Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A21

Computer Science Challenge

Lab Professor / Lab Session:

[Daniel Cormier / 302]

Team:

[Vijayalakshmi Krishnaraj] - Id: [041026347] - [Priya Patel] - Id: [041084491]

CS Challenge 2: Game of Life

|  |  |
| --- | --- |
| **Part**  **1** | **Implementing GL** |

**Note 1: Read Specification**

*This is only a suggested template. Please check instructions in the A21 specification.*

* 1. **Example UC Model (2pt)**

🛠 Repeat the ideas already defined in the A11: The UC diagram and the UC table (actors and functionalities).

**UC Diagram** (change this diagram to accommodate the actors and functionalities to be used):

A diagram of a person with text

Description automatically generated

**Actors table** (example):

|  |  |
| --- | --- |
| **Actors** |  |
| Player | Represents the only one person interacting with the Game of Life game. |
| Computer | Shows the GL grid |

**UC table** (example):

|  |  |
| --- | --- |
| **Use Cases** |  |
| Game | Provides basic information about the game. |
| Language | Player selects a language for internationalization. |
| Help | Player views information about the game. |
| Random | The computer randomly generates the binary numbers. |
| Manual | The player enters the manually enter the binary numbers. |
| Model (Input) | Player provides an 18-bit binary sequence as input |
| Multicolor | To create random color |
| Steps | Player advances the execution by one step. |
| Start | Player starts the execution of the Game of Life. |
| Stop | Player stops the execution of the Game of Life. |
| GL grid | This is the central area where the grid of cells is displayed and evolves over time.  It's a grid of cells, each of which is either "alive" or "dead" and evolves according to the rules of the Game of Life. |

* 1. **Example CD Solution (2pt)**

🛠 To draw the diagram, you can use tools (ex: <https://app.diagrams.net/>) or desktop applications (ex: Visio / Powerpoint) or simply take photos from drawings.

**Class Diagram** (change this diagram to accommodate the actors and functionalities to be used):

A screenshot of a computer

Description automatically generated

**Class table** (example):

|  |  |
| --- | --- |
| **GameOfLife** | **Order** |
| Inner Fields[[1]](#footnote-1) | **board**: A 2D array of Cell objects representing the game grid.  **generation**: An integer representing the current generation of the game. |
| Relationships[[2]](#footnote-2) | The GameOfLife class has a **composition** relationship with the Cell class, as it contains a 2D array of Cell objects to represent the game grid. This means that the existence of Cell objects is closely tied to the existence of a GameOfLife instance. |
| Methods | **GameOfLife(rows: int, cols: int):** Constructor for initializing the game with the specified number of rows and columns.  **initializeBoard(binaryConfig: String):** Initializes the game board based on a binary configuration.  **evolve():** Advances the game to the next generation.  **isCellAlive(row: int, col: int): boolean:** Checks if a specific cell is alive.  **getGeneration():** int: Returns the current generation. |

|  |  |
| --- | --- |
| **Cell** | **Order** |
| Inner Fields[[3]](#footnote-3) | **alive**: A boolean indicating whether the cell is alive or dead. |
| Relationships[[4]](#footnote-4) | The Cell class is part of the GameOfLife class, forming a **composition** relationship. The Cell objects are components of the game grid maintained by GameOfLife. |
| Methods | **Cell():** Constructor for creating a cell with an initial state.  **isAlive():** boolean: Checks if the cell is alive.  **setAlive(alive: boolean):** Sets the state of the cell (alive or dead).  **toggleState():** Toggles the state of the cell between alive and dead.. |

|  |  |
| --- | --- |
| **GameOfLifeGUI** | **Order** |
| Inner Fields[[5]](#footnote-5) | **numRows**: An integer representing the number of rows in the game grid.  **numCols**: An integer representing the number of columns in the game grid.  **cellSize**: An integer representing the size of each cell in pixels.  **grid**: A 2D boolean array representing the state of the game grid.  **timer**: A Swing Timer for controlling game updates.  **startButton**: A Swing JButton for starting the simulation.  **stopButton**: A Swing JButton for stopping the simulation.  **configField**: A Swing JTextField for entering binary configurations. |
| Relationships[[6]](#footnote-6) | GameOfLifeGUI has **associations** with JButton, JTextField, and Timer, indicating that it uses these Swing components for user interaction and timing. |
| Methods | **GameOfLifeGUI**(): Constructor for setting up the GUI and initializing the game.  **initializeGridRandomly():** Initializes the game grid with random cell states.  **parseBinaryConfiguration(input: String):** Parses a binary configuration string and updates the game grid accordingly.  **countLiveNeighbors(x: int, y: int): int**: Counts the number of live neighbors for a given cell.  **actionPerformed(e: ActionEvent):** Handles button clicks and timer events to control the simulation.  **paint(Graphics g):** Overrides the paint method to draw the game grid on the GUI. |

*Create tables for all classes.*

**Details**

*Drawn the Class diagram (ex: in an image from Paint / Visio / Powerpoint slide, or any sketch tool), describing:*

* *Class definition (properties / methods).*
* *Relationships between classes.*
  1. **Visual Components (1pt)**

**GL Implementation**

**GL Window:** This is the main window where the player can see the simulation in action. It displays the grid of cells and shows how they evolve over time.

* + - * Title (showing image). Game of Life
      * Menus: To select some options such as: These allow the player to access various options like Internationalization, information about the game, and help.
        + Internationalization: Showing at least two items (English and another).

Allows the player to change the language of the interface.

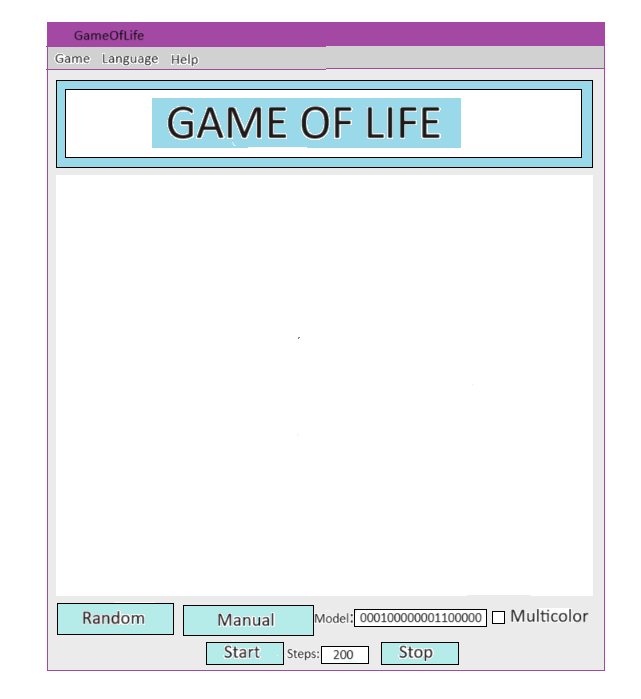
* + - * + About: Basic info about the game.

Provides basic information about the game.

* + - * Area to show the GL grid: This is the central area where the grid of cells is displayed and evolves over time.
      * Input (given by **binary numbers** – 18bits sequence): This is input area where the player can provide an 18-bit binary sequence to influence the initial configuration.
      * Multicolor options: This button to create random colors for the cells on the grid.
      * Execution options: Start / Stop and step execution: Buttons that control the execution of the simulation. “Start” begins the simulation, “Stop” halts it, and “Step” advances it one step at a time.

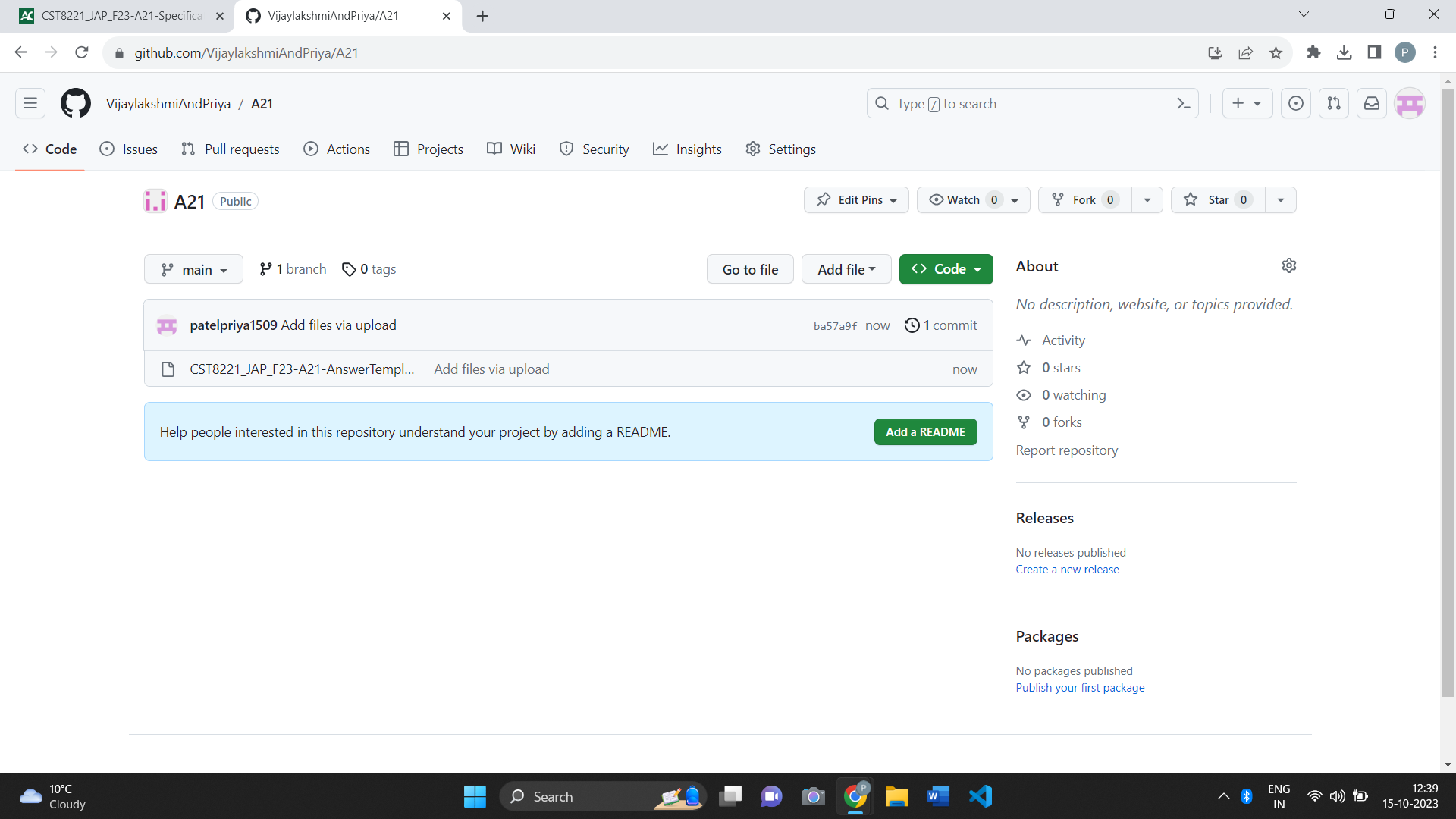
GL Grid (Game of Life Grid): This is where the actual cellular automaton is displayed. It's a grid of cells, each of which is either "alive" or "dead" and evolves according to the rules of the Game of Life.

The cells may be displayed using the selected colors, and they will change state (alive or dead) as the simulation progresses.

**

**FINAL SUGGESTIONS**

GitHub: <https://github.com/VijaylakshmiAndPriya/A21>



**References**

<https://www.jetbrains.com/help/idea/class-diagram.html>

<https://www.geeksforgeeks.org/program-for-conways-game-of-life/>

Algonquin College

Fall, 2023

1. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-1)
2. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-2)
3. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-3)
4. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-4)
5. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-5)
6. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-6)