write a short README for your tutor. In this you should detail the work you have done. This will make it easier for the tutor to mark your work. Any problems you encounter should be explained in there. If you know why these problems occured you should also say that. Any deviations from the project specification should be noted. Any extra ideas such as having a grid that wraps, or removing constants(such as the breeding ages) and replacing them with changeable parameters, will earn you extra credit.

Your tutor will also mark your code on neatness and usefulness of comments, etc. Make sure that your code is neat and well commented.

- Basic functionality: 40
  - These marks are given for the basic Cellular Automata. Have you implemented all the rules? Are they correct?
- Code Efficiency: 10
  - Have you used efficient methods? Have you favoured short, readable code over long, complicated code?

- Extra features: 20
  - What extra features have you implemented that are above and beyond the standard Cellular Automata.
- Documentation and Style: 30
  - Do you have proper comments in your code? Is your coding style consistent and neat? Do you have an adequate README?

Week 2

Using a terminal or console for your Cellular Automata and outputting in text is not an ideal way to visualise what is happening. This week we will require you to create a Graphical User Interface for your program.

The library that you will be using for the GUI is TkInter. TkInter lets you easily make interfaces within the python language. For those of you familiar with Java Swing or Java AWT, it is considerably different. There is no need for you to write action listeners, only functions that are called when buttons are pressed.

There will be a lecture given on how to use the TkInter library and a demonstration of the basic tools that you will need. The TkInter library is able to do various things such as image loading and transforming so there is no need to use additional libraries in this respect.

You can find documentation about TkInter at `http://wiki.python.org/moin/TkInter`, however, if you are interested in using a different library for the GUI you should tell your tutor or me first.

Your GUI should be able to continuously display the state of the Cellular Automata. At the least there should be basic controls that allow you to pause, stop and restart the Cellular Automata. Controls for changing each of the parameters, such as the size and breeding ages, to the Cellular Automata are also vital and marks are awarded for them.

Marks will be awarded to a neat and functional interface. Next week's project will require you to add at least two buttons to your form, so you may want to leave room on your GUI for this.

There will also be a portion of the marks awarded to displaying statistics about the simulation. Your GUI should at the least display the number of living lions and antelope. This will only get you part of the mark and you should add additional statistics to your GUI to get the remainder.

Requirements

Extend your program to add a GUI to your Cellular Automata. Your program must do following:

- Continuously display the state of the Cellular Automata via the GUI.
- Have controls to play, pause and restart the Cellular Automata.
- Display statistics about the Cellular Automata via the GUI.

Again, it is important to write a short README for your tutor. You need to document what works and what doesn't. You should also explain what extra work was done, so that the tutor can award marks for these. Your tutor will also mark your code for neatness and appropriate comments.

- Basic functionality: 60
  - Of which, 30 marks will be awarded for the displaying of the Cellular Automata. 15 marks will be awarded for basic controls and 15 marks will be awarded for basic statistics.
- Extra features: 20
  - These marks will be awarded for implementing extra controls and functionality to your interface and displaying more statistics.
- Documentation and Style: 20
  - Do you have proper comments in your code? Is your coding style consistent and neat? Do you have an adequate README?

Week 3

Cellular Automata simulations can take a long time to run as you may have found out from the previous two weeks of work. It may not be possible to watch the entire thing from start to finish. Often these simulations are recorded to a file so that they can be viewed later. For this week you will be adding recording and replaying features to your Cellular Automata so that it can be viewed later.

Your program needs to be able to write out and read in log files. Your GUI should have buttons to specify which log file to read from or which log file to write to. When reading from log files it is not necessary to run the simulation, as all the information required to get the next frame should be stored in the log file. Your GUI should provide the ability to seek to arbitrary positions in the file.

You will need to design the layout of the file format carefully so that you don't store unnecessary information and that all the information necessary to replay the original is kept. The file formats for simulation log files generally broken up into the two sections: The file header and the frame information.

- File header
  - The file header contains information about the simulation that doesn't change from frame to frame. Often these are details such as the total number of frames and the size of the grid.
- Frame information
  - The frame information section contains each frame of the simulation. The information for each frame should be logically separated in this section.

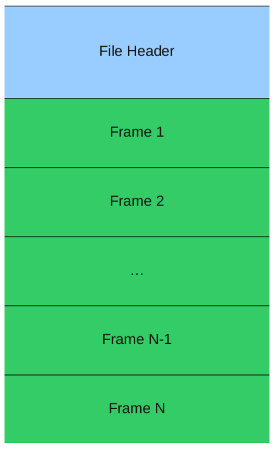

Figure 11: A graphical example of a standard file format.

These simulations can grow to be very large. A log file for a standard simulation with 100x100 grid simulated for 10000 frames can be as large as 100MB in size. For this reason, you will need to implement compression of the log files. As the majority of the file size is made up from the frames we want you to compress each individual frame in the log file.

You should not use any libraries for compression and you should not use any external utilities. The code for the compression should be entirely written be yourself. Remember that you need to have the ability to seek in the log files so the method you use for compression should not negatively impact this.

An idea to start off with is simple run-length compression of each frame. Run-length compression would encode consecutive cells with the same content as one number, representing the number of consecutive cells, and the contents of the first cell. While this may help you decrease the file size by a small amount, more complicated schemes will need to be used to achieve the full mark. You will be marked on the complexity and overall performance of your scheme.

Requirements

Add the ability to generate and review log files from your program. The program must do the following:

- Have controls to save and load compressed log files.
- Be able to seek to arbitrary locations with in the simulation.

A README must be submitted for your tutor. In it you must describe your file format. The file must also describe your method of compression. Be sure to detail any tests that you performed to ensure that your compression method works. You need to document what works and what doesn't so that your tutor can award part marks for things that are broken. Your tutor will also mark your code for neatness and documentation.

- Basic functionality: 50
  - These marks will be awarded to programs which fulfill the basic requirements. That is loading and saving of log files, arbitrary seeking and basic compression.
- Extra features: 25
  - These marks will be awarded for using a significantly better compression method than simple run-length compression.
- Documentation and Style: 25
  - Do you have proper comments in your code? Is your coding style consistent and neat? Do you have an adequate README? This includes 10 marks awarded to the description of your file format and compression.