Project 1 Checkpoint 1

Group 59

I. Project Description

For this project, we will implement a 2D snake game. The general idea is that a user controls a snake via keyboard to move around a 2D map and try to eat as many randomly appearing dot/food as possible. The rule is the same as that of the standard snake game we all played before. The map will be a 2D cell grid and the snake will be connected line segment(s). Below is a picture of what this system looks like:

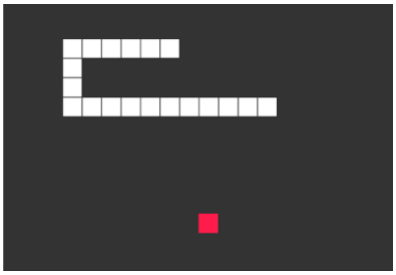

Figure 1. Visualization of Snake Game

Our goal of the project is broken into two main parts. For the first part, we will implement this simplest version of the snake game. Specifically, there's a real-time display window for the game. Then, the snake should move correctly based on its own dynamics and external keyboard inputs. The snake grows length after eating a block and dies if it hits any boundary or itself. For the second part of our goal, we will add additional simulated snakes into the game. These 'computer' snakes move according to some preprogrammed patterns. Now the user's snake can either eat a 'computer' snake or die from hitting it based on some rules. If we still have time after reaching this goal, we will attempt additional functionalities for the game.

The main course concept behind this project is cellular automata (CA). Essentially, the entire game can be represented by the 2D cell grid map. This 2D cell array is a fairly complex cellular automata system. Each cell is a pixel in the real-time display window. A cell has several states and continuously updates those state values at each time step. The states update rule for each cell is a core component of the CA system. The real-time display window just continuously displays each cell by evaluating its states at each time step, thus creating the game animation.

To complete this project, we expect to take the following approaches. First of all, we will set up a proper data structure scheme to represent the 2D cell grid game map. We need to figure out how to store all the information associated with each cell. Second, we will need to implement codes that transfer keyboard inputs to the main function flow in real time. Third, we will design algorithms to correctly update the cell state values. This will be essential to the CA system dynamics. Finally, we will utilize graphing tools to visualize the dynamic 2D cell grid. Precisely, what should be displayed at each cell based on its current state values.

The programming language we plan to use is Python, and the platform is very likely to be PyCharm. After looking at several online resources, we are sure that this project is achievable by Python. In addition, for the final tutorial format document, we can directly convert our Python codes to a Jupyter Notebook file.

Finally, listed below are some online resources/literature we found useful for our project:

1. https://natureofcode.com/book/chapter-7-cellular-automata/
2. http://grammars.grlmc.com/LATA2008/slides/lata2008_cellular_automata.pdf

II. Current Progress

First of all, we have to point out that this project is completely different from the idea we wrote in our proposal. Earlier, we planned to implement a soccer match prediction model using Poission distribution and data analytics. However, we couldn't find a convenient way to connect it with any course concept we have learned so far. Therefore, we decided to choose a new project that's more related to the course topics. This is why we choose this new project now. We believe it's a great change and we are all very excited about this new project.

Now, we will talk about our current work status. We haven't written much official project code yet, but we used codes to test and verify our analysis work. For the representation of the 2D cell grid (game map), we chose to use a 2D array with integer element type. The integer value of a cell encodes this cell's state. So far, we set 3 possible state values for each cell: 0 means empty, 1 means part of the snake, 2 means location of 'food' block. Any cell is always in one of these 3 states, i.e. having value between 0 and 2. For the real-time visualization window, we will use some Python graphing library functions to dynamically display the 2D integer array. For each cell, if it's empty, we display it as black. If it's part of the snake, we display it as white. If it's the 'food' block, we display it as red.

For the characterization of system dynamics of the cell array, we explored many different options. Our goal is to find an efficient way to update the state value for each cell, based on different internal and external factors. So far, we think it's a great idea to represent the snake by a linked list. Each node in the list order represents one block of the snake, sequentially from head to tail, with the head of list being also the head of the snake. Each node is an object class containing the x and y coordinate of this snake body part on the 2D cell grid, while the head node includes additional information of the snake's current moving direction. Keyboard inputs will be transferred via a Python API to the main function loop. The head node's moving direction field will be updated based on any keyboard input. Finally, it comes to the algorithm for updating the cells. At each time step, we replace a cell's x-y coordinates by those of its predecessor. For the head node, we update its coordinates based on the current snake moving direction. This way, we effectively make the snake move by one unit of time. Moreover, we need to check if the snake head hits a 'food' block (the coordinates of this block are stored as a global variable that can be accessed at any time). If so, we create a new node with the 'food'

coordinates values and make it the new head of the snake. A new 'food' block will be randomly generated on the grid. Finally, at each time step we will also check for any game-over condition using each node's coordinates information. After all these snake status updates, the new 2D cell array values at the next time step will simply be set to be in accordance with the position of the snake and a potentially new 'food' block. This completes one cycle of CA system updates.

The thinking above is for a single snake controlled by the user. For additional computer simulated snakes, the snake status updating process is the same. We just need to program a movement pattern and feed it to the head node of the list representing this snake. The intelligence level of the movements can vary based on our design. This snake can randomly move around the map, or randomly move and avoid dying, or actively search for 'food'. This completes our current working progress. Our next immediate step is to start writing Python codes.

Finally, we will talk about division of labor. The interaction between the main function flow and external keyboard inputs is one module of the project. The real-time visualization of the game animation is another module. The linked list data structure creations and update algorithms is a module. The game-over condition check, random 'food' block generation, as well as the 2D cell array updates based on the snake and 'food' locations is another module. Finally, the generation of additional 'computer' snakes and their moving pattern designs is yet another module. We will evenly distribute these modules among us. It's obvious that these modules are closely related to one another to create the whole AC system. Therefore, we will definitely collaborate with each other as we work towards our final goal.

III. Git Repository for Final Submission

This is the link to our GT GitHub repository for final submission:
https://github.gatech.edu/tzheng34/CSE6730-Group-59-Project1