1 Overview (Joel)

1.1 Project summary

1.1.1 Purpose, scope, and objectives

Provide the academic researchers, students, or teachers a simple and easy tool to perform rapid experiments with cellular automata and how emergence can arise from setting simple interaction rules and initial conditions. However not everyone who might be interested in CAs will have the time or technical prowess to write programs to run CA simulations. Our aim is to provide a GUI tool to minimise the latency between ideas and tangible results for professors, researchers, teachers, students, or hobbyists interested in the realm of cellular automata.

1.1.2 Assumptions and constraints

- Meet final product deadline: April 4, 2025.
- Application must be stable and able to run without crashing unexpectedly.
- The primary client is the course instructor.
- No budget for additional spending.
- Clients have familiarity with general computer use, but not necessarily programming or advanced computer skills.
- The first release of the project must be open for later features and further development to be added.
- Clients should be able to understand from the provided documentation what cellular automata are and how to create them within the application.

1.1.3 Project deliverables

- Full complete application available on Mac, Linux, and Windows
    - GUI interface for setting 1D and 2D rules along with initial conditions.
    - Interface section for starting, stopping, pausing, and stepping through the simulation.
    - Exporting and importing simulations to provide ease of sharing results.
- User manual giving a brief outline of what CAs are and examples of how they can be created within the application.
- Example simulations that can be imported into the project
- The complete project including the above elements will be delivered within 9 weeks.

1.1.4 Schedule and budget summary

No specific financial allocations have been made for this project. All work must be completed by the 12 members of the group assigned to designing and developing the product.

- Requirements
- Software Management Plan
- Object Oriented Analysis
- Object Oriented Design
- Project Implementation
- Workflow testing and Revision

1.2 Evolution of the project management plan

Changes to the project direction are inevitable and sometimes major. The following outline details procedures for addressing and implementing various changes.

1. **Requirements changes**
    a. Update the requirements document.
    b. Highlight the changes that have been made.
    c. Notify the group that changes have been made to the requirements.
2. **Updating project plans**
    a. Add plan revisions to the software project management plan (SPMP) flagged as a proposal.
    b. Notify the group members of the update to the plans.
    c. Once the proposal is approved by the relevant group members remove the proposal flag and accept it as part of the project
3. **Change in standards**
    a. Update relevant standards docs.
    b. Conform all existing work to match the new standard.
    c. Notify the group of the changes.
4. **Code changes**
    a. Create a new git feature branch from the main branch.
    b. Implement the required changes.
    c. Test, refine, and verify the code.
    d. Write appropriate documentation for the code changes.
    e. Merge the changes from the feature branch back into main with a descriptive commit message.
5. **Minor changes**
    a. Make the required change.
    b. When committing back into the git repo, provide a descriptive commit message.

2 Reference materials (Devarth)

All artifacts will conform to the course's programming, documentation, and testing standards.

3 Definitions and acronyms (Devarth)

3.1 Definitions

Cellular Automata (CA) : A mathematical model used to simulate complex systems through a grid of cells, each of which evolves based on predefined rules and the states of neighboring cells.
Grid: A structured arrangement of cells in either one-dimensional (1D) or two-dimensional (2D) space, used to represent the state of the cellular automata simulation.
Simulation Speed: The speed at which the cellular automata simulation progresses, controlled by the user through a slider or manual input.
Display Format: The visual representation of cell states, black and white squares or numerical values (0 and 1).
Preset Rules: Predefined configurations or rule sets (e.g., Conway's Game of Life) that users can select to quickly start a simulation.
Export/Import Grid: The ability to save the current state of the grid to a file and reload it later for continued simulation or sharing.
Pattern Recognition: Identifying common structures or behaviors in the cellular automata simulation, such as gliders, still lifes, or oscillators.
Configuration Management: Controlling changes to the project, ensuring consistency and traceability.
Quality Assurance (QA): Ensuring that the software meets specified requirements and quality standards.

3.2 Acronyms

CA: Cellular Automata
1D: One-Dimensional
2D: Two-Dimensional
SRS: Software Requirements Specification
SPMP: Software Project Management Plan
QA: Quality Assurance
JSON: JavaScript Object Notation (a file format for exporting/importing grid states)
CSV: Comma-Separated Values (a file format for exporting/importing grid states)
CASE: Computer-Aided Software Engineering (tools used for development)

4 Project organization (Abi)

4.1 External interfaces

All members of the group will participate in each phase of the project design and development. Updated artifacts will be provided to the course instructor for feedback and reviewed by the team following the submission deadlines.

4.2 Internal structure

The group members are Suleman Ali, Amelie Chu Moy Sang, Gurbir Dhaliwal, Ayush Gogne, Abigail Grimoldby, Zenil Riteshbhai Lakhani, Jonas Mogilnicki, Kunj Patel, Zulqarnayeen Sadid, Devarth Trivedi, John Uribe, and Joel Weber.

4.3 Roles and responsibilities

Roles and responsibilities will be assigned with each phase of the project, allowing for the work to be evenly distributed among the group. During the requirements phase, Ayush, Jonas, Joel and Zenil developed the description of the project. Gurbir, Devarth, Abigail, Zulqarnayeen derived the project's use-cases. Kunj, Suleman, John, Amelie constructed the GUI prototype. While developing the Software Management Plan, all twelve members of the group worked on the documentation simultaneously. It is essential for the roles and responsibilities of the individual members to remain flexible, so that tasks for the individual phases can be distributed as necessary.

5 Managerial process plans (Zulqarnayeen Sadid, Zenil Lakhani, Gurbir Dhaliwal, Ayush Gogne, Devarth)

5.1 Start-up plan

5.1.1 Estimation plan

The total development will be completed over a 10 week period from the first project requirement date being Jan 31st, and the final submission date being April 4th. The remaining submission deadlines are the SPMP on February 7th, Object-oriented analysis on March 7th, and Object-oriented design on March 21st. The total cost of this product will be $0. This is a pre-determined figure as there is no allocated budget for this project.

5.1.2 Staffing plan

All 12 group members will be required for all 10 weeks, being assigned their respective tasks for each assignment. During the first week, Ayush, Jonas, Joel and Zenil were needed for the description of the project. Gurbir, Devarth, Abigail, Zulqarnayeen were needed for the project's use-cases. Kunj, Suleman, John, Amelie were needed for the initial rough GUI of the project. During week 2, all 12 members are needed to develop the SPMP where members will work on different aspects of the IEEE Standard 1058. The remaining 8 weeks will further focus on the project's programming where all 12 members will be needed and expected to contribute their own code. Tasks for the last 8 weeks will be evenly distributed amongst the group members.

5.1.3 Resource acquisition plan

All resources required for this project are readily available to the 12 group members. Each member has access to the software that will be used to develop the product. No external administrative services will be required for this project.

5.1.4 Project staff training plan

No additional training is required as team members are proficient in Java and cellular automata concepts.

5.2 Work plan (Ayush Gogne)

5.2.1 Work activities

The work activities for developing the application starts first with research and analysis, this involves studying Conway's game of life and wolframs 1D cellular automata, analyzing key application concepts such as chaos theory, along with identifying functional and nonfunctional requirements. The second activity would involve designing the system and relationships between components. This includes aspects such as planning data structures for storing grid states, rules, and evolution history. The third step would be implementation, this involves building the main program that updates the grid by following the rules for how cells live, die, or change, building a graphical user interface so users can see and interact with the simulation, adding customization options such as the grid speed, display type (ex. Black and white or 1's or 0's) etc. The fourth work activity would be testing and debugging. This involves testing each part separately (ex. Cell updates, grid behaviour), checking how the app performs when all components come together, and making sure the user interface is easy to use.

5.2.2 Schedule allocation

The development of our cellular automata application follows a structured workflow, where each phase depends on the completion of the previous one. The main dependencies are as shown below:

1. **Research & Analysis:** Must be completed before system design can begin. Understanding cellular automata, understanding of relevant terms used in the project, and system requirements allows for an informed and quality design to be made
2. **System Design:** Follows research and defines the data structures and framework. This phase must be completed before coding starts, as it provides a blueprint for the implementation. Everything must be well thought out before the application is coded to avoid unnecessary coding and rework, ensuring a smoother development process.

3. **Implementation:** Depends on the design phase and involves developing aspects like the user interface which can only be done once the requirements and design is layed out so we know what we are creating and why.
4. **Testing and Debugging:** Happens after the implementation to ensure that all features work correctly, the application runs efficiently, and the user interface is functional. Issues found during testing may require updates to the implementation.

5.2.3 Resource allocation

Developers will handle coding, while testers perform unit and stress testing. Development tools and libraries will be used for coding and visualization. High performance computers and storage will be needed for simulations.Time will be allocated to each phase, ensuring that all tasks are completed efficiently and on schedule.

5.2.4 Budget allocation

Given that there will be no budget for the project, it will not be necessary to implement a budget allocation.

5.3 Control plan

5.3.1 Requirements control plan

Requirements are pre-determined and are not subject to change. However, any case where changes to the requirements are made will be discussed during weekly meetings. These changes must be approved by the whole group in order to officiate these new requirements.

5.3.2 Schedule control plan (Devarth)

Meetings will be held with the development team to review progress, identify roadblocks, and ensure alignment with the project schedule. Milestones will be defined and progress toward these milestones will be reviewed weekly. Key milestones are defined in the project schedule, such as completion of requirements analysis, design, implementation, testing, and delivery. Any delays will be reported immediately, actions will be taken to realign the project with the schedule.

5.3.3 Budget control plan

Given that there will be no budget for the project, it will not be necessary to implement a budget control plan.

5.3.4 Quality control plan

Team members will review each other's work, ensuring they are course standards and follow guidelines provided. This peer-review approach will help identify defects early in the development process.
Testing Strategy:

- Peer Testing: Each team member tests another team member's work products to ensure objectivity and thoroughness.
- For example:
    - Developer A tests the code written by Developer B.
    - Tester A reviews the test cases written by Tester B.

High-priority defects addressed immediately and lower-priority defects scheduled for future releases. Periodic quality audits are conducted to assess compliance with quality standards and processes. These audits cover code quality, test coverage, defect management, and adherence to the project management plan.

5.3.5 Reporting plan

A reporting plan describes the process to record project advancement as well as tracking performance metrics and challenges. The plans enable clear communication and responsibility management.

- **Progress Reports** : These will provide updates on the current status of the project,including completed tasks, ongoing activities, and upcoming works.
- **Performance Report** : These will include key performance indicators such as simulation accuracy, processing speed, and resource utilization, along with an analysis of performance trends over time to identify any patterns or issues.
- **Issue Report :** They will provide systematic issue tracking of the technical challenges faced during implementation, performance bottlenecks identified during testing, and user interface responsiveness problems.

All reports are maintained in a centralized google doc system accessible to all team members, enabling effective tracking of the project and facilitating timely interventions when needed.

5.3.6 Metrics collection plan

1. **Size:** LOC (lines of code) determines the number of lines of code being completed, and will be monitored per week to determine the speed of project development.
2. **Effort:** Group members will be assigned a rating from 0-5 to determine effort contributed per week.
3. **Quality:** Number of faults per 100 lines of code can be used to help us determine the reliability of a member's work.

5.4 Risk management plan

Cellular Automata simulators exist already. Therefore in order to look for potential faults in the product output, this product will compare test results to that of other currently existing simulators and get approved by the group. The product output will be simple and user-friendly in order for any new user to have an easy simulating experience and mitigate users running into problems when going to run a simulation. All members who design their respective code will be required to test their code extensively. Any new changes must be pushed to the project Github and must be approved/looked over by the rest of the group. The simulation will be computationally feasible for most hardware due to the simulation being more lightweight, leading to a very low chance of hardware failure. In the case of hardware failure, the machine will be replaced.

5.5 Project close-out plan

Following the completion of the project submission, the group will work to develop a presentation in which each member will discuss their role in the cumulative efforts of this project's development. This presentation will also include some challenges faced and how the team was able to overcome them, as well as a demonstration of the product itself.

6 Technical process plans (Jonas)

6.1 Process model

The iterative-and-incremental life-cycle model is used to ensure work is being done in the most efficient way. This is where as each workflow is completed, the previous one is improved. Using this life cycle will allow errors to be found quickly and gives the team opportunity to fine-tune the product.

6.2 Methods, tools, and techniques

The waterfall methodology will be used since each phase of the process will be completed before starting the next. However, still using the iterative and incremental life cycle, the previous workflows will be improved. Java is the programming language used with the help of java libraries like Java Swing, JavaFX, jTrend and libGDX.

6.3 Infrastructure plan

The product can be run on operating systems : >= Mac OS X 10.0, >= Windows 10, >= Linux 5.0 on a personal computer.

6.4 Product acceptance plan

Acceptance of the product by our client will be achieved by following the steps of the iterative and incremental process.

7 Supporting Process Plan (John Uribe)

7.1 Configuration management plan

“#” The pound symbol will represent a configuration option that has functional restrictions and requires safety measures to ensure functional code execution. The list of these specific options will further be referred to by “Safety Controls”

- **Grid**
  - Responsible for displaying cell life actively as the game progresses
  - Responsible for giving the user the ability to initialize a custom cell pattern
- **Control Panel (Tabs)**
  - **Parameters(Tab)**
    - Main panel that is responsible for the game’s fundamental functions
    - **Button Layout**
      - #(Initialize Random): Initializes a random pattern on grid
      -
      - (Play/Pause): Starts the life cycle of each cell continually until pressed again at which the game will pause, showing the user the current state of the game’s grid.
      - #(Next): Is only functional when the game is in a paused state, purpose is to give the user the ability to view the games life state in a step-by-step procedure showing the next step in the cells life every time the button is called
      - #(Previous): Only functional in a paused state, the previous button will revert the grids life state one step
      - for every instance of the button call.
      - (Reset): Has no restrictions the reset button can be called at any time and will halt any and all code execution, reverting the grid to its default state, all parameters and configurations of the control panel remain unaffected.
    - **Grid Dimensions #**

        - The Grid Dimensions panel consists of two JRadio Buttons that give the user the options to initialize the game configuration in a 2D or 1D state; these options will be unable to be changed once the first life cycle starts, until the game state has been reset.
    - **Grid Size #**
        - The Grid Size panel consists of one JSlider with a minimum value of 0, maximum value of 100 and a step value of 10, responsible for setting both the width and length sizes of the grid effectively changing how many cells are available. The slider will be unable to be changed after the initialization of the grid, until the game state has been reset.
    - **Simulation Speed**
        - The Simulation Speed panel consists of one JSlider with a minimum value of 0, maximum value of 10, and a step value of one. It can be changed at any time in an inactive or active game state.The slider value will affect the life cycle speed of the game to its respective slider value.
    - **Cell Display Format**
        - The Cell Display Format panel will consist of two JRadio Buttons giving the user the option to display the grid in a Black(Active) and White(inactive) format, or a 1(Active) or 0(inactive) format the option will be able to be changed in any game state as its a purely graphical change will no effects to the games functionality.
- **Visualization(Tab)**
    - The visualization tab will only be available to the use if the “Black and white” Option is selected from the Cell Display Format in the Parameters tab, this tab will allow the user to customise the colour configuration of the inactive and active cells, this option will be functional in any game state as its a purely graphic change, not affecting fundamental code execution.
- **Statistics(Tab)**
    - The statistics tabs will offer the user no ability to change the configuration of the game, it only offers the user a view of multiple statistics regarding the current game state such as Active and inactive cells, cell population, percentage of gridspace used etc…
- **Import/Export(Tab) #**
    - The import/export tab will only be available to the user once the game is in a paused state.
    - **Import:** Will allow the user to import a file containing preset game configuration settings which will automatically adjust the current game configuration to match those of the import file.

        - **Export:** Will allow the user to export the current game configuration and state to a save file which will contain each and every specific game configuration value, for later use

7.2 Testing plan

The testing plan will consist of a cross reference between all safety controls, and functional code execution, each of the safety controls will go through multiple safety checks to ensure the control restrictions are functional and without bugs. For instance the restrictions on the initialise random control is that it must only be called when the game is in a paused state, so specific code will have to be implemented to prevent the use of this control when in a active game state, furthermore testing will have to be done once code has been implemented to ensure there are no bugs in the code that will allow the user to bypass the control restrictions.

7.2.1 Restrictions to Implement

- **(Initialized Phase):** A phase of the game where no active cell life has been created configurations are permitted to change although only considered an initialization phase under the circumstance where the lifetime of the game is equal to 0.
- (**Paused game state):** the game is considered as a paused state when the total lifetime is neither increasing or decreasing.
- **(Initialize Random):** Only functional in a paused game state.
- **(Next)/(Previous):** Only functional in a paused game state.
- **(Grid Dimensions):** Only functional at the initialized phase
- **(Grid Size):** Only functional at the initialized phase.
- **(Import/Export):** Only functional at a paused game state

7.3 Documentation plan

The documentation plan will be split into two categories: User Level and Developer level

7.3.1 User level

- **(Rule Documentation)**
    - Game Documentation will provide the user with the rules to the game specifically how the cells act in relation to one another; this is fundamental for proper use of the game.
- (**Config Documentation)**
    - The configuration documentation will explain the restriction on the configuration options telling the user when specific changes can and cannot be made.

7.3.2 Developer level

- Developer Documentation will be in the back end of the project within the java files, Documentation will be provided above each sector the the GUI it will cover function purpose and restrictions, explaining how the user is intended to interact with the configurations and how those configurations should interact with the game memory.

7.4 Quality Assurance Plan (Kunj)

The *Cellular Automata Simulation* will undergo rigorous testing to ensure correctness, efficiency, usability, and performance.

7.4.1 Approach

This project will adopt a systematic quality assurance strategy to validate both performance and functionality. It is to assure that:

- The software product meets functional/nonfunctional requirements
- The development process follows best practices and standards
- The delivered product is error-free, user-friendly, and efficient

7.4.2 Quality Assurance Methods

Testing Strategy

- **Unit Testing**
  - Ensure individual modules, such as the grid initialization, step updates, visualization, will be tested independently
- **Integration Testing**
  - Verifies the interaction between modules, such as how changes to the cell state impact visualization/statistics.
- **System Testing**
  - Tests the entire system under various configurations *(grid sizes, speeds, rules, and so on)* to validate compliance with requirements
- **Performance Testing**
  - Simulation speed for large grid sizes *(100x100)*
  - Memory usage while running extended simulations
  - Efficiency for smooth visualisations
- **User Testing**
  - Conducted with a sample of target users to assess usability

Code Standards

- Code will adhere to Java coding practices and IEEE Standard 1058 to make sure it is clearly defined, complete, and testable
- Version control ensures stable builds and rollback capabilities

Configuration Management

- A Git-based workflow is used to track changes and manage versions

- Bugs and issues are logged and categorized

Quality Control Methods

- Bug count per module, tracked via Github
- Performance benchmarks
- User feedback ratings from test participants

7.5 Reviews and audits plan

7.5.1 Review and Audit

- Ensure software development aligns with project requirements
- Detect and resolve defects before major releases
- Verify compliance with quality assurance standards

7.5.2 Methods and Procedures

Management Progress Reviews

- Conducted weekly to assess project status and task completion
- Discuss potential next steps

Developer Peer Reviews

- Each new feature will undergo peer review before merging
- Review will check for
    - Code correctness
    - Adherence to code standards

7.5.3 Final Submission

The client (instructor) will conduct a final assessment to:

- Confirm all project requirements are met
- Evaluate software functionality, usability, and documentation
- Identify any defects or areas for improvement

7.6 Problem resolution plan

Effective problem resolution is essential for maintaining software quality. This plan establishes the methods, tools and procedures for reporting and analyzing software issues throughout the project.

7.6.1 Issue Reporting and Tracking

All software issues will be logged and tracked using GitHub

- Issue description will give clear details on the problem and its impact
- Assigned developer responsible for investigating and fixing the issue
- Status updates to ensure progress

7.6.2 Resolution Workflow

- Issue Logging - Team or users report defects
- Investigation/Fixing - Debugging
- Review/Testing - Peer Review
- Closure - Merge once fixed

7.7 Subcontractor management plan (Suleman)

Identification of Needs:

- Determine which tasks require expertise beyond the internal team's capabilities for example UI design, performance optimization and security audits.

Vendor Selection:

- Evaluate potential subcontractors based on criteria like experience, cost, reliability, and availability.
- Selection criteria ensures that the best candidate is chosen for the task.

Agreement and Documentation:

- Establish a clear contract specifying the project scope, deliverables, timelines, and payment terms.
- Helps in avoiding misunderstandings and ensures accountability.

Performance Monitoring:

- Regularly review the subcontractor's progress and provide feedback.
- Ensure that their work aligns with the project's quality and timeline expectations.

.**Risk Management:**

- Develop contingency plans in case subcontractors fail to meet expectations or deadlines.

Escalation Process:

- Establish procedures for handling disputes or performance issues with subcontractors.

7.8 Process improvement plan

To enhance project efficiency and quality, the following process improvement strategies will be implemented:

Retrospective Analysis:

- Conduct regular team meetings to review past tasks.

- Identify strengths, weaknesses and areas for improvement

Issue Tracking:

- Maintain a log of common issues(bugs, design, flaws, inefficiencies)
- Learn from the past mistakes to prevent recurrence in future iterations.

Skill development:

- ENcourage team members to research, attend training, or share knowledge.
- Improves overall team capability and efficiency

Process Refinement:

- Adjust workflows, methodologies, and tools based on feedback.
- Example: If testing delays are common, integrate automated testing earlier.

Metrics for Improvement:

- Track key performance indicators (KPIs) such as defect rates, task completion efficiency, and adherence to deadlines to measure project success.

Automation Tools:

- Utilize software tools for version control, continuous integration, and testing automation to enhance efficiency and reduce manual errors.

8 Additional plans (Devarth)

Training: Given the intuitive design of the game, we will be training all group members to familiarize users with the features and functionality
Deployment: Running the game will be shown, and the development team will assist with initial set-up. Minimum hardware requirements will be specified to ensure smooth operation.
Maintenance: The team will provide corrective maintenance for the period of the course. This includes bug fixes, resolving errors, and addressing any other issues that may occur.
Post Delivery: A feedback mechanism will be implemented to collect user suggestions and issues after delivery. This will help prioritize corrective and enhancement maintenance tasks.