Sugarizer Game Activity Pack (GSoC 2020 Proposal)

Personal Details

Name: Utkarsh Raj Singh
E-Mail Id: u.rajsingh2503@gmail.com
GitHub Profile: https://www.github.com/utkarsh-raj
LinkedIn Profile: https://www.linkedin.com/in/utkarsh-raj25
First Language: English, Hindi (Native)
Location: India
Timezone: GMT +5:30

Open Source Contributions

Contributions to Sugarlabs

I have been an active member of the Sugarlabs community for more than a year. Started engaging and contributing in February 2019, and since then it has been a learning enriched journey. Mentored for Google Code In with Sugarlabs for Sugarizer Projects in December 2019. I am comfortable with the Sugarizer architecture and have built several patches:

- **#706**
  - **Built the Chess Activity as a part of the GSoC 2020 task**
  - **A full activity integrated in Sugarizer with Sugar Web, Bootstrap 3 and Presence**
- #386
  - Address the issue of non standard implementation of presencepalette.js in Memorize Activity
- #283
  - Fixed the issue of vanishing recent marks
  - The issue occured in the Stopwatch Activity for the recorded times vanished after a few recordings
- #282
  - Fixed the issue of responsiveness in the Memorize Activity
  - Tested for all modern viewport dimensions

- ➔ #280
    - ◆ Related to PR #283, discussed with the maintainers the justification of the issue
- ➔ #279
    - ◆ Fixed unexpected token in JSON Error and stopping the Activity from hanging
    - ◆ Applied data cleaning before parsing
- ➔ #278
    - ◆ Identified the vulnerabilities of RegEx DoS due to deprecated dependencies in the Developer Electron JS build for Sugarizer
- ➔ #277
    - ◆ Added the Click and Drag Feature after discussion with the maintainers
- ➔ #276
    - ◆ Discussed with the maintainers and safely closed the mild issue of extent of zooming out in the Color My World Activity
- ➔ #304
    - ◆ Proposed the Chopsticks Activity for Sugarizer
    - ◆ Worked on the JS version of the proposed implementation and submitted for review by the maintainers

Contributions to other Open Source Projects

I have been engaging with the Open Source community in general and contributing in various forms. Some of them are:

- **GirlScript Summer of Code** (My contributions) - Currently serving as a mentor for WebTech , we are guiding students to develop and maintain a Web Application for Testing the Tech Stack of websites.
- **freeCodeCamp** (#25829 , #25978) - Deliberated over the best programming practices with the developers for the guides in FCC.
- **Appwrite.io** (#107 , #116) - Fixed CSS bug in Hacktoberfest 2019, member of the organisation since then.
- **Mozilla Developer Network** (#26) - Fixed a bug in the tutorial code of the Web Speech API.

Open Source/Personal Projects

In the journey of learning, I also have developed several projects to implement the theory in practice:

- **Vue.js Activity Template for Sugarizer Activities** - A template inspired by the Ebook Reader Activity. Provides the basic functionality for the setting of Sugarizer Application on the web.
- **Vue.js practice** - A collection of mini projects and exercises to learn Vue.js.

    - **Monster Slayer** - A simple player vs. monster game, practice the basics of Vue.js
    - **Wonderful Quotes** - Components and slots practice
    - **Components - 1** - Practicing components fundamentals
    - **Components - 2** - Communication through components
    - **Directives** - Creating the directives to handle the click event
    - **Dynamic Components** - Switching between components with the component tag
- **Rich Text Editor** - A RTEditor developed as a prototype for the previous GSoC Application
    - Incorporates features like Font size, style, format, subscript, superscript.
    - Text align, copy paste and font color change and background color change.
    - Supports image uploads in the document.
- **Cook Book** - Bootstrap 3, Node.js, Express.js, MongoDB Atlas based web application with Multi-User Authentication, search and CRUD.
    - Cook Book - LIVE
- **APIs and Microservices** - Developed several services as APIs in JS and Python, including File Metadata, Header Parser, Timestamp and URL Shortener
- **Chat Mini** - WebSocket based online real time Instant Messaging Service, built with JS.
    - Chat Mini - LIVE
- **Chop Game** - The Japanese Chopsticks game written in Vanilla JS
    - Chop Game - LIVE
- **RESTful Blogging** - A Full Stack Web application developed in NodeJS
    - Utilizes MongoDB and has Express Framework and Semantic UI in the Frontend

Project Details

What am I making?

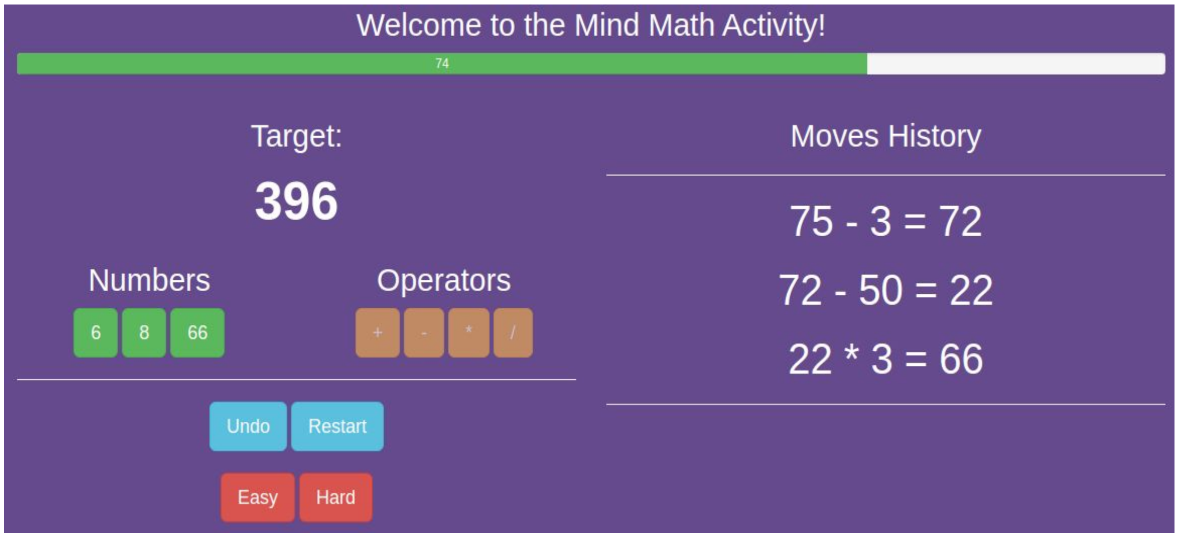
The Sugarizer Game Activity Pack includes the following activities:

MindMath Activity

The basic gameplay consists of the player trying to build a target number from the given numbers and the four basic mathematical operations. A developed prototype can be found here (GitHub)

For the development, the following features and use cases are proposed:

- The user gets **5 random numbers** and **4 basic mathematical operations** to start with.
- The user will have to use the numbers to reach a **target value**.
- Three difficulty levels:
  - **Easy** - The target number will be between **10 and 69**
  - **Medium** - The target number will be between **0 and 99**
  - **Hard** - The target number will be between **0 and 99** + **some operations will be mandatory**
- The 5 random numbers will be from **(1-4), (1-6), (1-8), (1-12), and (1-20),** respectively. Also, there will be 4 slots to calculate the answer.
- Tell the user, in case of loss, how far he was from the target value.
- Scoring - Every game will have scores, which will take into account the following factors:
  - The **number of steps (slots)** used (more is better)
  - **Number of operations** used (more is better)
  - **Time spent to solve the challenge** (less is better)
  - **Number of helpers** used (less is better)
- Two types of hints:
  - Solver - Will show the end to end possible ways to go from the starting numbers to the target number. The user will be able to **see the complete solutions for the given game.**
  - Helper - The helper will be able to **suggest the possible next moves based on the user's current moves.** Will be able to extend the user's idea and help him build a solution in his way.
  - For example:



75 - 3 = 72

72 - 50 = 22

22 * 3 = 66

Suppose this is the current state. The Solver will take the initial state and the target to display all the possible end to end solutions. On the other hand, the Helper will take the current state ([6, 8, 66] and 396), and tell the user the possible next steps he can take. Thus the Helper does not give away the complete solution, but instead helps the user in steps.

- Playing Modes:
    - Single Player Mode - Will play with the **timer** and can have **helper, solver, undo and restart.**
    - MultiPlayer Mode - Will play **against other users** in the **network** and will get a **common leaderboard sorted according to the score.** Will also have **best player features** for a series of rounds.
- Outline Design Modes - To let the user create custom shapes and share them as outlines for other players on the network.
- Timer Integration
- Restart, Fullscreen, Network Sharing, Difficulty Selection(Easy, Medium and Hard), Hints(Solver and Helper) and Undo Buttons.
- Local Database Storage (Journal Integration).
- Interactive tutorial and documentation.
- Sugar Web UI, Buddy Colors when possible.
- Multi device support and responsiveness on all screen sizes.
- Localisation with the webL10n library.

Tangram Activity

The activity will essentially give the player a set of Tangram pieces and will be expected to form a figure. The objective of the game will be to form a figure using all the seven pieces making them touch and not overlap anywhere. A developed prototype can be found here (GitHub)

For the development, the following features and use cases are proposed:

- The classic seven pieces of the Tangram will be given to the user.
    - Two large right triangles
    - One medium right triangle
    - Two small right triangles
    - One square
    - One parallelogram - has no reflectional symmetry, but only rotational symmetry, and its mirror image can only be obtained by flipping it over
- 3 Difficulty levels:
    - Easy Level - The user will be **guided on how to rotate and place** each of the pieces, **one at a time**. This will help the user to understand the game and how to approach each puzzle effectively.

      In other words, this will be like helping the user by hand - telling him **how to rotate and place each piece before the next one**. He will **not be able to pick up another piece before the current is placed properly**.
    - Medium Level - The **dotted lines will be for each of the pieces**. The user **will be able to rotate and place the pieces in any order they wish to**. The **color of the correctly placed piece will change**, indicating that the user has successfully placed that piece.

    - Hard Level - **Only the outline of the complete figure will be given**. The user will himself figure out how to correctly rotate and place the pieces.
- Special bonuses for pre defined complex figures - some figures will be more complex and difficult to complete than the others. **Special bonuses can be incorporated for hard figures.**
- **Timer based bonuses** for certain game modes and use cases.
- Save in the local database (Journal Integration)
- Multiplayer Mode:
    - Timer integrated plays to see who solves first.
    - Leaderboard according to the score, which will depend on:
        - Difficulty of the Tangram figure
        - Difficulty Level of the Game
        - Time taken to solve the puzzle
        - Number of hints used
    - Showcase mode so that the user can see others' Tangram Creations.
- Hint Feature - Show one of the pieces in the correct position.
- Restart, Fullscreen, Network Sharing, Difficulty Selection(Easy, Medium and Hard), Undo and Hints Button for the Tangram Activity.
- Local Database Storage (Journal Integration).
- Interactive tutorial and documentation.
- Sugar Web UI, Buddy Colors when possible.
- Multi device support and responsiveness on all screen sizes.
- Localisation with the webL10n library

Implementation Details

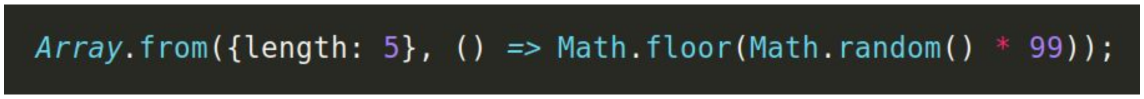
MindMath Activity

The features proposed are discussed in the implementational level:

- The solver based strategy solidifies the following game rules:
    - All given numbers are positive integers
    - Each given number may be used at most once
    - At no intermediate step in the process can the current running total become negative or involve a fraction
- The currently existing open source solvers are:
    - Numbers (numbers-js) - Implements an O(n) algorithm with the Reverse Polish Notation calculation strategy. We will use this to get all the possible solutions for a given set of numbers and a target value.
    - Cntdn - Utilises DFS on a graph such that to find the solutions. It is a library and the Readme encourages us to develop a custom UI for the solver.

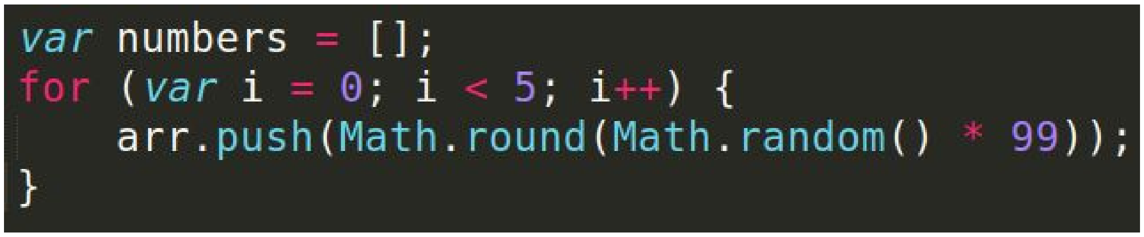
We can directly plug in these solvers, or write a custom version inspired by them. The latter will be better, because these are some of the most computationally efficient mechanisms to compute the result.

- Random numbers will be generated by JavaScript in the **numbers** array by the standard Math module and the Random function (Math.floor() and Math.random()), **according to the difficulty of the game**. The operators can be hard coded in the **operators** array.
  The ES6 way:



```
Array.from({length: 5}, () => Math.floor(Math.random() * 99));
```

  The Vanilla JS way:

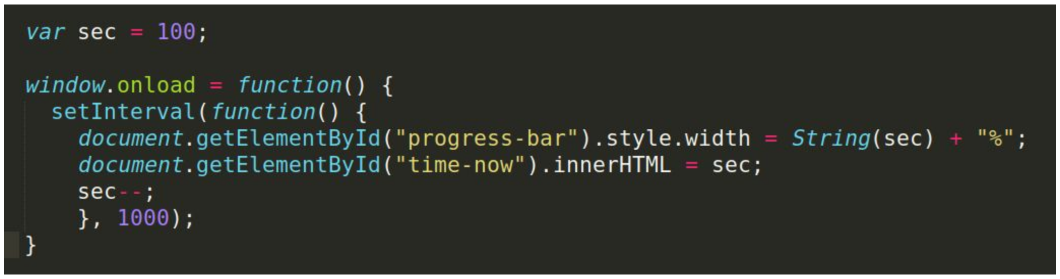


```
var numbers = [];
for (var i = 0; i < 5; i++) {
    arr.push(Math.round(Math.random() * 99));
}
```

- Using the numbers in the **numbers** array, we will obtain a **target** value for that round. This can be done by:
  - A random operational permutation of the operators and numbers
  - Generating a completely random target value and checking if it is possible to generate it from the given number, using backtracking algorithm

This generation will be in accordance with the difficulty level. This is necessary so that we know that it is indeed possible to reach the target value. In other words, we will be sure that at least one solution exists.

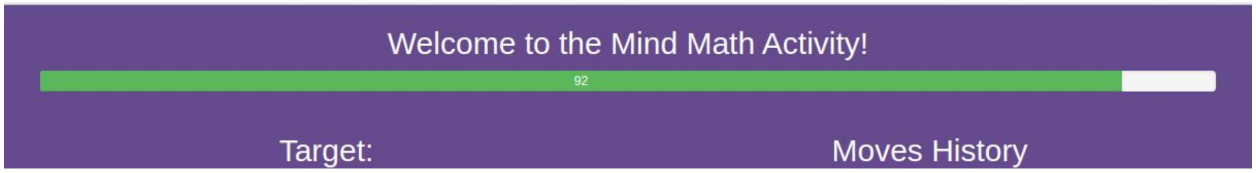
- For the timer, we can count in the logic and update the DOM with the values.The DOM can be updated either by:
  - The **watch** property in Vue.js can listen to the updates in the value of the time count variable and update the DOM.
  - Updating the **<span>** with JS, we can use the **setInterval()** function and configure it to listen to the **window.onload** event and start counting



```
var sec = 100;

window.onload = function() {
  setInterval(function() {
    document.getElementById("progress-bar").style.width = String(sec) + "%";
    document.getElementById("time-now").innerHTML = sec;
    sec--;
    }, 1000);
}
```

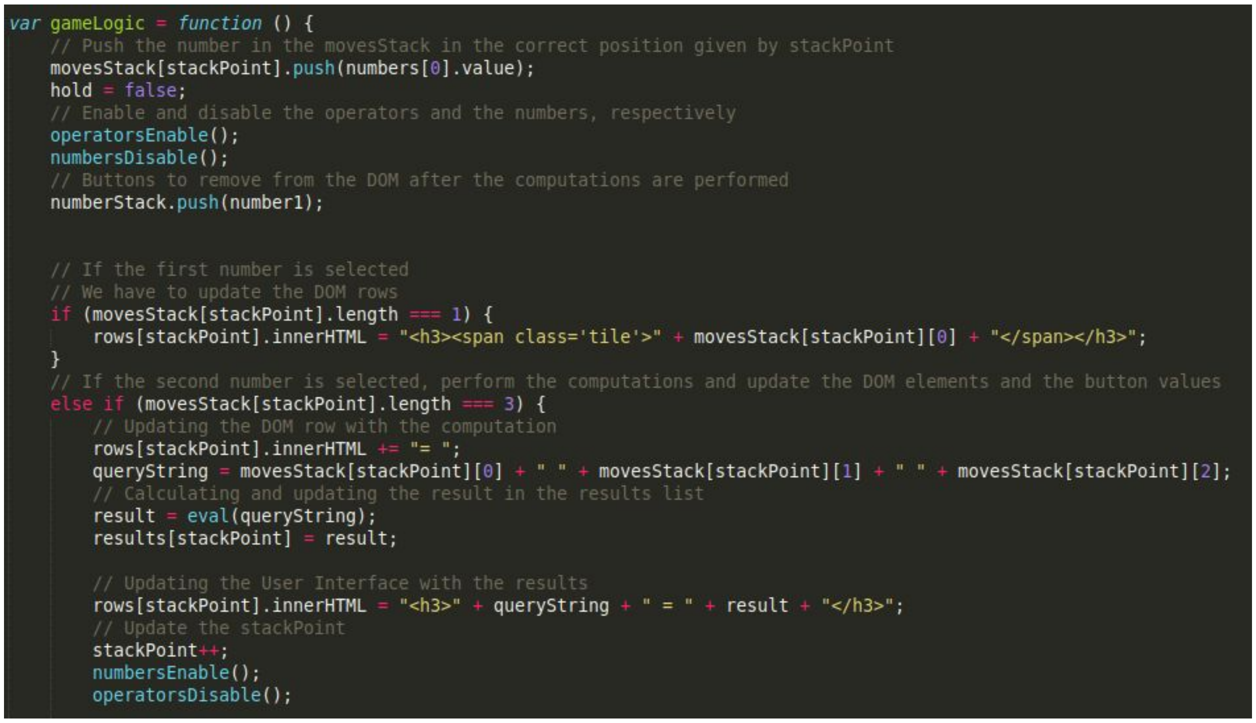
The player will still be able **to play the game after the timer ends, but extra points will be awarded if there is still time left.**

- **Enable/Disable Timer** - The user will be able to disable the timer as well. This will be possible in the single player mode and will help the user take his time to solve a puzzle. The UI will have an area to show the user the time he has spent on the problem.
- Bootstrap 3 Progress Bars can be used to display a dynamic and responsive timer in bar form. We have to routinely change the **style.width** to the new percentage.

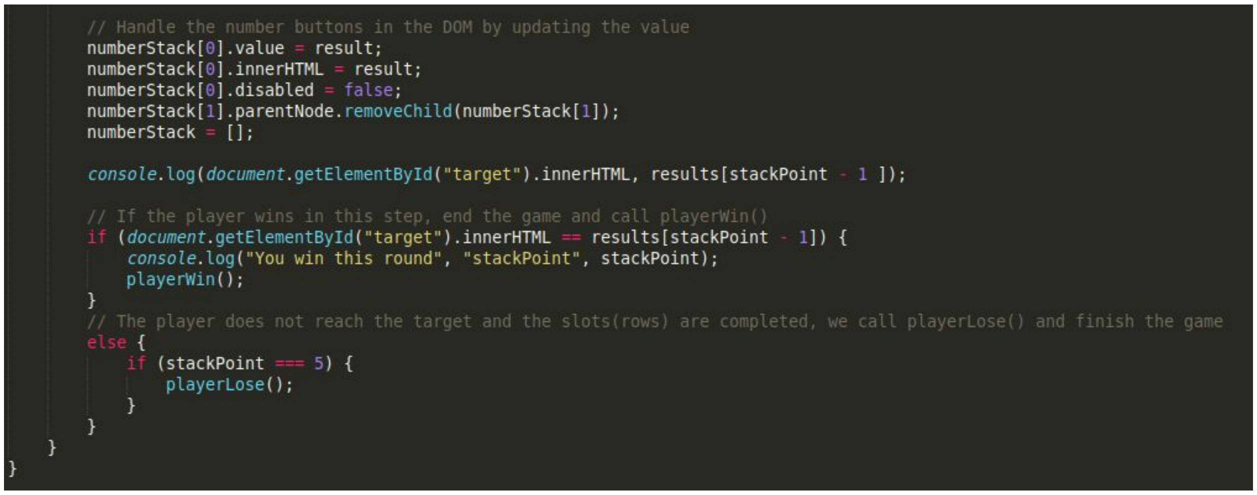


- The game state can be decided by the following variables and data structures:
    - State Arrays:
        - Numbers [] - Stores the randomly generated numbers and their corresponding DOM elements
        - Operators [] - Stores the 4 basic arithmetic operators and their corresponding DOM elements
        - movesStack [] - A 2D array which stores the computation for each move(slot or row).
          For example: **[[75, -, 3], [72, -, 50], [22, *, 6], [132, *, 3]]** is a movesStack for a game of 4 moves. This technique will help us in developing the Undo feature, as well as is good to display in the DOM using the **v-for** attribute given by Vue.js.
        - Results [] - Stores the result for each computation in the movesStack.
          For example: **[72, 22, 132, 396]** will be the Results array for the above movesStack
    - State Variables:
        - Target - Keeps the calculated target value for the current game.
        - Seconds - The time elapsed since the game started.
        - stackPoint - Keeps the pointer to the correct index of movesStack, that is, tells us which slot (or row) is to be filled next.
        - Difficulty - Keeps the current difficulty level picked by the user.
        - numOfOperators - Keeps the number of unique operators used by the user.
        - hintsUsed - Keeps the number of hints used
    - Methods:
        - numbersEnable()/numbersDisable() - To enable and disable the numbers and the operators
        - playerLose()/playerWin() - To show the post game analytics

- gameLogic() - To each of the button.addEventListener() to handle the logic

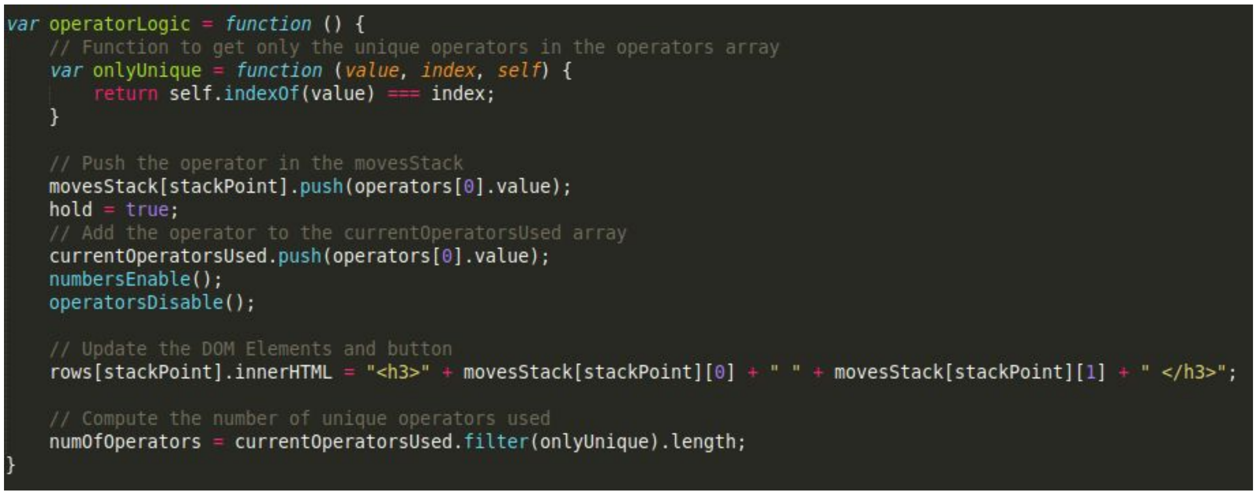


```
var gameLogic = function () {
    // Push the number in the movesStack in the correct position given by stackPoint
    movesStack[stackPoint].push(numbers[0].value);
    hold = false;
    // Enable and disable the operators and the numbers, respectively
    operatorsEnable();
    numbersDisable();
    // Buttons to remove from the DOM after the computations are performed
    numberStack.push(number1);


    // If the first number is selected
    // We have to update the DOM rows
    if (movesStack[stackPoint].length === 1) {
        rows[stackPoint].innerHTML = "<h3><span class='tile'>" + movesStack[stackPoint][0] + "</span></h3>";
    }
    // If the second number is selected, perform the computations and update the DOM elements and the button values
    else if (movesStack[stackPoint].length === 3) {
        // Updating the DOM row with the computation
        rows[stackPoint].innerHTML += "= ";
        queryString = movesStack[stackPoint][0] + " " + movesStack[stackPoint][1] + " " + movesStack[stackPoint][2];
        // Calculating and updating the result in the results list
        result = eval(queryString);
        results[stackPoint] = result;

        // Updating the User Interface with the results
        rows[stackPoint].innerHTML = "<h3>" + queryString + " = " + result + "</h3>";
        // Update the stackPoint
        stackPoint++;
        numbersEnable();
        operatorsDisable();
```



```
        // Handle the number buttons in the DOM by updating the value
        numberStack[0].value = result;
        numberStack[0].innerHTML = result;
        numberStack[0].disabled = false;
        numberStack[1].parentNode.removeChild(numberStack[1]);
        numberStack = [];

        console.log(document.getElementById("target").innerHTML, results[stackPoint - 1 ]);

        // If the player wins in this step, end the game and call playerWin()
        if (document.getElementById("target").innerHTML == results[stackPoint - 1]) {
            console.log("You win this round", "stackPoint", stackPoint);
            playerWin();
        }
        // The player does not reach the target and the slots(rows) are completed, we call playerLose() and finish the game
        else {
            if (stackPoint === 5) {
                playerLose();
            }
        }
    }
}
```

The operator button logic can be given here:



```
var operatorLogic = function () {
    // Function to get only the unique operators in the operators array
    var onlyUnique = function (value, index, self) {
        return self.indexOf(value) === index;
    }

    // Push the operator in the movesStack
    movesStack[stackPoint].push(operators[0].value);
    hold = true;
    // Add the operator to the currentOperatorsUsed array
    currentOperatorsUsed.push(operators[0].value);
    numbersEnable();
    operatorsDisable();

    // Update the DOM Elements and button
    rows[stackPoint].innerHTML = "<h3>" + movesStack[stackPoint][0] + " " + movesStack[stackPoint][1] + " </h3>";

    // Compute the number of unique operators used
    numOfOperators = currentOperatorsUsed.filter(onlyUnique).length;
}
```

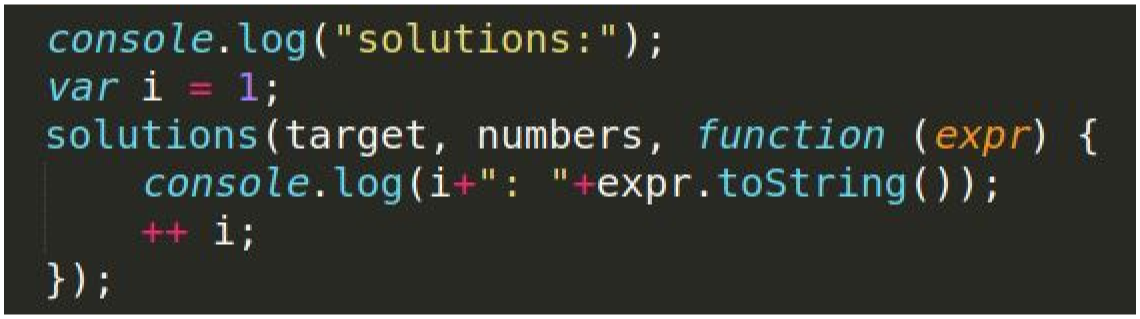
- We store the current user input of the 2 numbers and the operator in the movesStack array and update the results array dynamically.

- Scoring - as it is discussed in the use case section, we can develop an expression to calculate the score:
  - (stackPoint+1) - The number of slots used.
  - numOfOperators - The number of unique operators used.
  - Seconds - The time taken to solve the game
  - hintsUsed - The number of hints used

    Score = (((stackPoint+1)*numOfOperators) - (Seconds+hintsUsed))

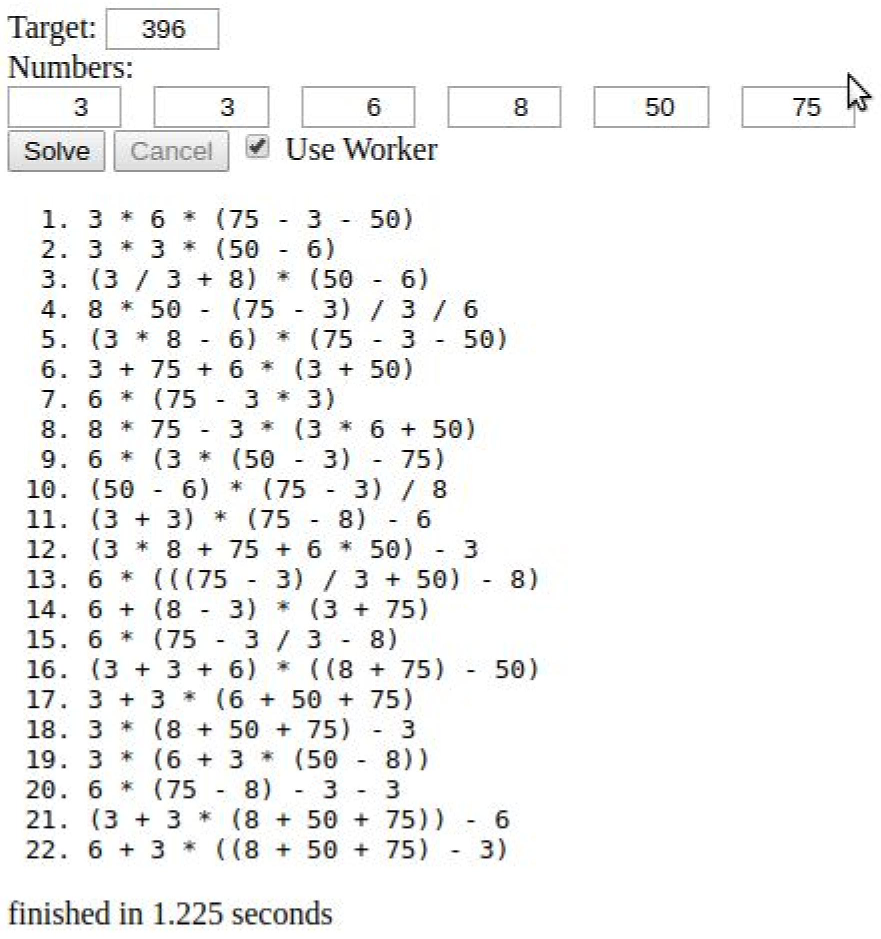
  Other expressions can be discussed and changed accordingly.

- Types of Hints:
  - Solver - Shows the complete solutions, end to end.
    For example, a call to the **solutions** function of the **Numbers-js** library gives us the solutions in the string form:



```
console.log("solutions:");
var i = 1;
solutions(target, numbers, function (expr) {
    console.log(i+": "+expr.toString());
    ++ i;
});
```

The solution example shows an implementation given in the **Numbers-js** GitHub Page:


Target: 396
Numbers:
3 3 6 8 50 75

```
 1. 3 * 6 * (75 - 3 - 50)
 2. 3 * 3 * (50 - 6)
 3. (3 / 3 + 8) * (50 - 6)
 4. 8 * 50 - (75 - 3) / 3 / 6
 5. (3 * 8 - 6) * (75 - 3 - 50)
 6. 3 + 75 + 6 * (3 + 50)
 7. 6 * (75 - 3 * 3)
 8. 8 * 75 - 3 * (3 * 6 + 50)
 9. 6 * (3 * (50 - 3) - 75)
10. (50 - 6) * (75 - 3) / 8
11. (3 + 3) * (75 - 8) - 6
12. (3 * 8 + 75 + 6 * 50) - 3
13. 6 * (((75 - 3) / 3 + 50) - 8)
14. 6 + (8 - 3) * (3 + 75)
15. 6 * (75 - 3 / 3 - 8)
16. (3 + 3 + 6) * ((8 + 75) - 50)
17. 3 + 3 * (6 + 50 + 75)
18. 3 * (8 + 50 + 75) - 3
19. 3 * (6 + 3 * (50 - 8))
20. 6 * (75 - 8) - 3 - 3
21. (3 + 3 * (8 + 50 + 75)) - 6
22. 6 + 3 * ((8 + 50 + 75) - 3)
```

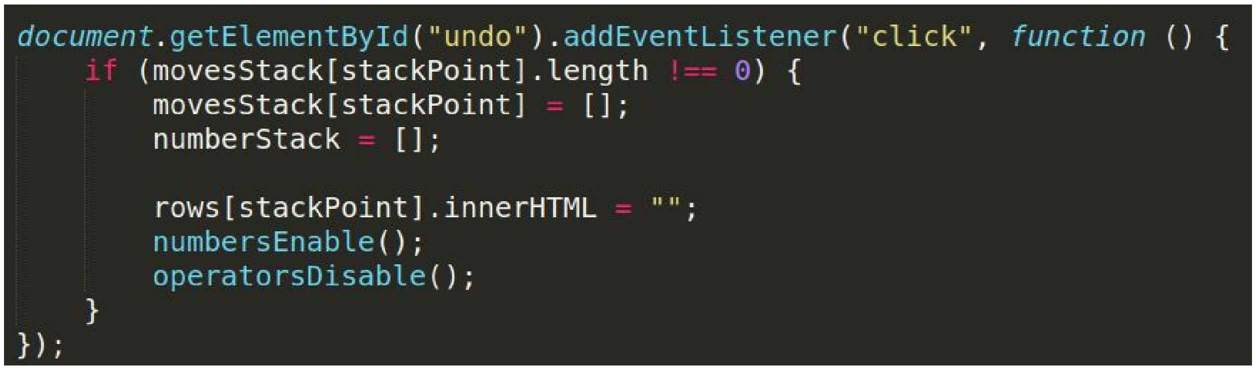
finished in 1.225 seconds

Any one (or all) solutions can be taken.

- Helper - Helps the user with the next step by dynamically analysing the moves the user has made till now and what are the possibilities. A custom implementation of the **solutions** function to handle any number of values can be made to achieve this.

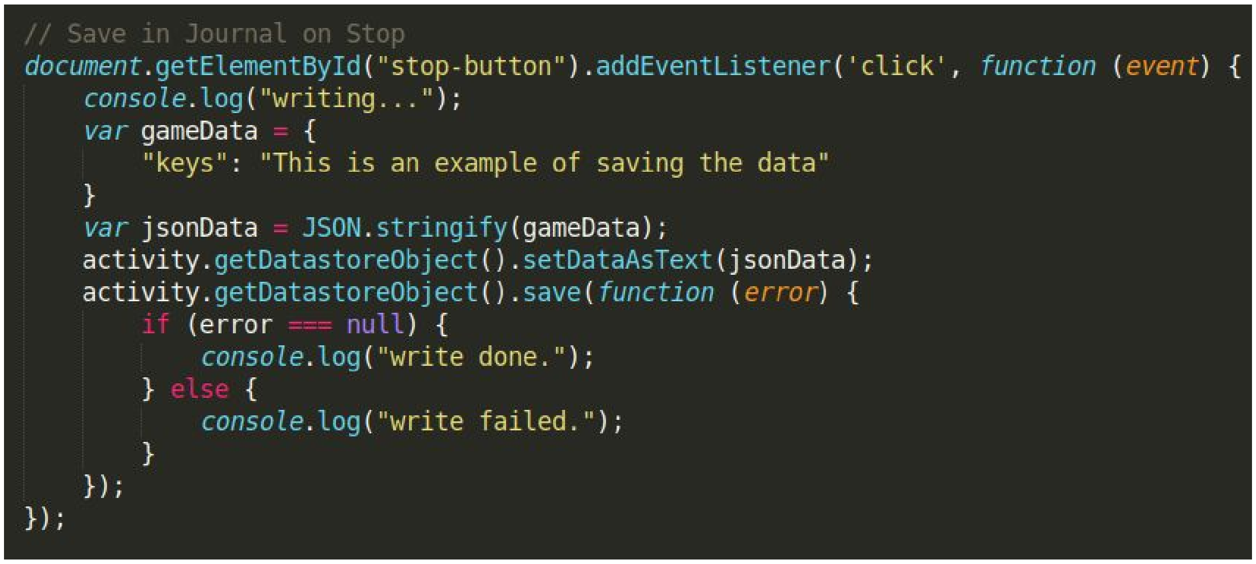
- The game end will give a popup which will show the user his score, the time he took, the number and kind of hints he took, and the bonuses applicable.

- Undo feature will reset the game state to a previous state, using the **movesStack** and the **results** array. A simple implementation of the Undo function is given below:



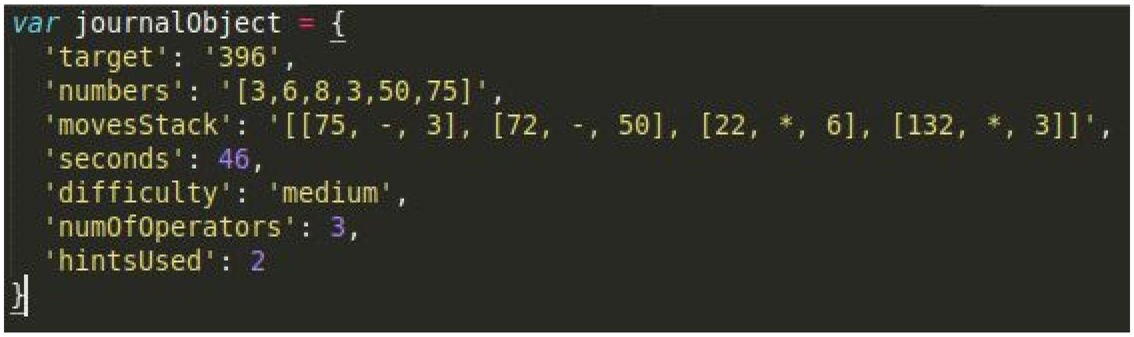
```
document.getElementById("undo").addEventListener("click", function () {
    if (movesStack[stackPoint].length !== 0) {
        movesStack[stackPoint] = [];
        numberStack = [];

        rows[stackPoint].innerHTML = "";
        numbersEnable();
        operatorsDisable();
    }
});
```

- Multiplayer Mode
    - The Presence Framework will be utilised for Network Integration
    - The standard procedure will be used as given in the Official Sugarizer Development Tutorial
    - Leaderboard - We will define the following presence actions:
        - Init - the current user's leaderboard will be synced with the host and a common leaderboard will be visible to all the players in the network (when the instance is shared)
        - Start - the host will select the question and send to all the connected players to sync and start solving
        - End - all the users will send their scores to the common leaderboard
        - Leave - if a user leaves the game, update the leaderboard and the present players lobby (or popup, as in the Memorize Activity)
    - If the host ends the game, we will declare the best player of the rounds played according to the score achieved and will be notified to all the players in the network
    - We will show the current and the high score in the leaderboard, along with the buddy icons, for each user in the network.
- Journal Integration
    - We can use the LZ based compression algorithm for JavaScript to compress the string before storing them in the Journal, taking inspiration from the Exerciser Activity.
    - The state variables will be converted to a JSON object.
    - The JSON object will be stringified and stored in the Journal in the standard manner
    - According to the features, we can discuss and decide as to exactly which variables and data structures should be stored in the object.



```
// Save in Journal on Stop
document.getElementById("stop-button").addEventListener('click', function (event) {
    console.log("writing...");
    var gameData = {
        "keys": "This is an example of saving the data"
    }
    var jsonData = JSON.stringify(gameData);
    activity.getDatastoreObject().setDataAsText(jsonData);
    activity.getDatastoreObject().save(function (error) {
        if (error === null) {
            console.log("write done.");
        } else {
            console.log("write failed.");
        }
    });
});
```

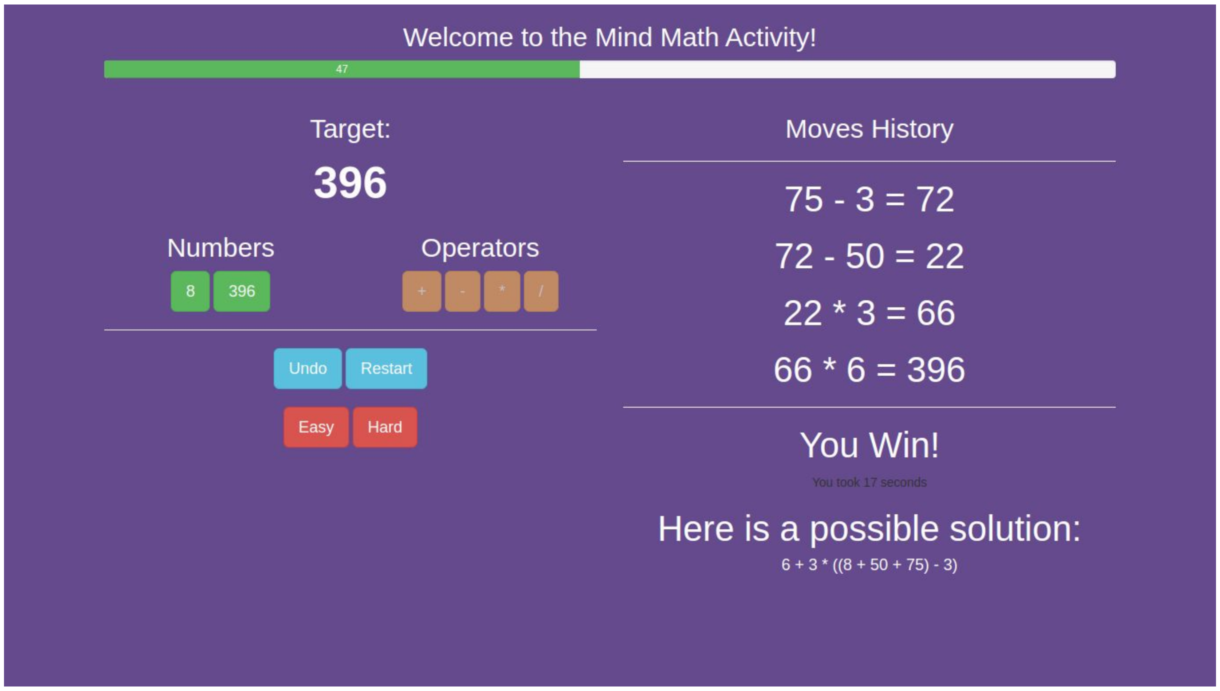
The journal object may look as -



```
var journalObject = {
  'target': '396',
  'numbers': '[3,6,8,3,50,75]',
  'movesStack': '[[75, -, 3], [72, -, 50], [22, *, 6], [132, *, 3]]',
  'seconds': 46,
  'difficulty': 'medium',
  'numOfOperators': 3,
  'hintsUsed': 2
}
```

- Localisation
    - Localisation will be done with the webL10n Library.
- Tutorial
    - We will use Bootstrap 3 Tutorial, as in the majority of Sugarizer activities to make the tutorial interactive
- Responsiveness
    - We will use the Bootstrap 3 Library and the Sugar Web Library for building the UI and taking care of responsiveness on all the screens.
    - The application will be tested to function on all the major Web Browsers (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) and Electron.
- User Interface
    - This will be the basic layout of the UI. I propose to discuss with the mentors on where will be the best to remove text and use figures and geometrical shapes. The UI will be Sugar Friendly and will use Buddy Colors for the Background and for the User Icons. I emphasize on using less text and more figures, in respect of the Sugar Standards.
    - If we use Vue.js as an inline framework:
        - mounted() lifecycle hook can be accessed so that we can access the reactive components after the DOM loads.

- created() lifecycle hook can be accessed in multiplayer mode to do the initial synchronisation of data before loading the DOM, like an API fetch
- v-for directive can be used to render the arrays in the DOM
- v-else and v-if statements can be used for conditional render in the DOM
- watch() property can be used to update the DOM one the data in the game changes in the Logic



75 - 3 = 72
72 - 50 = 22
22 * 3 = 66
66 * 6 = 396

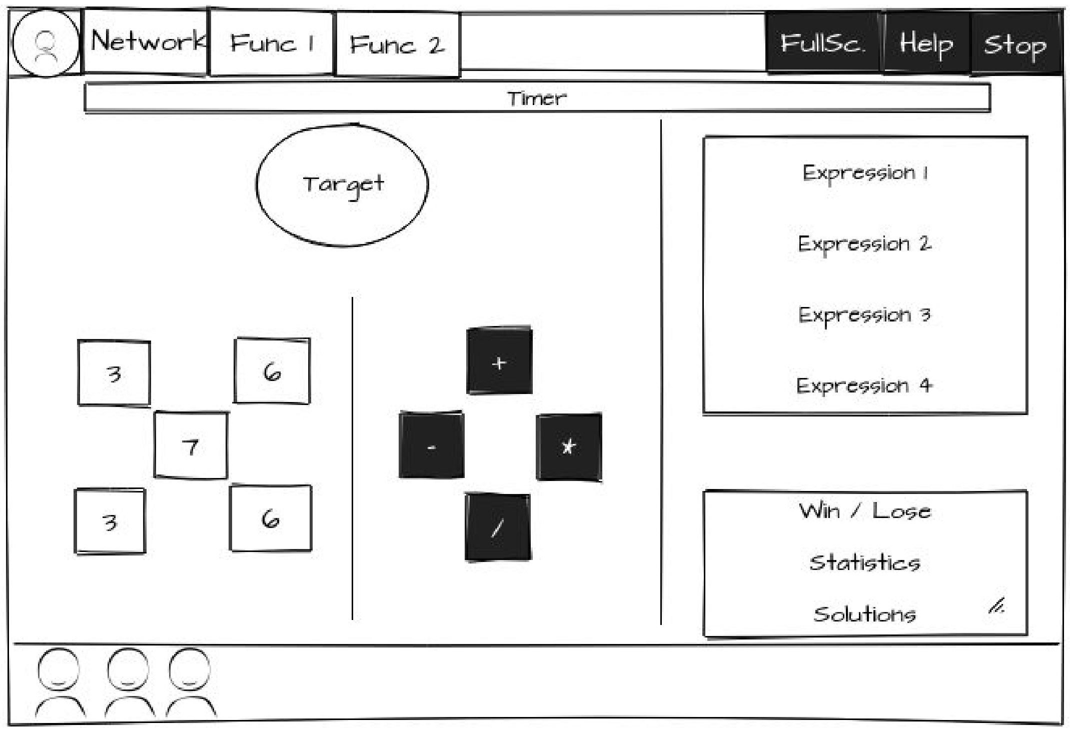
You Win!

You took 17 seconds

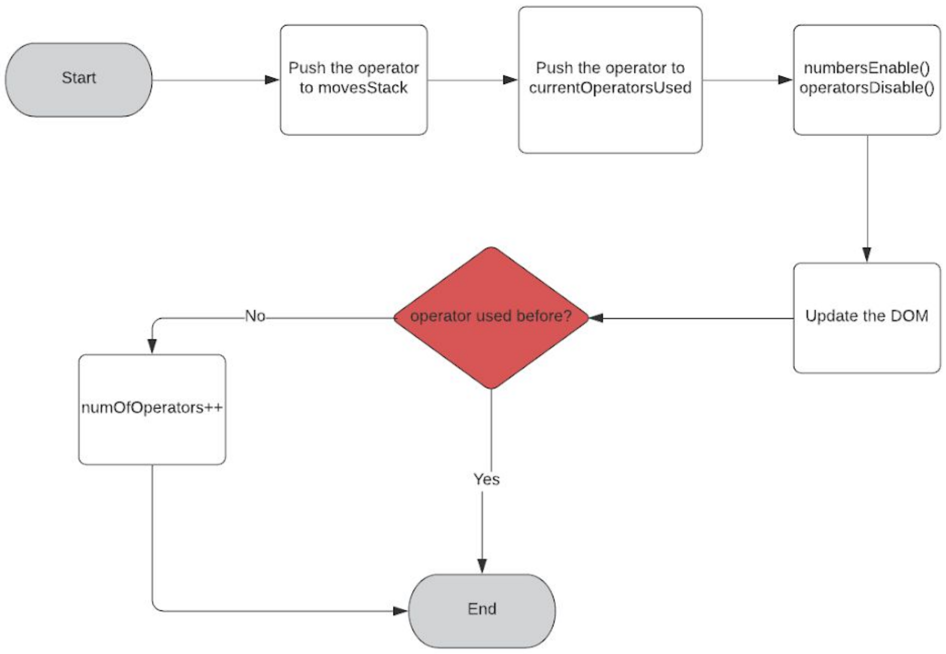
Here is a possible solution:

6 + 3 * ((8 + 50 + 75) - 3)

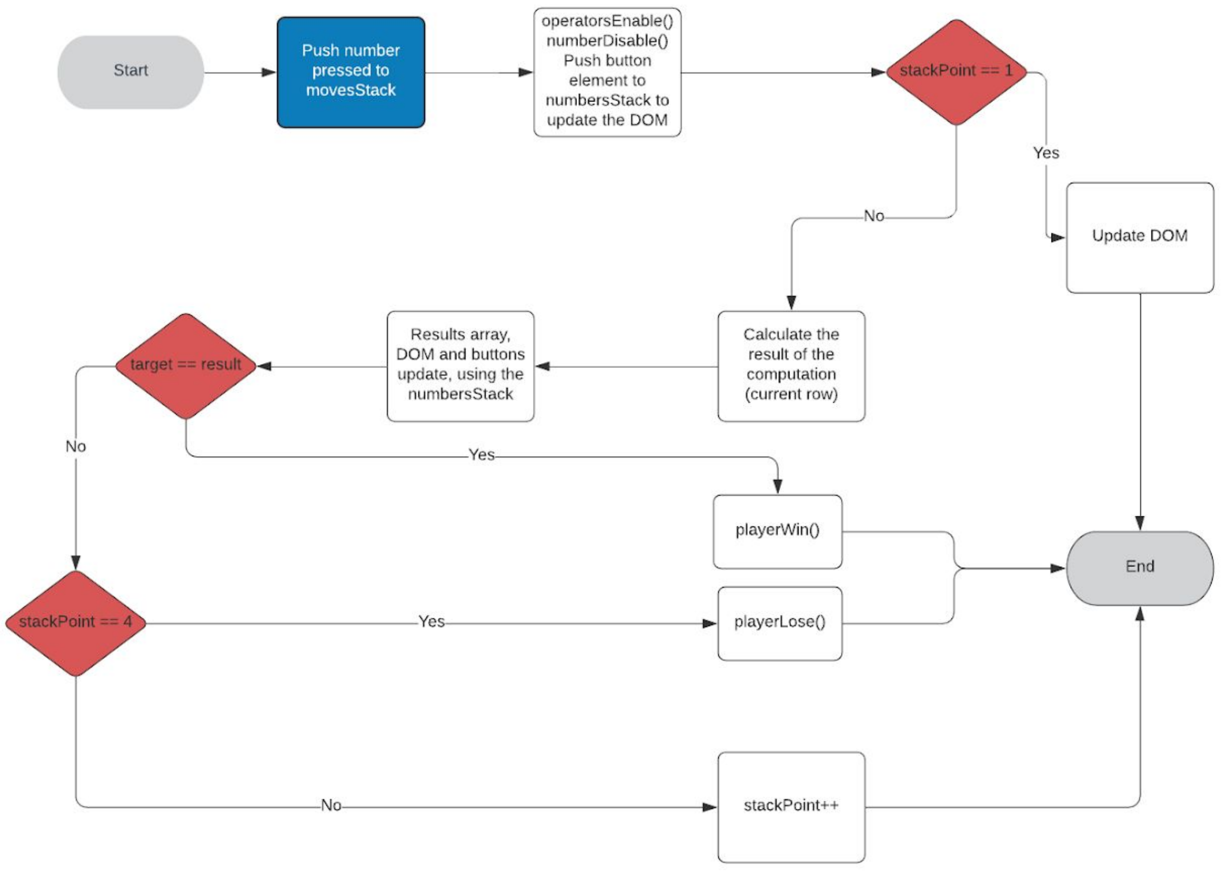
The proposed UI of the Activity will be -



The operatorLogic is given in the following flowchart:



The gameLogic() implementation flowchart is (Note: The logic fires every time the button is clicked):


Start
Push number pressed to movesStack
operatorsEnable() numberDisable() Push button element to numbersStack to update the DOM
stackPoint == 1
Yes
No
Update DOM
Calculate the result of the computation (current row)
Results array, DOM and buttons update, using the numbersStack
target == result
No
Yes
playerWin()
End
stackPoint == 4
Yes
playerLose()
No
stackPoint++

- Additional features (can be implemented in the Future)
    - A story mode in the Activity with slowly increasing levels of difficulty and points for the user who completes the levels.
    - The score for the story mode can be saved locally and mastery score can be made for the user to assess his skills.
    - A progress visualisation dashboard for teachers to assess the students.
    - We can also include simple words in the Activity and the user will have to form the words from the letters.

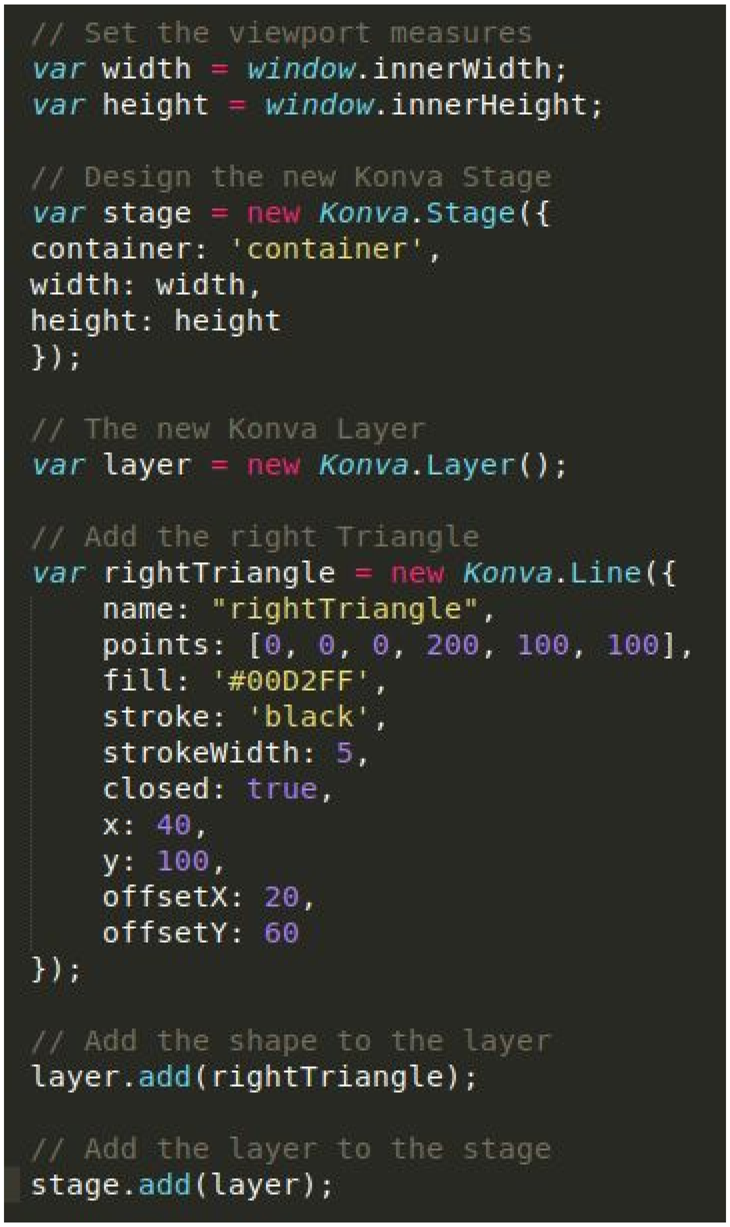
Tangram Activity

The features are discussed in the implementation level:

- We can utilise the following excellent resources for developing the activity:
    - Konva.js - A versatile JavaScript Canvas Library for 2D Desktop and Web Apps. We can use them to create the Tangram Pieces and the dotted lines for the activity.
    - JavaScript Clipper - A clipping and offsetting library in JavaScript. Can be utilised to get the design outlines for the figures.

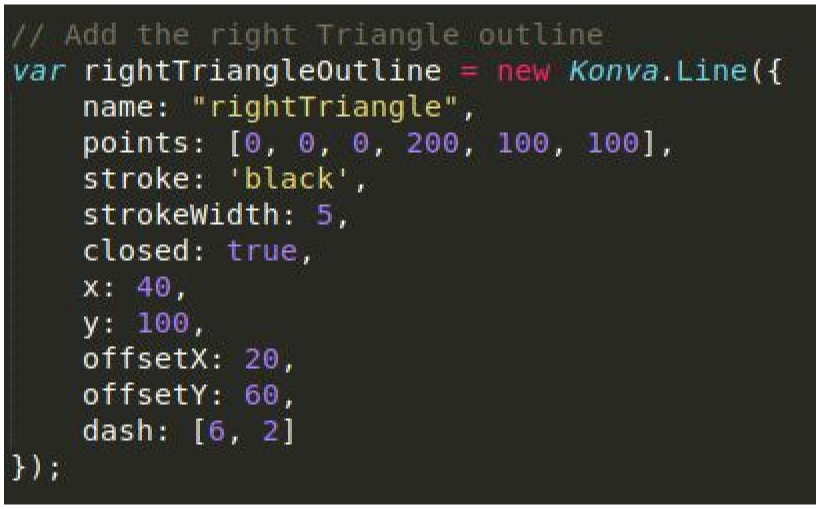
- **Tangram-channel.com** - Lots of pre defined figures, sorted according to the difficulty. Can be used to set the questions.
- **w8r** - A JavaScript Library for Polygon Offsetting and can be utilised in making the outlines of the shape.
- Several Tangram examples available on the internet like here, here and here, which can be used for understanding the pedagogic aspects and help in designing the game flows.

- Tangram Pieces from Konva.js
  - The seven pieces from the basic Tangram activity will be featured. We can create the Polygon in Konva.js to make the shapes.
  - To create a polygon with Konva.js, we can instantiate a **Konva.Line()** object with **closed = true** attribute.
  - These will be draggable pieces, 7 in the pieces array. The pieces will be as discussed. A sample object in Konva.js is given by:



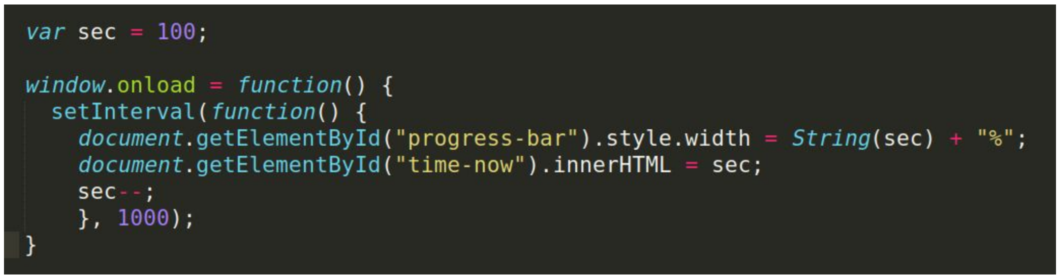
```
// Set the viewport measures
var width = window.innerWidth;
var height = window.innerHeight;

// Design the new Konva Stage
var stage = new Konva.Stage({
container: 'container',
width: width,
height: height
});

// The new Konva Layer
var layer = new Konva.Layer();

// Add the right Triangle
var rightTriangle = new Konva.Line({
    name: "rightTriangle",
    points: [0, 0, 0, 200, 100, 100],
    fill: '#00D2FF',
    stroke: 'black',
    strokeWidth: 5,
    closed: true,
    x: 40,
    y: 100,
    offsetX: 20,
    offsetY: 60
});

// Add the shape to the layer
layer.add(rightTriangle);

// Add the layer to the stage
stage.add(layer);
```

- Shape outlines from Konva.js
  - A similar mechanism can be made to make the outlines. A **outlines** array will store the outlines.
  - **‘x’** and **‘y’** properties will define the position of these outlines in the **Konva Stage**.
  - If we specify only the **‘stroke’** and do not specify the **‘fill’** property, we will essentially get the pieces, but with only the outlines.



```
// Add the right Triangle outline
var rightTriangleOutline = new Konva.Line({
    name: "rightTriangle",
    points: [0, 0, 0, 200, 100, 100],
    stroke: 'black',
    strokeWidth: 5,
    closed: true,
    x: 40,
    y: 100,
    offsetX: 20,
    offsetY: 60,
    dash: [6, 2]
});
```

- Timer Integration - we can count in the logic and update the DOM with the values.The DOM can be updated either by:
    - The **watch** property in Vue.js can listen to the updates in the value of the time count variable and update the DOM.
    - Updating the **<span>** with JS, we can use the **setInterval()** function and configure it to listen to the **window.onload** event and start counting



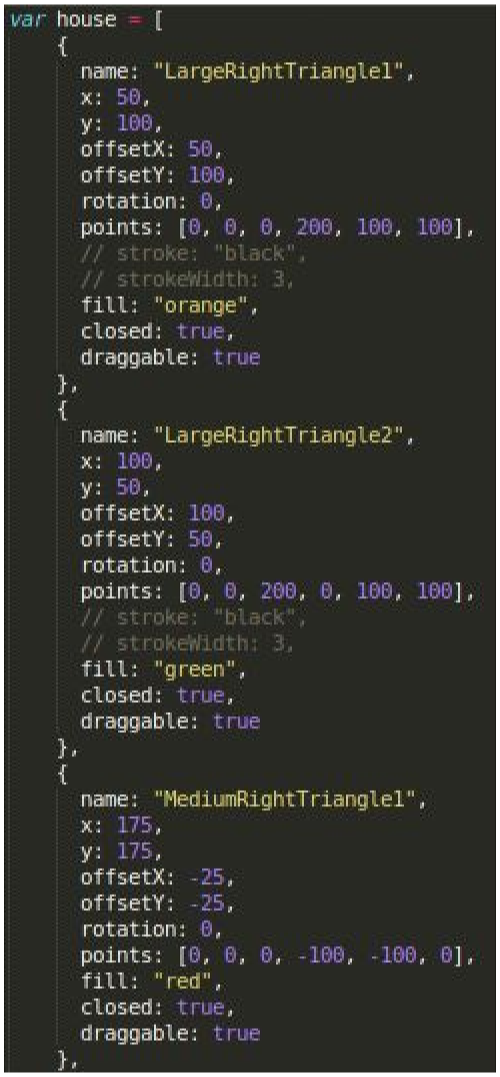
```
var sec = 100;

window.onload = function() {
  setInterval(function() {
    document.getElementById("progress-bar").style.width = String(sec) + "%";
    document.getElementById("time-now").innerHTML = sec;
    sec--;
    }, 1000);
}
```

The player will still be able **to play the game after the timer ends, but extra points will be awarded if there is still time left.**

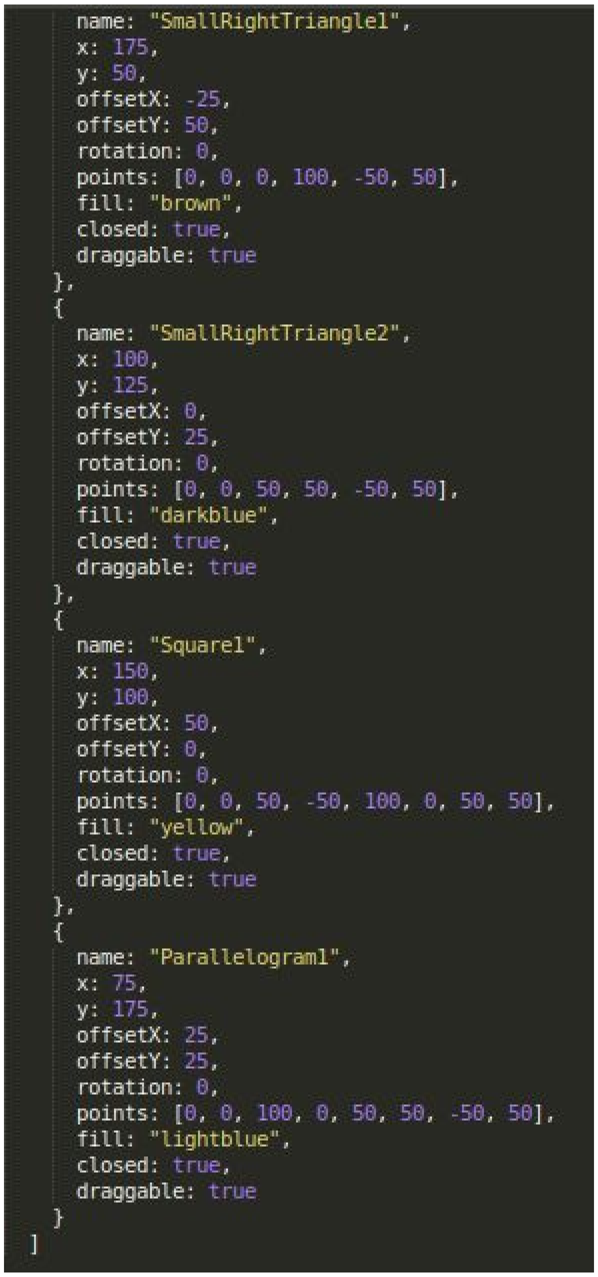
- **Enable/Disable Timer** - The user will be able to disable the timer as well. This will be possible in the single player mode and will help the user take his time to solve a puzzle. The UI will have an area to show the user the time he has spent on the problem. The easy mode will not have the timer by default. The timer can be integrated into the harder difficulty levels.
- Bootstrap 3 Progress Bars can be used to display a dynamic and responsive timer in bar form. We have to routinely change the **style.width** to the new percentage.

- Shape (Filled, Outlines) from Konva.js
    - We have discussed as to how we can form individual shapes. The **shape** array will be an array for 7 such objects, with their relative positions. In other words, a shape is a collection of piece objects.
    - For example, if we have a house shape, it can be defined as given.

- The outline of the design can be made by uncommenting the **stroke** and **strokeWidth** property, and commenting the **fill** property.



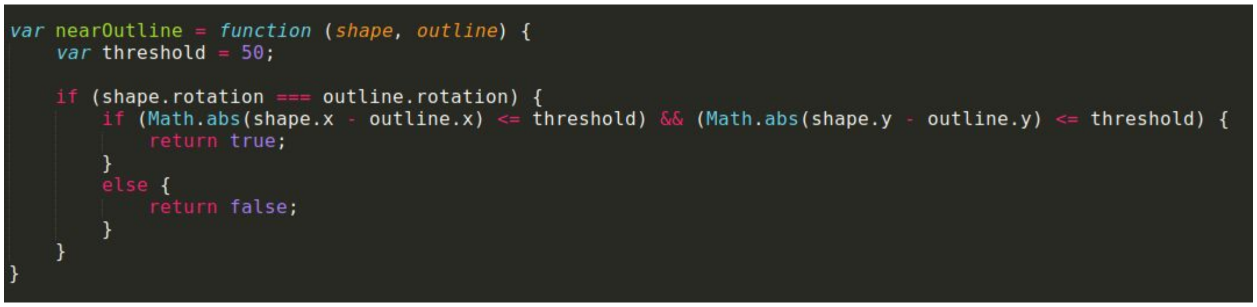
```
var house = [
    {
      name: "LargeRightTriangle1",
      x: 50,
      y: 100,
      offsetX: 50,
      offsetY: 100,
      rotation: 0,
      points: [0, 0, 0, 200, 100, 100],
      // stroke: "black",
      // strokeWidth: 3,
      fill: "orange",
      closed: true,
      draggable: true
    },
    {
      name: "LargeRightTriangle2",
      x: 100,
      y: 50,
      offsetX: 100,
      offsetY: 50,
      rotation: 0,
      points: [0, 0, 200, 0, 100, 100],
      // stroke: "black",
      // strokeWidth: 3,
      fill: "green",
      closed: true,
      draggable: true
    },
    {
      name: "MediumRightTriangle1",
      x: 175,
      y: 175,
      offsetX: -25,
      offsetY: -25,
      rotation: 0,
      points: [0, 0, 0, -100, -100, 0],
      fill: "red",
      closed: true,
      draggable: true
    },
```



```
      name: "SmallRightTriangle1",
      x: 175,
      y: 50,
      offsetX: -25,
      offsetY: 50,
      rotation: 0,
      points: [0, 0, 0, 100, -50, 50],
      fill: "brown",
      closed: true,
      draggable: true
    },
    {
      name: "SmallRightTriangle2",
      x: 100,
      y: 125,
      offsetX: 0,
      offsetY: 25,
      rotation: 0,
      points: [0, 0, 50, 50, -50, 50],
      fill: "darkblue",
      closed: true,
      draggable: true
    },
    {
      name: "Square1",
      x: 150,
      y: 100,
      offsetX: 50,
      offsetY: 0,
      rotation: 0,
      points: [0, 0, 50, -50, 100, 0, 50, 50],
      fill: "yellow",
      closed: true,
      draggable: true
    },
    {
      name: "Parallelogram1",
      x: 75,
      y: 175,
      offsetX: 25,
      offsetY: 25,
      rotation: 0,
      points: [0, 0, 100, 0, 50, 50, -50, 50],
      fill: "lightblue",
      closed: true,
      draggable: true
    }
  ]
```

- Hint Feature - One of the outline of any one piece from the **outlines** array could be made into the shape by uncommenting the **stroke** and **strokeWidth** property, and commenting the **fill** property of any one of the outline. The corresponding piece will be removed from the available pieces. The number of hints used by the user will be added to calculate the score in the **hintsUsed** game state variable.

- Near Outline - When a piece is dropped near the outline, it will be snapped to align with the outline, with respect to a certain threshold value. This can be developed by calling the **nearOutline()** function as given below, everytime the x and y coordinates of the piece change in the dragging.



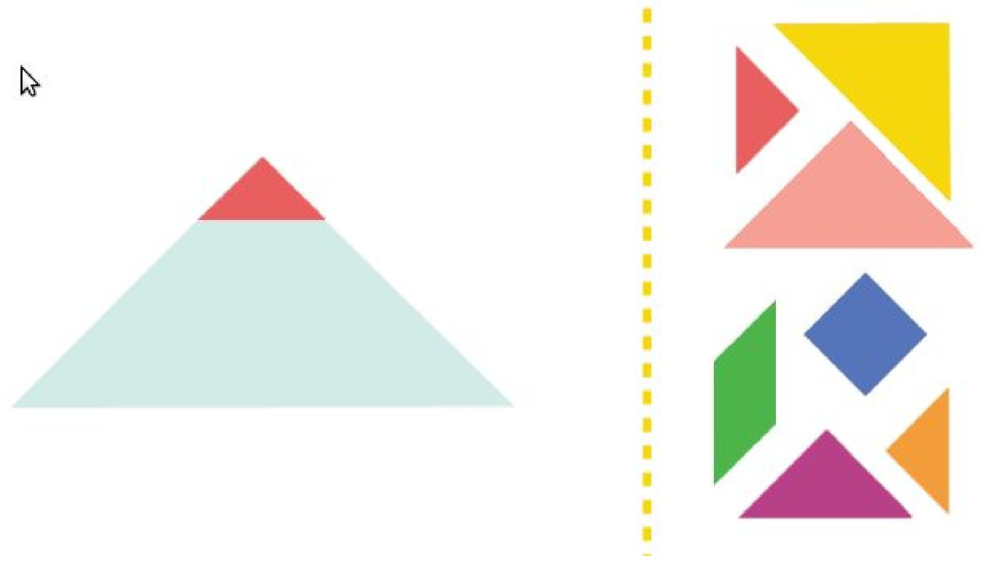
```javascript
var nearOutline = function (shape, outline) {
    var threshold = 50;

    if (shape.rotation === outline.rotation) {
        if (Math.abs(shape.x - outline.x) <= threshold) && (Math.abs(shape.y - outline.y) <= threshold) {
            return true;
        }
        else {
            return false;
        }
    }
}
```

- Undo Functionality - the following variables need to be stored:
  - x and y coordinates of the piece, before and after each drag and drop.
  - rotation of each of the pieces before and after the rotation by the player.

  The data will be stored in the **history** array, which will be initialised when the window loads, or the **mounted()** lifecycle hook is accessed in Vue.js.

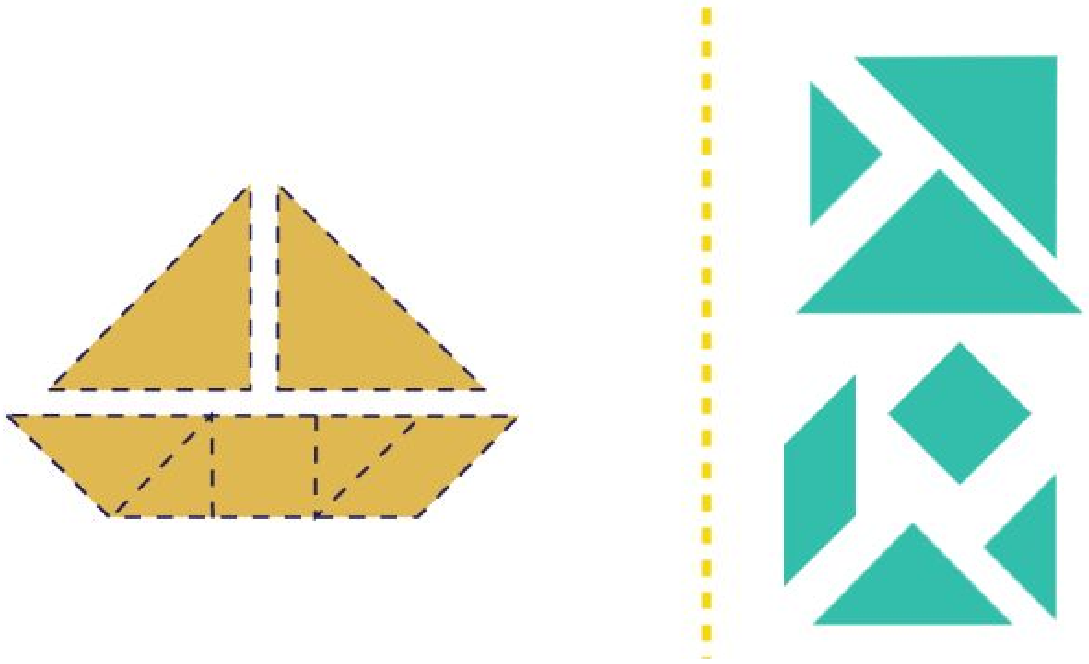
- Game State
  - The game state will be stored in:
    - pieces [] - stores information about the 7 pieces of a shape
    - outline [] - stores information about the outlines of the shape
    - history [] - stores the data of the moves played till now
    - Seconds - the time elapsed from the start of the game
    - Difficulty - the current difficulty level of the game by the user
    - piecesPlacedCorrectly - the number of pieces correctly placed
  - More state variables will be discussed and decided with the mentor to load and save in the database.

- Difficulty Levels
  - Easy Level -

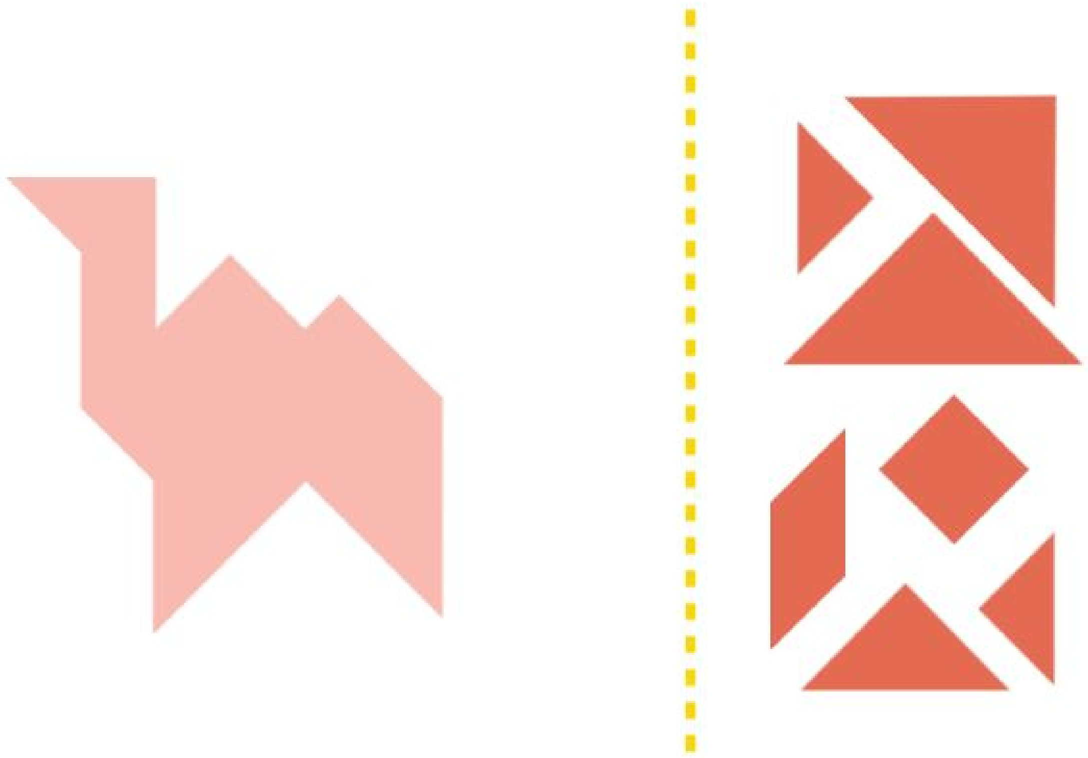


    - In the easy mode, we can put only one piece at a time in the highlighted area.

        - The **outlines** array will give a random piece's outline object (by the **Math.floor()** and **Math.random()** function), and we will highlight the area enclosed by the outline by the **fill** property of the outline object.
        - We will wait till the user correctly rotates and puts the correct piece in the highlighted area, and in the meanwhile we will block the selection of any other piece.
        - When the user places the piece correctly, the highlighted area takes the color of the original piece, and the next piece is highlighted.
    - Medium Mode -



        - In the medium mode, all the outlines, and only the outlines will be visible for the complete shape.
        - The user can take and rotate any piece he wishes to in any order.
        - The piece, if correctly placed, will change the color to signify the player that he is correct.
        - The **piecesPlacedCorrectly** variable is incremented, and will be utilised to calculate the score
    - Hard Mode -



        - In the hard mode, only the outline to the complete shape will be given to the user.
        - We can utilise the **JavaScript Clipper** Library to get the outline of the shapes.
        - All the timer variables and the game state variables will update as in the Medium Mode.

In all the difficulty levels, special attention will be given to effectively color code the outlines and the pieces, so as to provide the players visual feedback and build their skills.

- Multiplayer Mode and Leaderboard
    - We know that the Presence Framework is centralised on the server, and not the host, so we can define the points -
    - When the user shares the instance (isHost == true)
        - The host chooses the difficulty level and the shape to build
        - The host sets everything and waits for other player to join the game
        - As soon as players join, the host can press **start** to start the timer and the puzzle will be visible to everyone
        - If the host leaves the game, every other players' games are automatically stopped and the final result is declared as per the leaderboard/all the other players keep the game playing. (to be discussed with the mentor)
    - When the user joins a game (isHost == false)
        - The player waits till the host starts the game

        - When the host starts the shape is visible to all the players, and everyone can start building
        - The leaderboard will be updated whenever the current score of any of the player changes
        - If the player leaves the game, this is shown as a popup to all the players

  On every correct drag and drop by every player on the network, his score will be calculated and it will be reflected to every player's leaderboard. The player join and player leave will be shown in the Lobby and with popup, as in the **Memorize Activity**.

- Outline Design Modes - We can incorporate a game mode in which the user will be able to design outlines of shapes, in a way to generate more puzzles (questions) for other users on the network.

This could be designed by allowing the **draggable** property of the outline object, and then saving the set of 7 outlines in the database. One more possible implementation could be to let the user build any shape he likes, in a free draw kind of mode, and when he thinks he has completed a custom shape, we can take the outline of the custom shape using the **JavaScript Clipper** Library and save it in the database.

- Scoring - The score can be calculated by taking into account the following variables:
    - Difficulty - 1/2/3, by the current level of the game
    - Seconds - Time taken to complete the game
    - hintsUsed - The number of hints used by the user

Score = (Difficulty / (hintsUsed * Seconds)) * 100

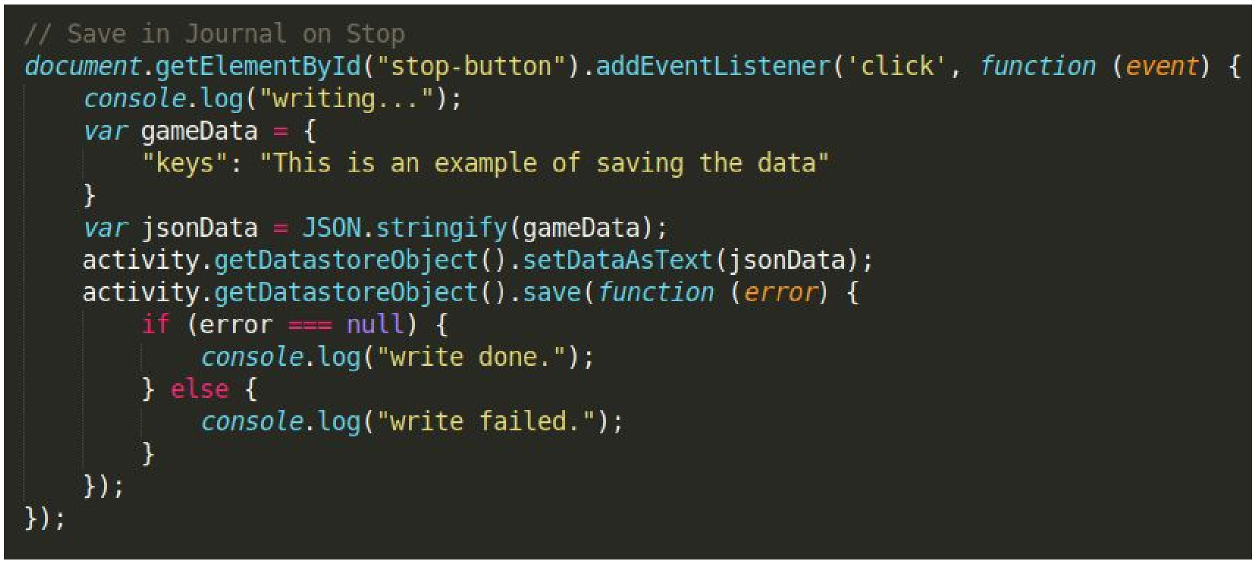
The actual expression for calculating the score will be best discussed with the mentor.

- The game end will give a popup which will show the user his score, the time he took, the number and kind of hints he took, and the bonuses applicable.

Most of the standard features(Journal Integration, localisation, tutorial, responsiveness, UI) are the same as discussed in the previous activity.

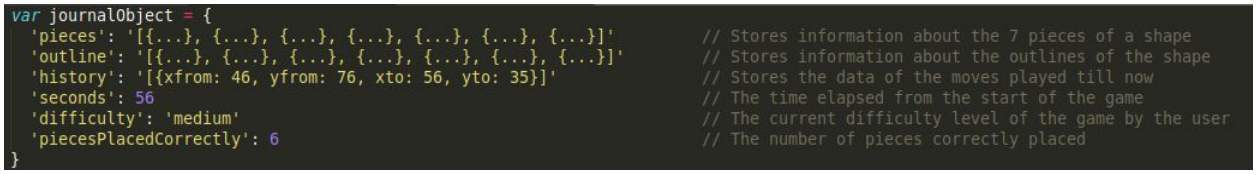
- Multiplayer mode can be implemented by the **Presence Framework**
    - We will define the following presence actions tentatively:
        - Init - the current user's leaderboard will be synced with the host and a common leaderboard will be visible to all the players in the network (when the instance is shared)
        - setDifficulty - the host will select the difficulty level and the shape (question) and send to all the players in the network, but the timer and the shape will not be visible
        - Start - the host will start the timer and the question will be visible to everyone in the network
        - updateScore - the users score will be updated in all the leaderboards.

        - Finish - the user will send his score if he finishes the game and the leaderboard will be updated
        - Leave - if a user leaves the game, update the leaderboard and the present players lobby (or popup, as in the **Memorize Activity**)
- Journal Integration
    - We can use the LZ based compression algorithm for JavaScript to compress the string before storing them in the Journal, taking inspiration from the Exerciser Activity.
    - The current game variables of game state will be taken as the data.
    - Custom shapes designed can also be stored in the journal.
    - The **outlines** of the shapes will be given as the data payload.
    - The state variables will be converted to a JSON object.
    - The JSON object will be stringified and stored in the Journal in the standard manner
    - According to the features, we can discuss and decide as to exactly which variables and data structures should be stored in the object.



```
// Save in Journal on Stop
document.getElementById("stop-button").addEventListener('click', function (event) {
    console.log("writing...");
    var gameData = {
        "keys": "This is an example of saving the data"
    }
    var jsonData = JSON.stringify(gameData);
    activity.getDatastoreObject().setDataAsText(jsonData);
    activity.getDatastoreObject().save(function (error) {
        if (error === null) {
            console.log("write done.");
        } else {
            console.log("write failed.");
        }
    });
});
```

The journal object may look as -

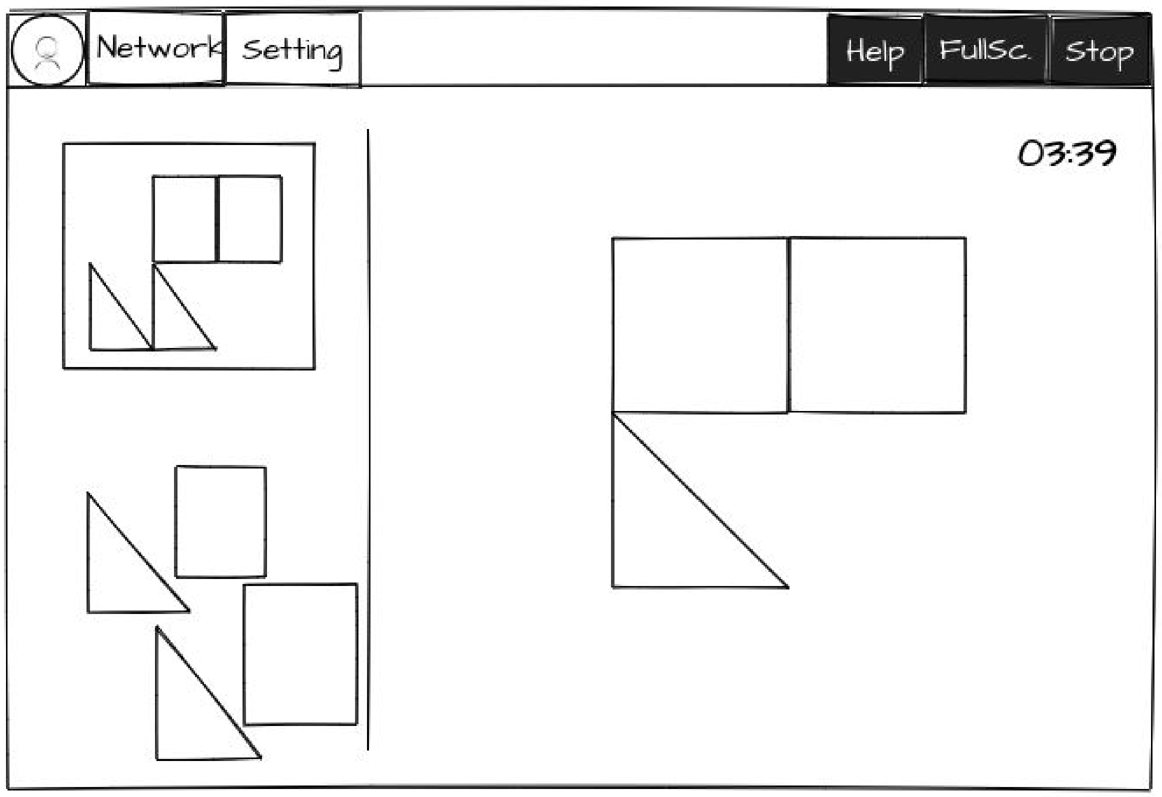


```
var journalObject = {
  'pieces': '[{...}, {...}, {...}, {...}, {...}, {...}, {...}]'        // Stores information about the 7 pieces of a shape
  'outline': '[{...}, {...}, {...}, {...}, {...}, {...}, {...}]'       // Stores information about the outlines of the shape
  'history': '[{xfrom: 46, yfrom: 76, xto: 56, yto: 35}]'              // Stores the data of the moves played till now
  'seconds': 56                                                        // The time elapsed from the start of the game
  'difficulty': 'medium'                                               // The current difficulty level of the game by the user
  'piecesPlacedCorrectly': 6                                           // The number of pieces correctly placed
}
```

- Localisation
    - Localisation will be done with the webL10n Library.
- Tutorial
    - We will use Bootstrap 3 Tutorial, as in the majority of Sugarizer activities to make the tutorial interactive
- Responsiveness
    - We will use the Bootstrap 3 Library and the Sugar Web Library for building the UI and taking care of responsiveness on all the screens.

- The application will be tested to function on all the major Web Browsers (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) and Electron.

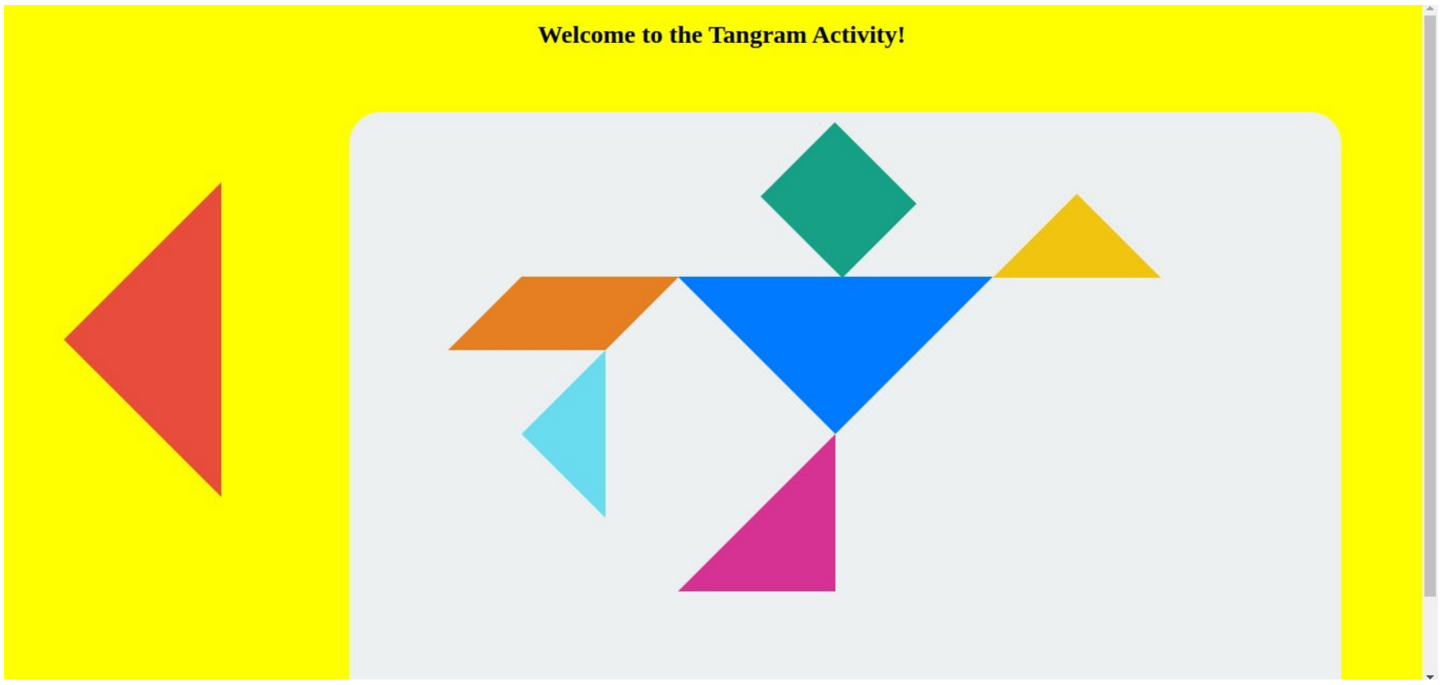
- User Interface



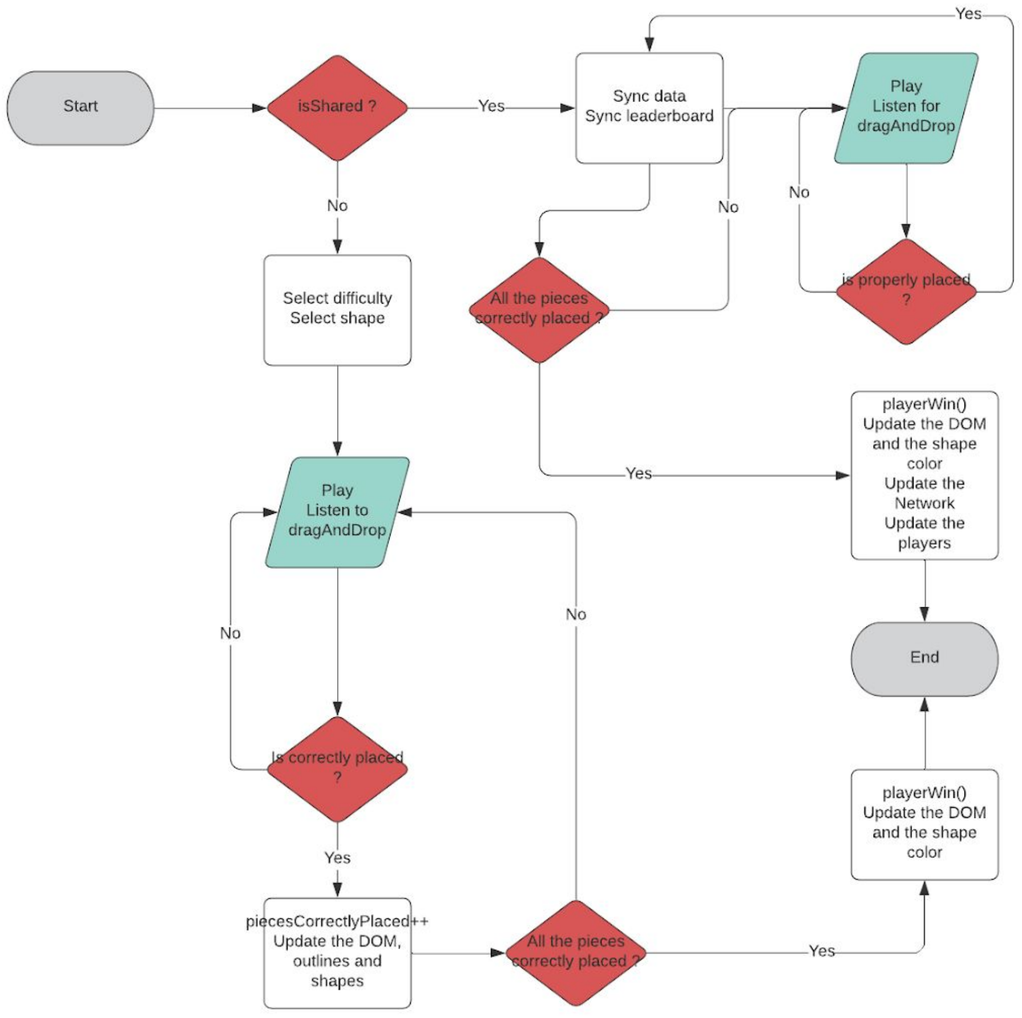
- This will be the basic layout of the UI. I propose to discuss with the mentors on where will be the best to remove text and use figures and geometrical shapes. The UI will be Sugar Friendly and will use Buddy Colors for the Background and for the User Icons. I emphasize on using less text and more figures, in respect of the Sugar Standards.
- If we use Vue.js as an inline framework:
  - mounted() lifecycle hook can be accessed so that we can access the reactive components after the DOM loads.
  - created() lifecycle hook can be accessed in multiplayer mode to do the initial synchronisation of data before loading the DOM, like an API fetch
  - v-for directive can be used to render the arrays in the DOM
  - v-else and v-if statements can be used for conditional render in the DOM
  - watch() property can be used to update the DOM one the data in the game changes in the Logic
- Additional features (can be implemented in the Future)
  - A story mode in the Activity with slowly increasing levels of difficulty and points for the user who completes the levels.

- The score for the story mode can be saved locally and mastery score can be made for the user to assess his skills.
- A progress visualisation dashboard for teachers to assess the students.

The basic UI of the prototype is **(will change according to the requirements, very basic version to test the Libraries and the Functions)**



The basic Game Logic is depicted in the FlowChart



Project Timeline

The timeline incorporates all the considerations I can take into account at the time of developing, including the COVID-19 pandemic. My semester examination will be tentatively finished by the 10th of May, so I will put in extra hours to cover up for the deviation. I have no other projects in the Summer and can devote **40-50** hours per week (approximately 6-7 hours per day). The timeline is very flexible, and the exact dates can be decided with discussion with the mentors.

Since the activities are complete games, I would like to work in the **Incremental** manner, developing the basic prototype first and then enhancing it with new features.

I have made the project into modules, and hereby explain what are the deliverables for each week -

Community Bonding Period (4 May to 1 June) -

- Discuss with the mentor and the members of the community about the use cases, technologies to be utilised and the best practices to incorporate.
- Discuss with the mentors about coding practices and how to exactly implement the features. Try to decide upon the game state variables and other aspects according to the use cases and the requirements.
- Start Blogs about the project and update regularly.

Week 1 (June 1 - June 7) -

- Develop the basic UI, integrate Bootstrap 3 for responsiveness
- Making of the difficulty levels
- Game state variables finalisation, Journal Integration

Week 2 (June 8 - June 14) -

- Timer integration
- Make the hints feature (solver)
- Make the hints feature (helper)

Week 3 (June 15 - June 21) -

- Restart, Undo features
- Presence Integration
- Leaderboard development

Week 4 (June 22 - June 28) -

- Score feature development
- Player Lobby and popup on join / leave of the players
- Testing on different Web Browsers and in the Electron
- UI improvement, Matching up to the Sugar Standards, Buddy Colors Integration

Evaluation 1 (June 29 - July 3) -

- The fully functional prototype of the MindMath Activity
- Includes all the gameplay features, enhancements and multiplayer mode.
- Blog updates
- Learn more about Konva.js and polygon offsetting (for the Tangram Activity development)

Week 5 (July 4 - July 10) -

- Localisation (MindMath Activity)
- Tutorial in the Bootstrap Tour, along with the documentation (MindMath Activity)
- Vue-js integration, wherever possible and more efficient that the JavaScript way (MindMath Activity)

- Developing the basic UI for the Tangram Activity and taking care of responsiveness in all the screen sizes

Week 6 (July 11 - July 17) -

- Pieces building using Konva.js
- Outlines building using Konva.js
- Drag and drop and rotation for all the pieces and outlines
- Journal integration and game state variables finalisation

Week 7 (July 18 - July 24) -

- Find shapes for all the three difficulty levels from the internet and make outlines
- Easy mode building
- Medium mode building

2 Days before Evaluation 2 (July 25 - July 26) -

- UI improvement, with Vue.js integration wherever possible and when more efficient than the JS way
- Extra time for error and bugs

Evaluation 2 (July 27 - July 31) -

- The MindMath Activity is fully completed
- The Tangram Activity basic functionality is completed
- Easy and Medium Modes functional
- Shapes and their outlines from the internet are saved
- Update the blogs

Week 8 (August 1 - August 7) -

- Timer integration
- Hint feature
- Near outline snapping
- Undo feature
- Score feature

Week 9 (August 8 - August 14) -

- Presence Integration
- Leaderboard Development
- Hard mode building and learning more about the JavaScript Clipper Library for getting the outlines of the shapes

Week 10 (August 15 - August 21) -

- Tutorial and documentation
- Localisation
- Custom shapes creations and sharing with the network
- Extra time for error and bugs

2 Days before the Final Evaluation (August 22 - August 23) -

- Testing on different Web Browsers and on the Electron
- Extra time for unforeseen work and errors.

Final Evaluation (August 24 - August 31) -

- Completed both the activities
- The documentation and tutorial is finished
- The blogs are updated.

How will it impact Sugarlabs?

The MindMath activity is of immense importance. It is a really engaging way to learn about mathematical operations. The fact is evident that in the United Kingdom, there was a TV Show for the same game, and they used to play with 6 numbers and the four basic operators. I believe that addition of this activity to Sugarizer will provide the students a fun way to learn. I myself used to like this game a lot and I am very excited to be able to develop the game for students, if I am selected.

The Tangram is a very celebrated Chinese Game, and of immense value for visual development. Lot of academic work has been carried out on the game and it is very popular all over the world. The students can draw shapes and share them with their friends over the network. The visual imaginative capacity is greatly improved which will be good for other areas in life. The addition of this activity to Sugarizer in my opinion will enhance the already rich set of activities and will help expand the horizons for Sugarizer in the pursuit of providing world class pedagogic activities for students.

My plans post GSoC 2020

I will be maintaining the activities and fixing the error and bugs regularly, and will keep discussing to further improve the activities. Since the activities are game based, we might require frequent feature changes and updates. I will be available all round the year for the issues related to the activities and all the Sugarizer activities.

I have been contributing and engaging in conversations and discussion in Sugarlabs and Sugarizer for more than a year. I will continue to do my best to be an active member of the community. I will love to welcome new members to the community and mentor when possible in the upcoming events. I would like to take this opportunity to thank Sugarlabs for being such a supportive place for my Open Source Journey, and all the admins, mentors and contributors, who have been wonderful in making the organisation a constructive and beautiful place to work and keep learning and developing for the students all over the world.