Google Summer of Code' 2025 Proposal

Math Games

March 2025

—

Bishoy Wadea - Cairo, Egypt

bishoyw.fathy@gmail.com

+201556562502

GitHub: Bishoywadea (-NoName)

LinkedIn: Bishoy Wadea | LinkedIn

Resume: Bishoy Wadea

University and Current Education: Faculty of Computer Engineering, Cairo University

Timezone: Cairo, Egypt (UTC+2)

Why sugar Labs?

When I came across Sugar Labs, their mission to enhance education through technology immediately resonated with me. Their dedication to providing open-source educational tools and engaging activities for children, along with their involvement in the One Laptop per Child initiative, aligns perfectly with my passion for using technology to support learning. With a background in Python and experience in game development using Pygame, I explored their projects and found a strong connection with my interests. Now, I'm excited to apply for the Google Summer of Code (GSoC) 2025 program to contribute to Sugar Labs and positively impact education through technology.

More about Me?

I'm Bishoy, a third-year Computer Engineering student with a deep passion for technology and the open-source software philosophy. Previously, I interned at Microsoft Egypt as a Software Engineer, serving as an Applied Scientist on the Cairo Shopping Team. My experience spans backend development and mobile application creation using Android and Flutter. As a member of Cairo University's Formula Racing Team's Autonomous Driving division, I contribute to developing autonomous systems for our vehicles, integrating my expertise in Python to enhance their performance. With a solid foundation in computer science, including studies in algorithms, data structures, and computer architecture, I'm eager to leverage my skills and experiences to make meaningful contributions to the tech community.

Contributions To Open Source Community

1. **Connect Four Activity**
   - Code : http://github.com/BishoyWadea/ConnectFour
   - Video for Game: 🎬 2025-03-13 06-51-43.mkv
   -
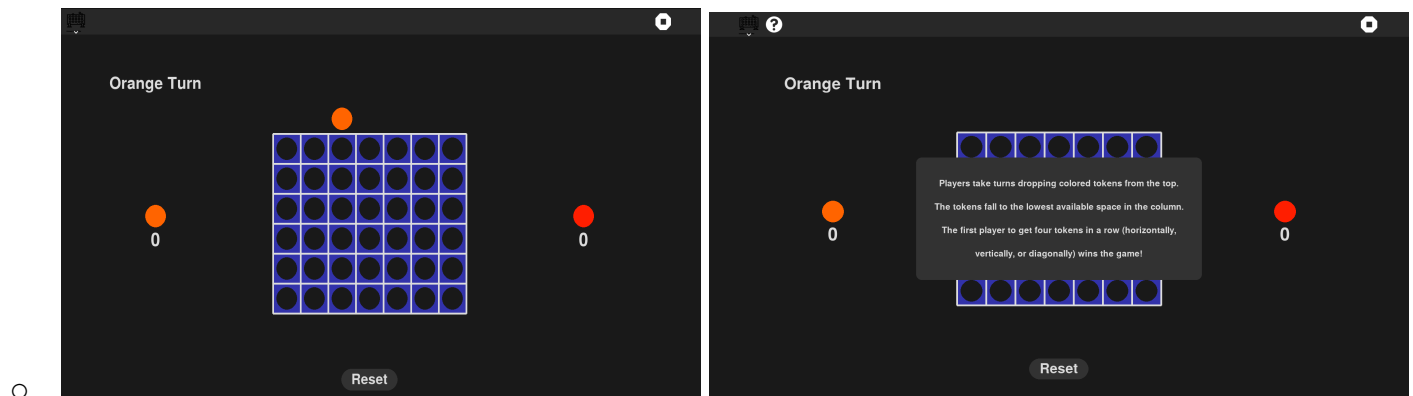
- ○ I have created a [Connect Four game](#) Activity for the Sugar desktop environment. This is a classic two-player strategy game where players take turns dropping colored tokens into a grid with the goal of connecting four tokens in a row.

  Key Components:
  - `Main` class serves as the primary controller managing the game flow
  - `Frame` class handles the game board logic
  - `Animate` class manages visual animations

  Features:
  - Clear game status display showing current player's turnScore tracking for both players
  - Reset button functionality
  - Help system with game instructions
  - Visual feedback with hover effects
  - Turn-based gameplay between Red and Orange players
  - Token dropping animation
  - Win detection (horizontal, vertical, diagonal)
  - Score tracking system

  Technical Details:
  - Event-driven architecture for handling user inputs
  - Separation of rendering and game logic
  - Configurable parameters through a config module
  - Pygame for graphics and animation

2. Fifteen Puzzle Activity
- ○ Code : http://github.com/BishoyWadea/FifteenPuzzle
- ○ Video for Game: 🎬 2025-03-23 13-39-09.mkv

- ○
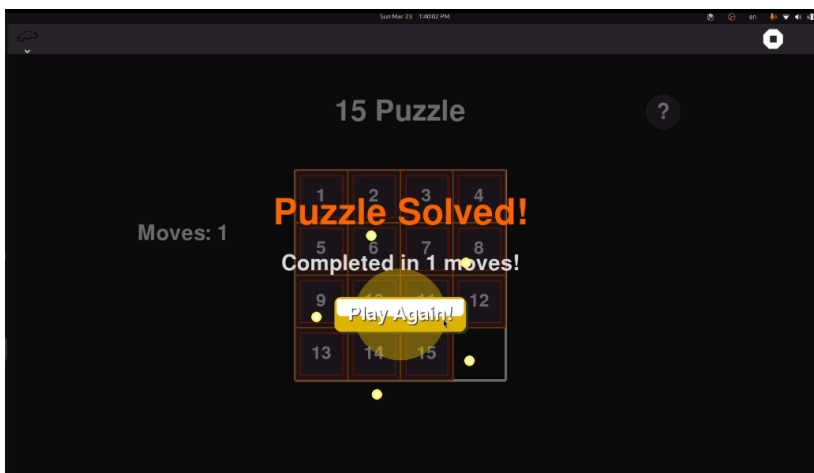
- I have created a [Fifteen Puzzle](#) (also known as Sliding Puzzle) Activity for the Sugar desktop environment. This classic puzzle game challenges players to arrange numbered tiles in sequential order within a 4x4 grid by sliding them into the empty space.

    Key Components:

    - `Main` class serves as the primary controller managing the game flow
    - `Board` class handles the game board logic
    - `Animate` class manages visual animations
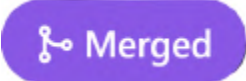   - `Tile` Individual tile behavior and rendering
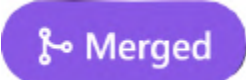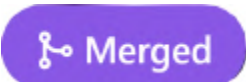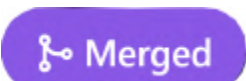
    Features:
    - Clean and intuitive grid-based layout
    - Visual feedback for tile movements
    - Move counter display
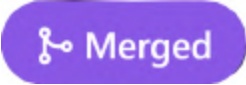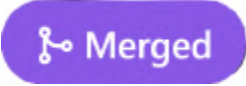   - New Game functionality
    - Smooth sliding animations
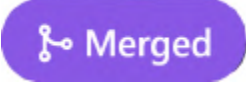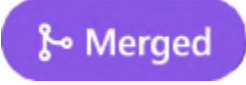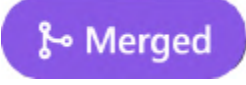
    Technical Details:

    - Event-driven architecture for handling user inputs
    - Efficient tile movement calculations
    - Separation of rendering and game logic
    - Configurable parameters through a config module
    - Pygame for graphics and animation

3. Other Contributions:

-

Organization	Pull Request	Status
Sugar Labs	https://github.com/sugarlabs/river-crossing-activity/pull/10	Merged
Sugar Labs	https://github.com/Bishoywadea/ConnectFour/pull/4	Merged
Sugar Labs	https://github.com/Bishoywadea/ConnectFour/pull/5	Merged
OneBusAway	https://github.com/OneBusAway/waystation/pull/26	Merged

OneBusAway	https://github.com/OneBusAway/waystation/pull/12	Merged
CheckStyle	https://github.com/checkstyle/checkstyle/pull/16468	Merged
CheckStyle	https://github.com/checkstyle/checkstyle/pull/16448	Merged
MetaCall	https://github.com/metacall/core/pull/553	Merged
MetaCall	https://github.com/metacall/currency-convert-example/pull/1	Merged

Project Goal

I plan to develop 9 Math games for Sugar within the GSoC'25 timeline.

The development of these activities will include basic functionalities of the game in addition to other features which are not mentioned particularly like Sound and themes.

How will it Impact Sugar Labs

Upon completion of this project, Sugar will feature 9 additional math games designed to make complex mathematical concepts accessible and engaging for children. These games, such as the Four Color Map Game, Broken Calculator, Soma Cubes, Fifteen Puzzle, Euclid's Game, Odd Scoring, Make An Identity, Number Detective, and Sorting Hat AI, are crafted to sharpen problem-solving abilities, enhance pattern recognition, and introduce foundational AI principles. By integrating these activities, Sugar aims to promote a deeper interest and understanding in mathematics among young learners, aligning with the educational benefits of using math games to improve engagement and learning outcomes.

Tools and Technologies:

Python, Pygame, Sugar

Project Size

350 hours

Proposal Description

The proposal will consist of the games implemented, an envisioned graphical interface, the features of the game, and the implementation approach taken.

1. Four Color Map

    a. **Idea:** The game inspired by [Four Color Theorem](#), The activity will consist of a map divided into multiple regions, and the player's goal is to color the entire map while following the four-color theorem—no two adjacent regions can have the same color.

    b. **Features:**

      i. **Settings** will allow customization of the game's visuals and configurations.

      ii. **Configurations** will include adjusting the complexity of the map, such as the number of regions and their shapes. As shown in point **'d'**

      iii. **Gamifications** will track the number of moves taken by the player, and a score system may be included to encourage optimization (e.g., completing the map in the fewest moves).

      iv. **Undo option** will be available, allowing the player to correct mistakes without restarting the entire game.

      v. **Judge** will notify the player when the map is successfully colored while following the four-color rule.

    c. **Graphical interface**
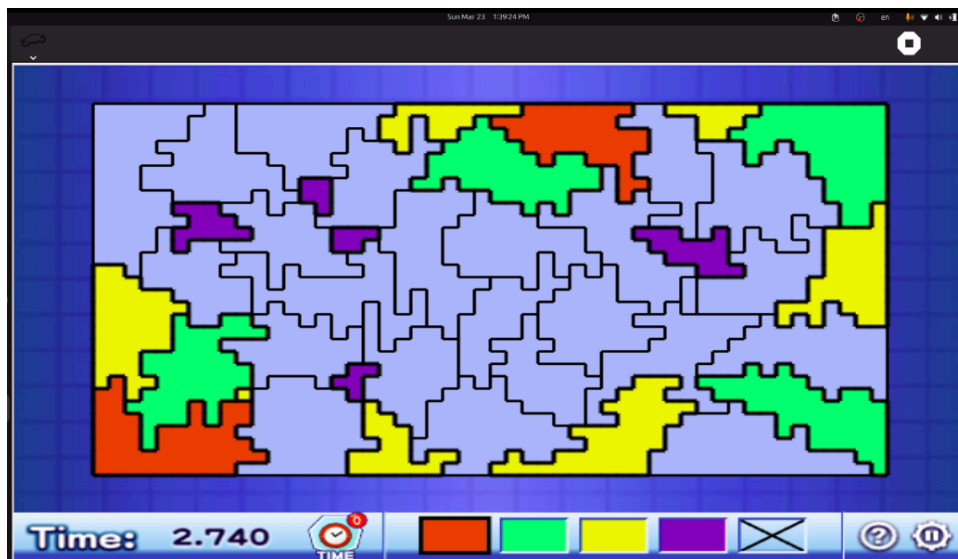
d. Types of Shapes: inspired by transum
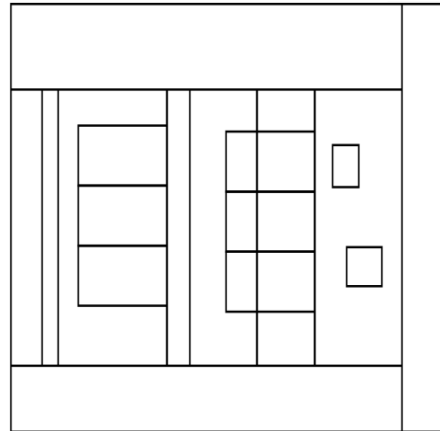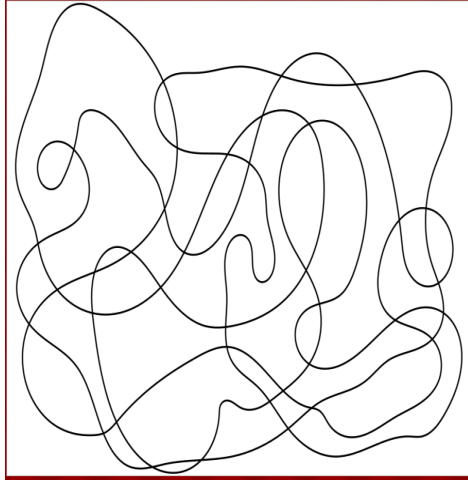
e. High level design details
  i. **Map Representation:** Model the map as a planar graph, where each region corresponds to a vertex, and edges connect vertices representing adjacent regions. This structure facilitates the application of graph coloring algorithms.
  ii. **Graph Coloring Algorithm:** Implement a backtracking algorithm to solve the M-Coloring Problem, ensuring that no two adjacent vertices (regions) share the same color. This approach systematically assigns colors to vertices while adhering to the four-color constraint.
  iii. **Undo Functionality:** Implement an undo stack data structure to store previous states of the map, enhancing the user experience by providing flexibility in gameplay.
  iv. **Completion Validation:** Develop a validation function that checks the entire map to confirm that all regions are colored and that no adjacent regions share the same color. Trigger a completion event when the player successfully colors the map according to the four-color theorem.

2. Broken Calculator

a. **Idea:** Inspired by [Broken Calculator](#) challenge, this game tasks players with creating five different equations that each equal a given target number using addition, subtraction, multiplication, or division. To score points, players must come up with unique equations of varying complexity, encouraging creative problem-solving. Beyond just arithmetic, the game helps build fluency in basic math operations while promoting flexible thinking and a deeper understanding of numbers.

b. **Features:**
   i. **Flexible Equation Creation:** The game supports the use of all primary arithmetic operations—addition, subtraction, multiplication, and division. Players can craft simple or complex equations with multiple terms.
   ii. **Scoring System:** Each valid equation that matches the target number earns points, motivating players to think creatively and explore various mathematical strategies.
   iii. **Fancy UI:** the game will have friendly light and dark theme
   iv. **Sounds:** interactive sound effects on correct and wrong equations

c. **Graphical interface**
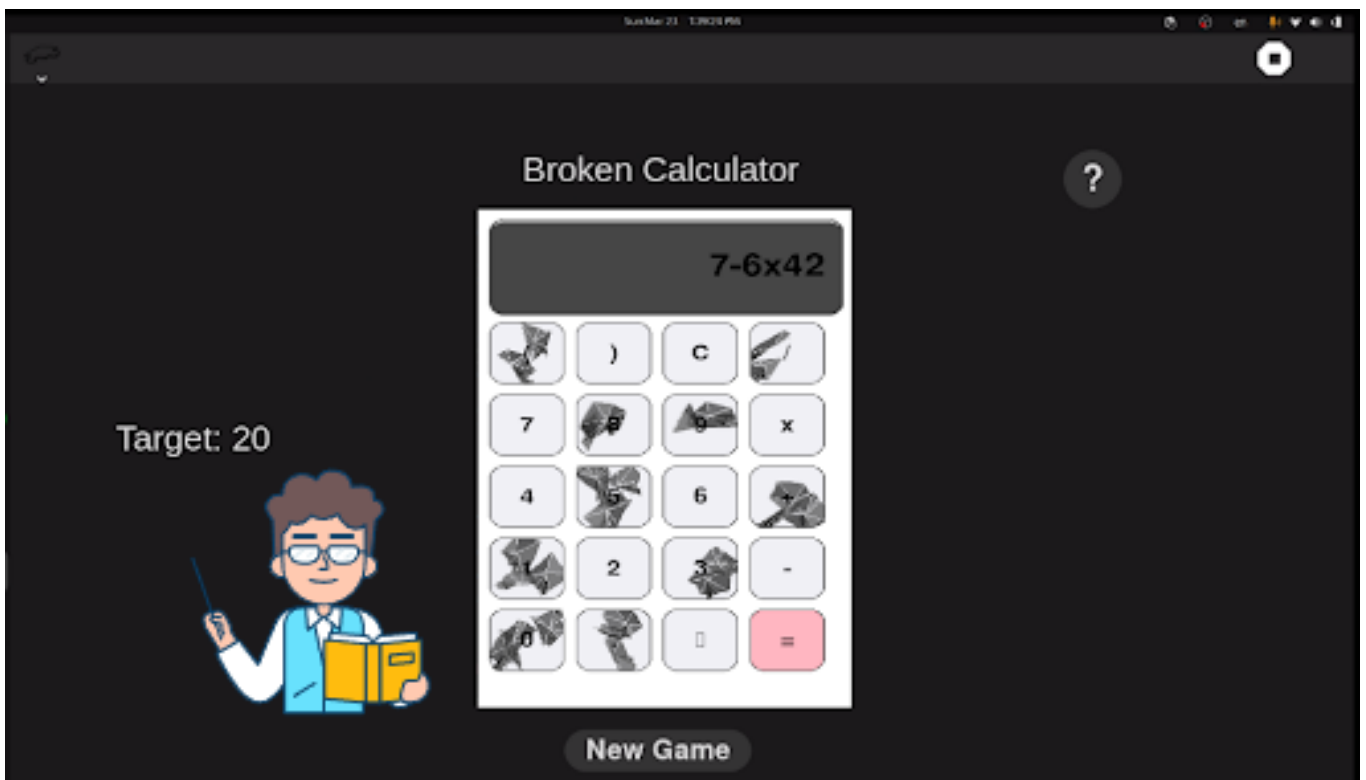
d. **High level design details**
   i. **Algorithm:** the game dynamically generates target numbers that can be achieved through at least five distinct equations using the available, non-disabled calculator keys.

ii. **Data Structure:** Utilize arrays or hash sets to represent the calculator's keys, marking certain digits and operations as disabled to simulate the "broken" aspect.

iii. **Data Structure:** Use a stack-based approach to evaluate postfix expressions, handling operations like addition, subtraction, multiplication, and division.

3. Soma Cubes

a. **Idea:** The game inspired by [Soma Cubes](#) The Soma Cube Game is a 3D puzzle game where players need to fill a target shape using various geometric pieces. The game combines spatial reasoning, strategy, and problem-solving skills in an engaging 3D environment.

b. **Graphical interface:** 🎬 2025-03-23 21-07-41.mkv

c. **Game Components**

   i. **Pieces:** The game features four different piece types: (images below)
   1. L-Shape (L): Four cubes forming an L shape
   2. T-Shape (T): Four cubes forming a T shape
   3. Z-Shape (Z): Four cubes forming a Z shape
   4. A-Shape (A): Four cubes forming a 3D corner shape
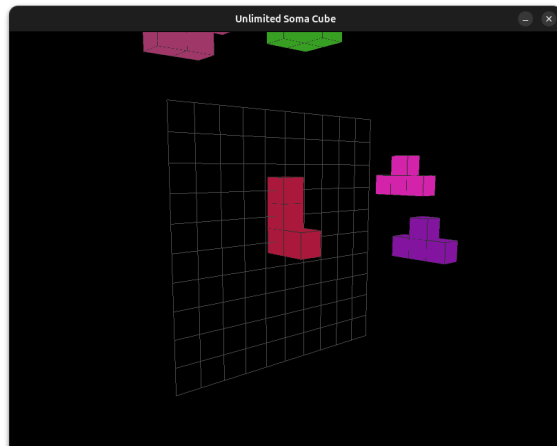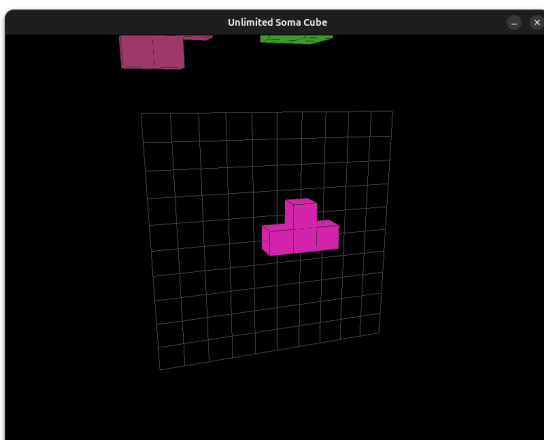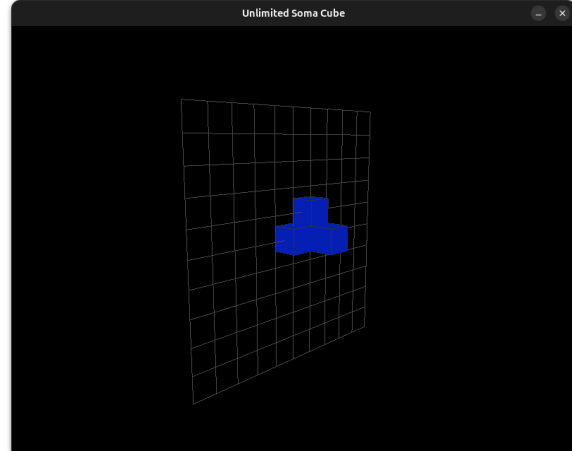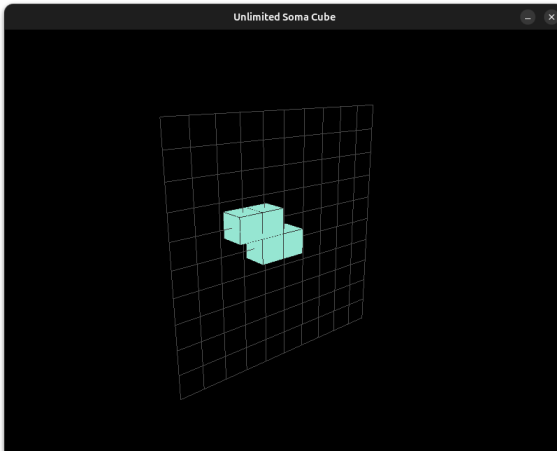
ii. **Target Shapes:** Available target shapes include:(images below)
1. L-Shape: A basic L-shaped pattern
2. Square: A 2x2 flat square
3. T-Shape: A T-shaped pattern
4. Stairs: A stepped pattern forming stairs
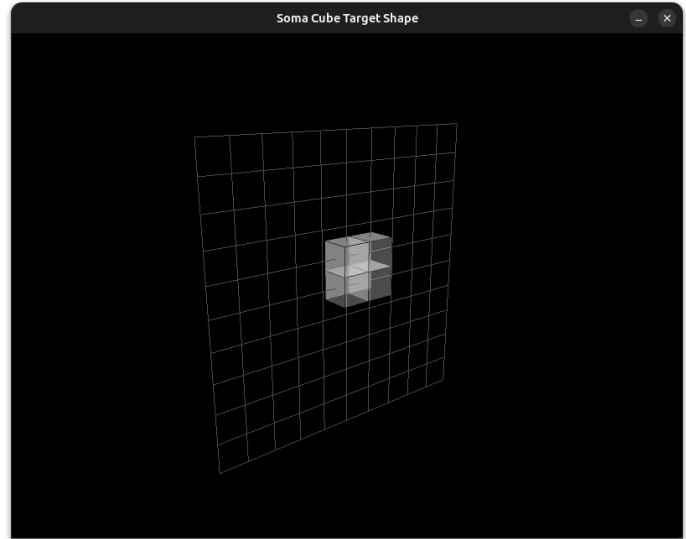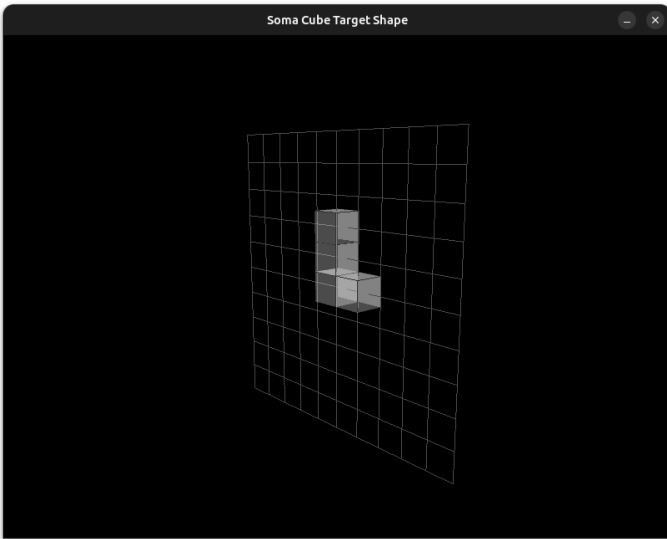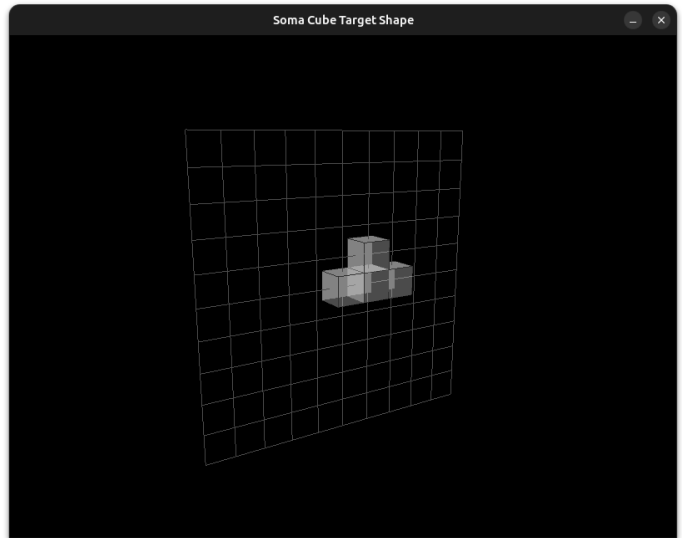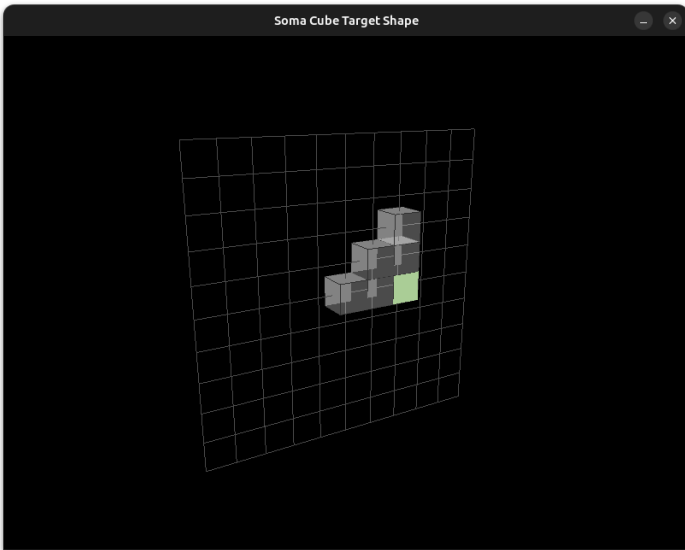
iii. **Rules:**
1. Players must fill the entire target shape with pieces
2. Pieces cannot overlap with each other
3. Pieces can only be placed in valid positions
4. The game ends when the target shape is completely filled
5. Pieces must be placed in empty spaces

    iv.   **Controls:**
1. **W:** Move piece up
2. **S:** Move piece down
3. **A:** Move piece left
4. **D:** Move piece right
5. **Left Arrow:** Rotate piece horizontally counter-clockwise (Y-axis)
6. **Right Arrow:** Rotate piece horizontally clockwise (Y-axis)
7. **Up Arrow:** Rotate piece vertically backward (X-axis)
8. **Down Arrow:** Rotate piece vertically forward (X-axis)
9. **Mouse Drag:** Rotate camera view
10.   **Spacebar:** Place current piece

  d. **High level design details**
    i.   **Piece Structure:**
1. vertices: Vec3[]   # Unit cube positions
2. transform: Matrix4 # Position & rotation
3. bounds: AABB   # Bounding box
    ii.   **Space Structure:**
1. occupied_spaces: HashSet<Vec3>
2. piece_registry: HashMap<UUID, Piece>
3. collision_grid: SpatialHash<Vec3>
    iii.   **Target Shape Structure:**
1. target: Set<Vec3>   # Required positions
2. bounds: AABB   # Shape boundaries
3. validation_map: BitSet # Filled states
    iv.   **Technical Stack:**
1. OpenGL/GLSL for rendering
2. Linear Algebra for transformations
3. Spatial Partitioning for optimization
4. Event-Driven Architecture
    v.   **Core Functions:**
1. def **check_collision**(piece: AABB, grid: SpatialHash) -> bool
2. def **transform**(vertices: Vec3[], matrix: Matrix4) -> Vec3[]
3. def **validate**(occupied: Set<Vec3>, target: Set<Vec3>) -> bool

4. Fifteen Puzzle: previously started working on it during the pre-GSoC period <u>refer to</u>

5. Euclid's Game

a. **Idea:** The game inspired by [Euclid's game](#) is a two-player mathematical strategy game that illustrates the principles of the Euclidean algorithm, particularly in finding the greatest common divisor (GCD) of two numbers. The game begins with two unequal positive integers written on a board. Players alternate turns, and on each turn, a player writes a new number on the board, which is the positive difference of any two numbers already present. The new number must be distinct from all numbers previously written. The game continues until a player cannot make a valid move; this player loses the game.

b. **Features:**

   i. **Adaptive Difficulty Levels:**
      1. Offer various difficulty settings to different skill levels.
      2. Adjust the complexity of starting numbers and provide AI opponents with varying strategies.

   ii. **Multiplayer Modes:** Enable local multiplayer for shared devices.

   iii. **User-Friendly Interface:**
      1. Design intuitive menus, buttons, and dialogs for easy navigation.
      2. Ensure the interface is responsive and accessible across devices.
      3. Has 2 themes (Light and Dark)

   iv. **Dynamic Animations:**
      1. Animate number selections and differences to visualize the game's progression.
      2. Use transitions to highlight the relationship between numbers, making abstract concepts more tangible.

   v. **Undo moves:** An undo option will be available, allowing the player to correct mistakes without restarting the entire game.

c. **Graphical interface**
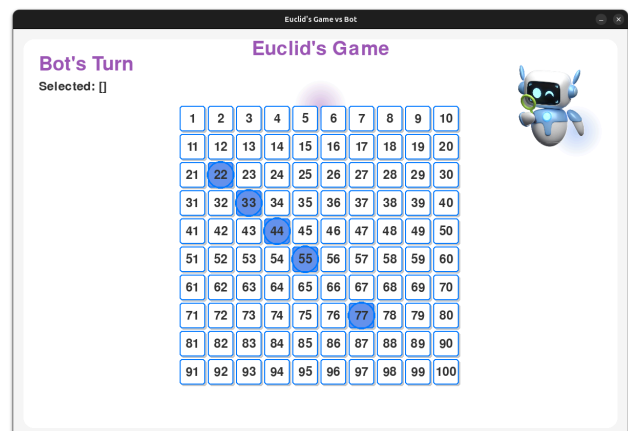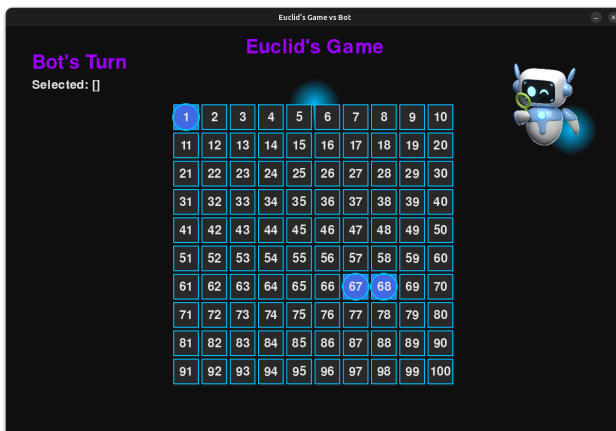
d. **High level design details:**
  i. **Bot:** bots will inherits from Player class
     1. **Easy:** Random valid move selection, No look-ahead, Ignores winning opportunities
     2. **Medium:** One-step look-ahead, Basic pattern recognition
   3. **Expert:** Full minimax implementation, Optimal play strategy

6. Magic Moving Game:
  a. **Idea:** is inspired by odd scoring, strategic turn-based game played on a N-cell grid where you compete against the computer. Starting from the rightmost cell, players take turns moving a character left by 1, 2, or 3 spaces using the numbered buttons. The game ends when the character reaches the finish line on the left, and the winner is determined by the total number of steps taken by both players - if the total is even, the player wins; if odd, the computer wins.
  b. **Features:**
    i. **UI:** the game have 2 theme (Light and Dark)
    ii. **Playing Modes:** the game can be played single player against computer bot or 2 players on the same machine
    iii. **Playing character:** you can choose the character to play with a boy or a girl
  c. **Graphical Interface:**
    i.
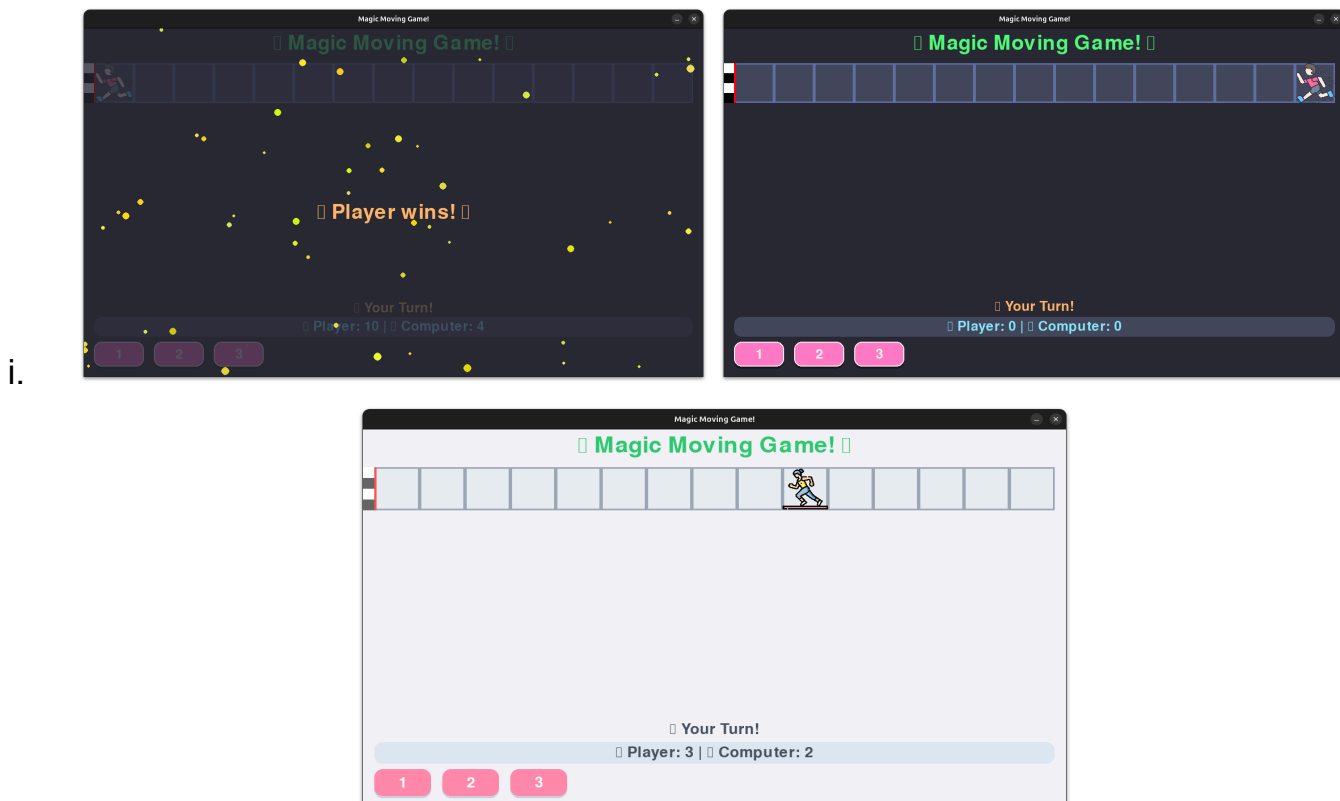
d. **High level design details:**
      i. **Bot:** Deterministic decision tree, [Minimax algorithm](#) for move evaluation

7. Magic Number Grid:

   a. **Idea:** inspired by [latin squares](#) and the game play by [puzzletronic](#), The objective of a Latin Square is to complete a partially filled grid so that each symbol appears exactly once in each row and exactly once in each column. Similar to Sudoku but without the block condition.

   b. **Features:**
      i. **UI:** the game has 2 theme (Light and Dark)
      ii. **Taking Notes:** taking notes on cells without playing by pressing 'N'
      iii. **Wrong Moves Alert:** the computer judge the play and alerting you it is in the wrong place
      iv. **Sound system:** the game has background music and sound effect for wrong plays and putting numbers
      v. **Animation:** each move will have special animation and the game has winning effect

   c. **Graphical Interface:**

      i.
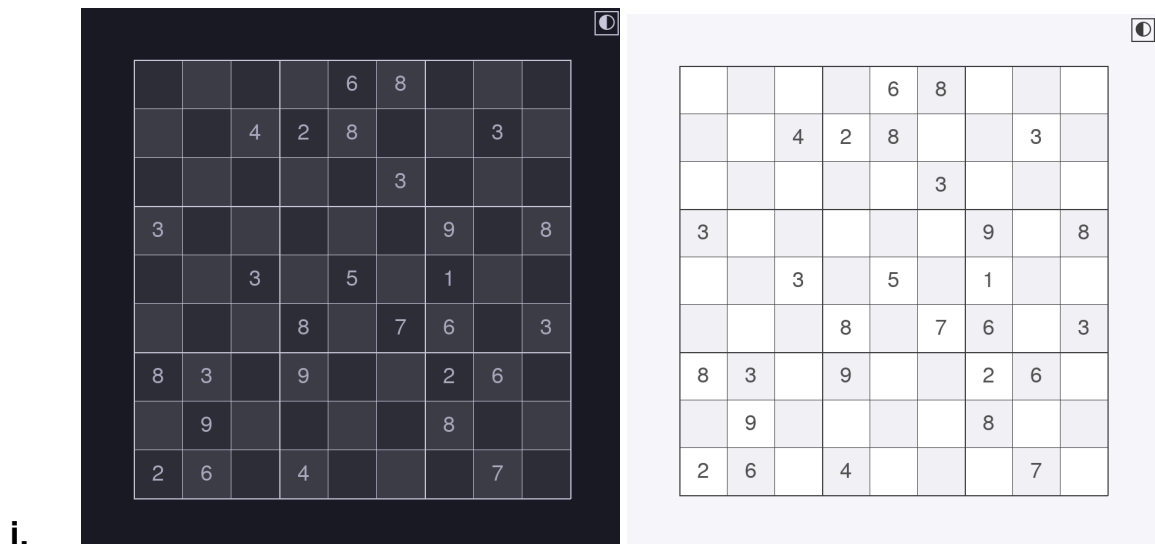
   d. **High level design details:**
   i. **Data structure:**
         1. 2D Array (Primary Grid)
         2. Boolean Mask Matrix (Fixed Numbers)
         3. Sparse Matrix (Notes/Pencil Marks)
         4. Hash Set (Error Tracking)

ii.   **Algorithms:**
1. Backtracking Algorithm
2. Random Permutation

iii.   **Time complexity:**
1. Grid Operations: O(n²)
2. Validation: O(n)

8. Sequence Wizard:

a. **Idea:** Number Detective is a fun and interactive math game designed to teach players about pattern recognition and AI learning. The game challenges users to input a sequence of numbers, and the AI tries to predict the next number based on observed patterns.

b. **Features:**

i.   **Learning:** AI gets smarter over time by learning from user corrections.

ii.   **Choosing Player:**
1. **Human:** Gives sequence of numbers.
2. **Wizard:** Guess the next number in the sequence

iii.   **UI:**
1. **Theme:** Light and dark themes
2. **Music:** interactive music along the game

c. **Graphical Interface:**
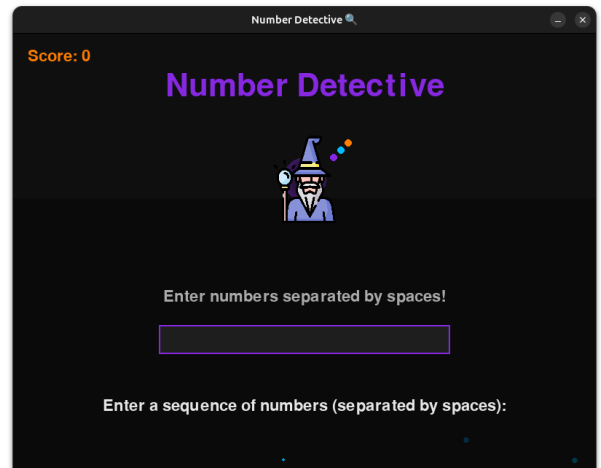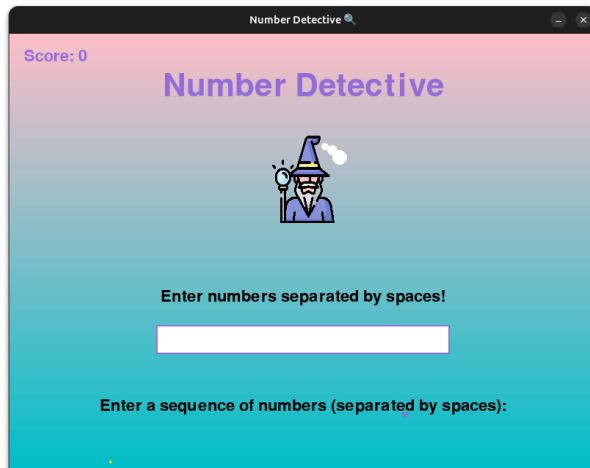
i.

d. **High level design details:**

i.   **System Architecture:**
1. **Game:** Pygame
2. **Storage:** JSON files (AI learning data)
3. **AI implementation:** Rule based + machine learning algorithms

ii.   **AI Implementation:**
  1. **Rule based:**Make algorithms to detect common sequences (Arithmetic, Geometric, Pascal's triangle, Fibonacci, etc)
  2. **ML algorithms:** If a pattern is not found, the player provides the correct answer then it is stored in a JSON/pickle file for future reference.AI reads the file at game startup to improve its predictions over time using one of the light weight algorithms (KNN, Decision Trees, etc) later we will judge each one performance and choose the best of them

9. AI Organizer:
  a. **Idea:** Sorting Hat AI is an interactive game that teaches how AI classifies objects.User label animals, shapes, or numbers, and the AI learns to classify new ones.If the AI makes a mistake, kids correct it, improving its learning over time!
  b. **Features:**
   i.   **Learning:** AI gets smarter over time by learning from user corrections.
   ii.   **Choosing Player:**
     1. **Human:** Gives sequence of numbers.
     2. **Wizard:** Guess the next number in the sequence
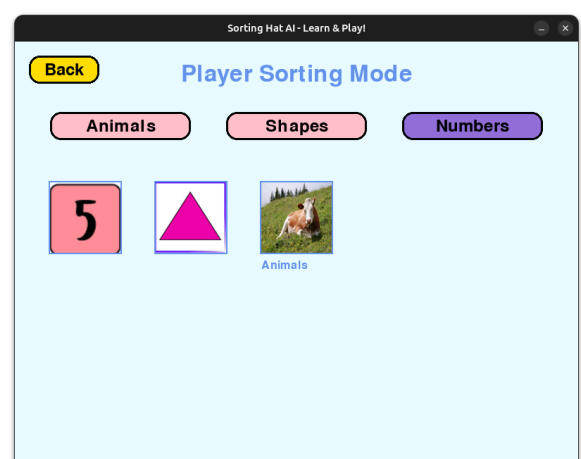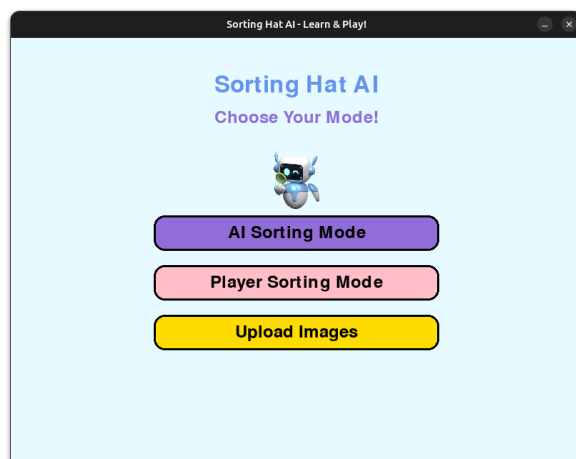   iii.   **UI:**
     1. **Theme:** Light and dark themes
     2. **Music:** interactive music along the game
   iv.   **Image Upload:** the user can upload any images he wishes to add which makes unlimited options for playing
  c. **Graphical Interface:**

   i.

d. **High level design details:**
      i.   **System Architecture:**
         1. **Game:** Pygame
         2. **Storage:** JSON files (AI learning data)
         3. **AI implementation:** Rule based + machine learning algorithms
      ii. **AI Implementation:**
         1. **ML algorithms:** If a pattern is not found, the player provides the correct answer then it is stored in a JSON/pickle file for future reference.AI reads the file at game startup to improve its predictions over time using one of the light weight algorithms (KNN, Decision Trees, etc) later we will judge each one performance and choose the best of them

Project Timeline:

The timeline will be divided into sprints every sprint will be a week

Sprint 0 8th may - 24th may	• Community bonding • Discuss meeting times • Discuss how evaluations will be conducted • Finalize ideas details • Create story diagrams for first 4 games
Sprint 1 25th may - 31st may	• Start working on Four Color Map • Develop reusable components to be used in other games (help buttons, color palette, etc) • Testing Four Color Map
Sprint 2 1st June - 7th June	• Start working on Broken Calculator • Testing Broken Calculator
Sprint 3 8th June - 14th June	• Get feedback on Four Color Map from mentors after testing and fix issues and bugs • Research space translation and rotation for Soma Cubes • Design the algorithms and data structures used in the game
Sprint 4 15th June - 21st June	• Get feedback on Broken Calculator from mentors after testing and fix issues and bugs if found • Start Working on Soma Cubes UI and animations • Integrate UI with moving Algorithms • Testing Soma Cubes

Sprint 5 22nd June - 28th June	● Continue working on the existing code of 15 Puzzle and fix issue and comments on it ● Get feedback on Soma cubes and 15 puzzle from mentors after testing and fix issues and bugs if found
Sprint 6 29th June - 5th July	● Create story diagrams for the remaining games ● Document the first four games and prepare for the Mid-Term Evaluation
Sprint 7 6th July - 12th July	● Start Working on Euclid's Game ● Compare between different implementations of the bot
Sprint 8 13th July - 19th July	● Start Working on Magic Moving Game ● Compare between different implementations of the bot ● Testing and get feedback over Euclid's Game
Sprint 9 20th July - 26th July	● Fixing bugs reported in Euclid's Game ● Start Working on Magic Number Grid Game ● Develop low complexity bot algorithm and check if the grid is solvable ● Testing and get feedback over Magic Moving Game
Sprint 10 27th July - 2nd August	● Fixing bugs reported in Magic Moving Game ● Start Working on Sequence Wizard ● Research the best model for efficiently predicting numbers in a sequence without requiring large datasets ● Testing and get feedback over Magic Number Grid Game
Sprint 11 3rd August - 9th August	● Fixing bugs reported in Magic Number Grid Game ● Start Working on AI organizer ● Research the best model for efficiently classifying images without requiring large datasets ● Testing and get feedback over Sequence Wizard
Sprint 12 10th August - 25th August	● Fix any remaining bugs or issues ● Documenting the last 5 games ● Write final evaluation report

Final notes:

1. **Testing:** Ensure compatibility across different screen sizes and devices, examine edge cases, and guarantee error-free performance.
2. **Regular meetings:** Gather feedback from mentors to ensure smooth progress and high-quality development.
3. **Availability:** I will be available throughout the GSoC period from 09:00 to 22:00 (UTC+2).
4. **Post GSoC Period:** I am willing to continue contributing to Sugar Labs and maintaining these activities in case any bugs arise.