COMP 426 Assignment #1 Report

Name: Mohamed Hossein Lakkis
ID: 40089703

Date: 30-09-2025

**Overview**

This project implements a multi-species game of life on a 2D grid using multithreading techniques to leverage multicore processors. The project also uses OpenGL to display the simulation of the game graphically. The simulation includes:

- A grid of cells, either alive and belonging to one of three species, or dead.
- Each species evolves independently according to the following rules:
  - A live cell with less than two neighbors dies (underpopulation)
  - A live cell with 2-3 neighbors survives
  - A live cell with more than three neighbours dies (overpopulation)
  - Dead cells with exactly three neighbors become alive.
- The grid updates happen 30 times per second

The project is programmed using python and using the PyOpenGL + GLUT libraries for graphics, and the **threading** library for multithreaded implementation.

**Code**

Imports:

- Random -> for random initial population
- Deepcopy -> ensures updates don't overwrite the current grid
- pyOpenGL
- threading.Thread -> parallel computation per species

parameters:

- WIDTH, HEIGHT -> simulation grid dimensions
- CELL_SIZE -> size of a cell in pixels
- SPECIES -> list of species names
- FPS -> updates per second
- COLOR_MAP -> RGB color mapping for each color (species)

Grid initialization:

- This line of code creates a grid using the HEIGHT x WIDTH parameters.
- Each cell is randomly initialized as either dead '.' Or alive for a species
- This way, random distribution of species is ensured.

Neighbor Counting function

- The function *count_neighbors* takes in three parameters (x, y, species)
- Species is given as a parameter to ensure that the evolution is species-specific
- The 8 surrounding cells are checked by looping over a vertical offset (-1,0,1) three times over three horizontal offsets and skipping checking the cell itself.
- Modulo is used to ensure grid wrapping is applied.
- The neighbor's position Is then computed and if the neighbor's species matches the one of the current cell, the count is incremented.
- Count is returned.

Species update function

- The function update_species goal is to compute the next generation for a single species without affecting other species.
- It loops through every row of the grid, and through every column
- It counts how many alive neighbors are alive around the current cell (x,y)
- It checks if the current cell is alive or dead
- It applies the rules of the game of life and updates the new grid

Update grid function

- The function *update_grid* computes the next generation for all species. It launches a separate thread for each species which allows parallel computation
- A new copy of the current grid is made using *deepcopy*.
- All thread read from the original grid "grid" and write to the new grid "new_grid"
- Threads = [] holds all the thread objects
- Loop over all species in the simulation
- Each loop creates a new thread "t" for each species
- Thread(target = update_species) makes the new thread call the function update_species and passes the new_grid and the species as parameters.
- The new thread starts executing.
- Wait until all threads are updated

- Replace old grid with new grid

Timer function

- The function *timer* acts as the control thread in this program.
- It triggers the creation and execution of computation threads by calling update_species on each time tick
- glutPostRedisplay() triggers the display callback, which re-draws the grid to the current state.
- glutTimerFunc(int(1000/FPS), timer, 0) schedules the timer function to run after ~33 milliseconds (30 FPS).

main function

- this function starts by initializing GLUT with command-line arguments
- the correct configuration of the GLUT window size is set.
- Opens the window and sets background color to black
- Sets up 2D orthographic projection so the grid can be drawn in pixel coordinates
- Registers the display callback
- Starts the timer function
- Enters GLUT main loop, which is the control thread

**Summary**

The implemented Python OpenGL program simulates a multi-species Game of Life on a 2D grid, where each species evolves independently. The computation for each species is performed in a separate thread, allowing parallel updates and demonstrating multithreading in action. A main control thread, implemented via the OpenGL GLUT main loop and timer function, orchestrates the simulation by triggering the species computation threads and updating the display at a consistent 30 frames per second. The grid is rendered using colored quads in OpenGL, providing a smooth visual representation of all species simultaneously. This design satisfies the assignment requirements by combining multithreaded computation, per-species independence, and continuous, efficient screen updates.