

DATA STRUCTURES AND ALGORITHMS

CSE 2003

Slot - L3+L4

Project supervisor – Arun Kumar G

MONEY TRANSFER GAME

BY

M SAMPREETH – 19BCE0865

RITVIK GUPTA – 19BCE0397

Email Id:

Sampreeth – sampreethm.2019@vitstudent.ac.in

Ritvik - ritvik.gupta2019@vitstudent.ac.in

ABSTRACT

The project Money Transfer Game is a website game where a users can play online or offline. This project was developed in Java Script. The project has a backend and frontend interface. In backend we have used **NodeJS**, Express and in frontend **Paper JS, CSS, Semantic UI** .The Data base used to save user scores is **MongoDB** .

In the data base we store the graph properties for a specific graph such as adjacency table, values of nodes and there positions. Along with that it also stores the best possible path for a given graph. To calculate the best possible path we have used DSA **Graph Structures, Adjacency list , Adjacency table**.

The main algorithms used in the game are as follows:

- 1) **Depth First Search (DFS)**
- 2) **Greedy Algorithm**
- 3) **Recursive Backtracking**

For the validation of the any given graph the DFS and money .Transfer game algorithms are used. For calculating the best possible path for a given graph we have used recursive backtracking and greedy algorithm along with money transfer game graph structure.

PROBLEM STATEMENT

Aim:

To create a user-friendly website where people can play the game and see the previous scores and there reports.

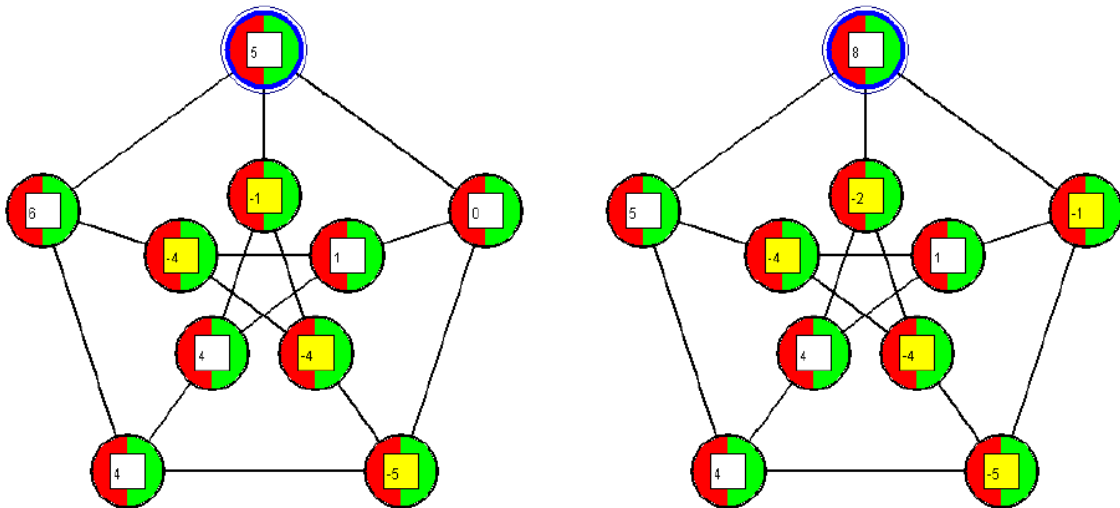
Objective:

To create a game which can store scores and make playing games in faster way to run smoothly.

INTRODUCTION

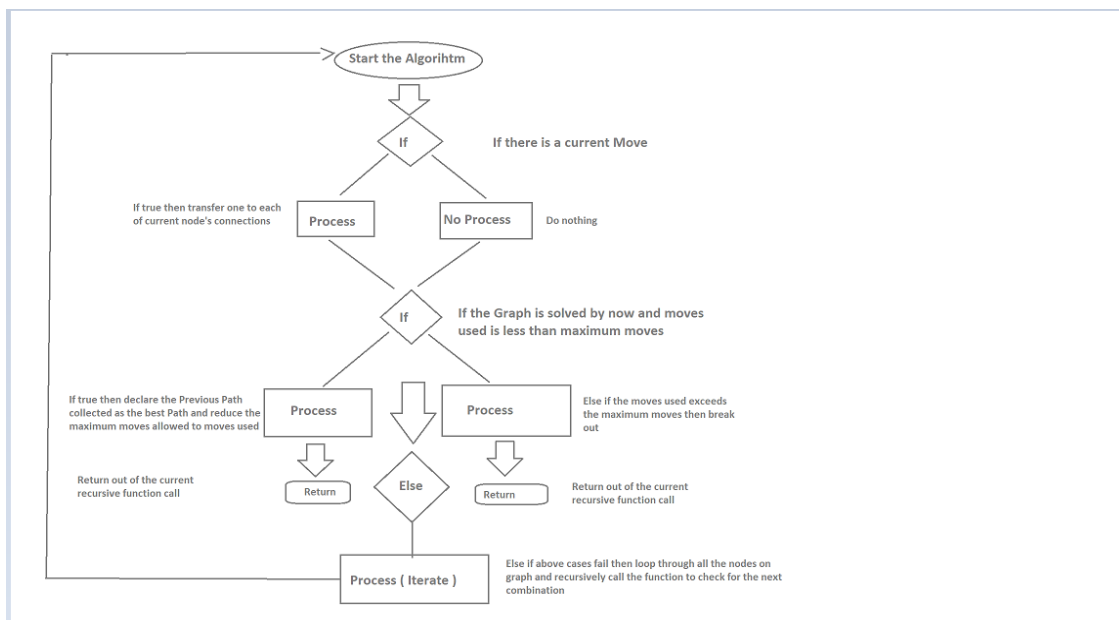
The Money Transfer Game is a game of solitaire played on a graph; the underlying mathematical principles are unexpectedly rich and beautiful . To play the game, select a graph G as the game board.¹ Then select an initial configuration of dollars on G . (A configuration is an assignment of some integer number of dollars – which could be positive, negative, or zero – to each vertex of G .) A vertex with a negative number of dollars is said to be in debt. There are two kinds of legal moves in the game. A lending move consists of selecting a vertex v and then moving one dollar across each edge adjacent to v in such a way that the money flows from v to its neighbors. A borrowing move is similar, except that the money flows in the other direction. For example, here is an illustration of a borrowing move performed on the famous “Petersen graph”. The picture on the left shows the starting configuration, and

the picture on the right shows the new configuration after the top vertex performs a borrowing move:



The goal of the dollar game is to get all the vertices of G out of debt by a sequence of legal moves.

PROPOSED SYSTEM ARCHITECTURE



The above flowchart describes how the Best Path Calculating Algorithm works.

As mentioned above the conditions need to be met by a Graph for it to be playable and solvable. Using the above conditions we can verify that a graph is valid or not and if a graph is valid it goes on to perform the MTG Algorithm Process to calculate the Best Path Solution for the given Graph. The Algorithm and Flowchart for the Verification of a Graph and the Best Path Solution Algorithm are given.

PROJECT PLAN

The Money Transfer Game Project is based on a famous game known as Dollar Game. The Game consists of a Graph with n number of vertices and the initial conditions are as follows :

Give each vertex of the graph a number...The goal is to make sure that none of the vertices have negative numbers anymore...and the moves you can make in the game are that either a person can donate money (meaning for each vertex that they're connected to, they can give a dollar), or a person can take money (meaning for each vertex that they're connected to, they can take a dollar). And, the goal of the game is to make sure that everyone is out of debt.

The Graph has to be shown to the User who is playing the Game and this is done using the Javascript Library called **Paper.js** which is a vector graphics library for web development. The Page for a Graph displays the Nodes of a sheet of **paper** and each Node has some connections to one or more of the other Nodes. Thus each Node on the screen is an object which

has certain properties and an array of all the connections or neighbors of that node.

The game is winnable in certain cases, and the ultimate question is: How few moves can you win the game in?

This is the main motive of the Money Transfer Game is to find the Best Possible Path for a given Graph Structure using **Greedy Algorithm and Recursive Backtracking**. When a User creates a new Graph a request is sent to the backend server where the Graph is validated and added to the Database. The request contains the Graph structure using **Adjacency Table** which is used to store all the neighbors of each and every node in a 2D table format.

For the Database **MongoDB** is used which is a Non-Sequential Database and stores the data in terms of JSON or Javascript Objects. A Graph Structure is created using **Object Oriented Programming** in Javascript and each node of a Graph is an object with certain properties. The Graph as a whole is also an object with a collection of all the nodes of that Graph and has method functions to verify and calculate the Best Possible Path for the Graph. For Verification of a Graph the **DFS** algorithm is used to check if the graph is a connected graph or not. A disconnected graph is considered as an invalid graph for the game.

CONDITIONS FOR VERIFICATION OF A GRAPH

- 1) If the game is winnable, then it can always be won using only borrowing moves. More precisely, any winnable game can be won by

simply using the “borrowing binge strategy”: any time there are vertices in debt, pick one of them and do a borrowing move. Repeat until everyone is out of debt!

- 2) There is a number associated to any graph G , called the Euler number, which plays a key role in analyzing whether or not the game is winnable. The Euler number g of a graph is defined as $\#edges - \#vertices + 1$. Let N denote the total number of dollars in the game at any time.

- If N is at least g , then the game is always winnable.
- If N is less than g , then the game may or may not be winnable.

For example, in the Petersen graph game depicted on page 1, we have $g = N = 6$ so we conclude that the game depicted there is winnable.

- 3) If two configurations are equivalent (meaning that one can get from one to the other by a sequence of legal moves), then they obviously have the same degree (the total number of dollars in the configuration). The converse is false: there are usually many inequivalent configurations of the same degree. This can be quantified in the following surprising way: For any fixed integer N , the maximal number of inequivalent configurations of degree N is equal to the number of spanning trees² in G . (This is closely related to Kirchhoff’s celebrated “Matrix-Tree Theorem”.) Indeed, if one fixes a reference vertex v together with an ordering of the edges of G , it is

possible to give an explicit one-to-one correspondence between equivalence classes of degree N configurations and spanning trees in G .

- 4) Fix an arbitrary labeling v_1, \dots, v_n of the vertices of G . Define the associated order configuration M by

$$M(v_i) = (\text{\#edges connecting } v_i \text{ to a vertex } v_j \text{ with } j > i) - 1.$$

Then:

- 1) $\deg(M) = g - 1$ (i.e., the total number of dollars in M is $g - 1$).
 - 2) The game with initial configuration M is not winnable.
- 5) If D is a configuration of degree $g - 1$, then the game with initial configuration D is winnable if and only if the game with initial configuration $K - D$ is winnable, where K is the canonical configuration given by

$$K(v) = (\text{\#edges adjacent to } v) - 2.$$

This can be deduced from (?) using the fact that the order configurations associated to a vertex ordering and its reverse ordering always sum to K .

IMPLEMENTATION DETAILS

a. Validation of a Graph :

A Graph needs to be verified and validated to be deemed as a valid graph for playing the game on. If the Graph is verified then the system will go on to calculate the Best Possible Path for that Graph, but if a Graph is deemed invalid by the function then an error message is displayed explaining to the User why the Graph he entered is invalid and what needs to be done to fix it

PSEUDOCODE:

```
BEGIN
```

```
CREATE graph as an object in memory for a NodeGraph(AdjacencyTable, Values)
```

```
CALL display method of graph
```

```
CREATE data and CALL the validate method of graph
```

```
IF data is valid {
```

```
    CALL calcPath method of graph
```

```
    IF bestPath property of graph has length > 0 {
```

```
        RETURN Object({ success is true,
```

```
            path is bestPath property of graph,
```

```
            message is Success message of graph  })
```

```
    } ELSE{
```

```
        RETURN Object({ success is false,
```

```
            path is NONE or null,
```

```
            message is Faliure messages of graph  })
```

```
    }
```

```

} ELSE {

    RETURN Object({ success is false,

                    path is NONE or null,

                    message is Faliure messages of graph })

}

END

```

b. Verification of Graph :

As a Graph has multiple nodes and each node has multiple properties it is essential to first verify a graph so that it is playable and it is not impossible to win on it. The function is a member function of the class **NodeGraph** and thus shares its properties.

PSEUDOCODE:

```

START

CREATE message and INITIALIZE as an empty array []

CREATE valid and INITIALIZE as true

CREATE edges, i, j

FOR i in RANGE 0 to Number of Nodes {

    FOR j in range 0 to i {

        IF node i in table is connected to node j

            INCREMENT edges by 1

    }

}

CREATE g and INITIALIZE as #edges - #vertices +1

CREATE N and INITIALIZE as Sum of all values of nodes

IF EVERY node has value >= 0 {

```

```

    PUSH ("All Values > 0. Easiest Game.") in message

    CHANGE valid to false

}

IF  $N < 0$  OR  $g$  {

    PUSH ("Sum of Values < Graph Number or 0. Unsolvable Game.") in message

    CHANGE valid to false

}

Call DFS Algorithm method of the class

IF length of DFS List < Number of Nodes {

PUSH ("Disconnected Graph. Invalid Game.") in message

    CHANGE valid to false

}

CREATE M and INITIALIZE as an empty array []

FOR EACH node at index i in table {

     $M[i] = (\text{node connections connecting } v[i] \text{ to a vertex } v[j] \text{ with } j > i) - 1$ 

}

IF M is similar to values of graph {

    PUSH ("Fails Riemann-Roch theorem. Impossible Game.") in message

    CHANGE valid to false

}

IF valid is true

    PUSH ("Normal Game.") in message

RETURN Object({ message and valid})

END

```

c. Best Path Searching Algorithm:

When a Graph is validated by the program it calls the Best Path Algorithm function. The function utilizes the **Geedy Algorithm** along with **Recursive Backtracking** to calculate the Best Path that suits the path. A Best Path is the one which takes the least time to finish or finishes in the least number of clicks. This is useful for Calculating the least possible outcome of the Graph and thus can be used to accurately describe the model of the Graph and the moves allowed to the player for winning on a particular Graph

PSEUDOCODE:

START

IF currentNode EXISTS OR is NOT null {

 FOR EACH el at index i in connections OF currentNode {

 IF el > 0 {

 DECREMENT value of currentNode

 INCREMENT value of connection of currentNode

 }

 }

 PUSH (name of currentNode) in previousMoves

 INCREMENT movesCount

}

IF EVERY value of nodes in graph ≥ 0 and movesCount < maxMoves allowed {

 ASSIGN previousMoves as the bestPath

 ASSIGN movesCount as the maxMoves

```

        RETURN null
    }

    IF movesCount exceeds maxMoves allowed

        RETURN null

    FOR EACH value at index i in values of nodes in graph {

        IF value > 0

            CALL RECURSIVELY with currentNode as i-th node

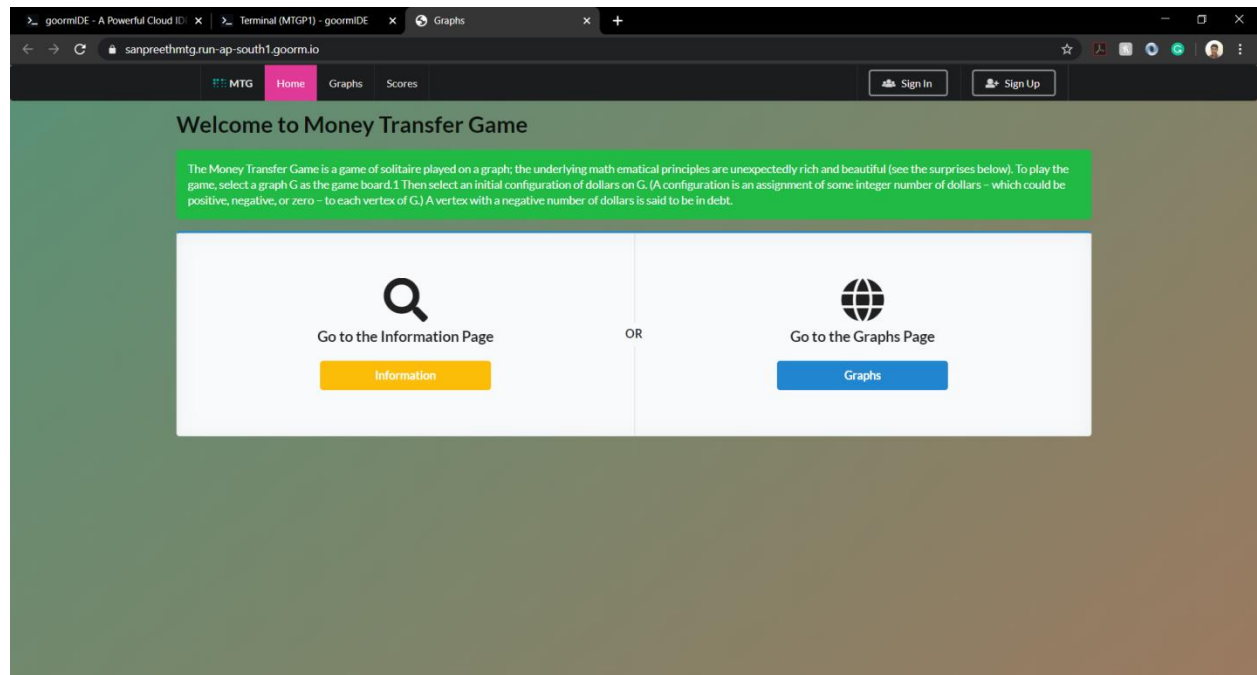
    }

END

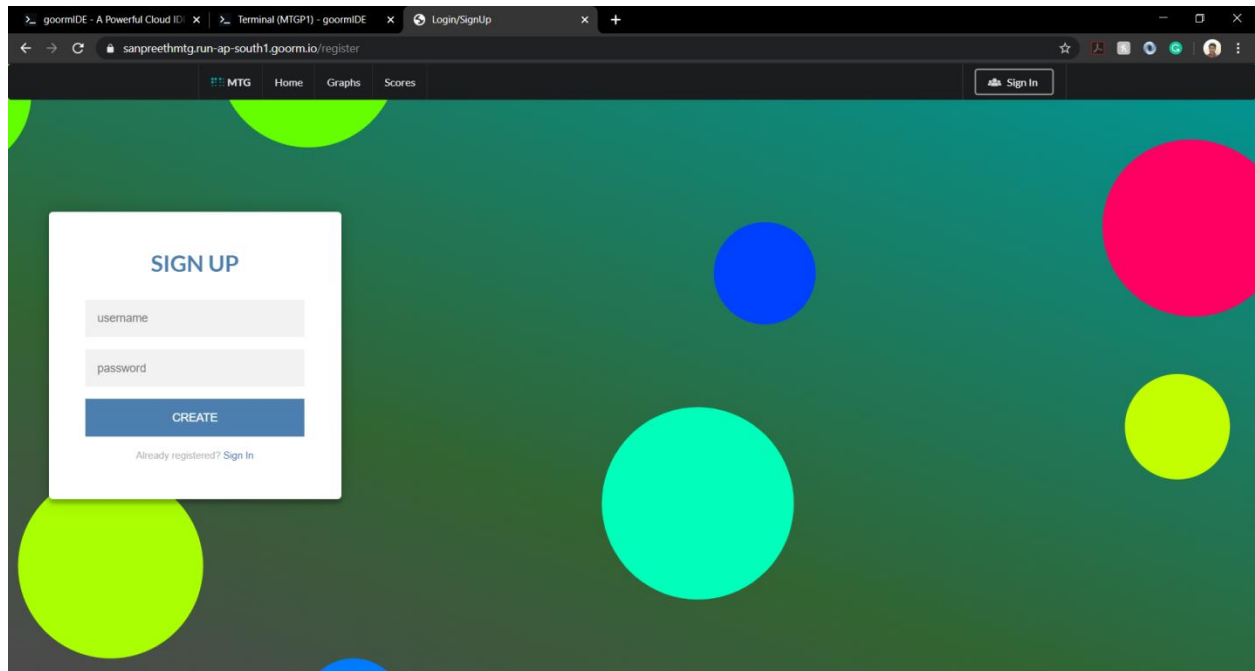
```

TESTING AND SAMPLE OUTPUTS

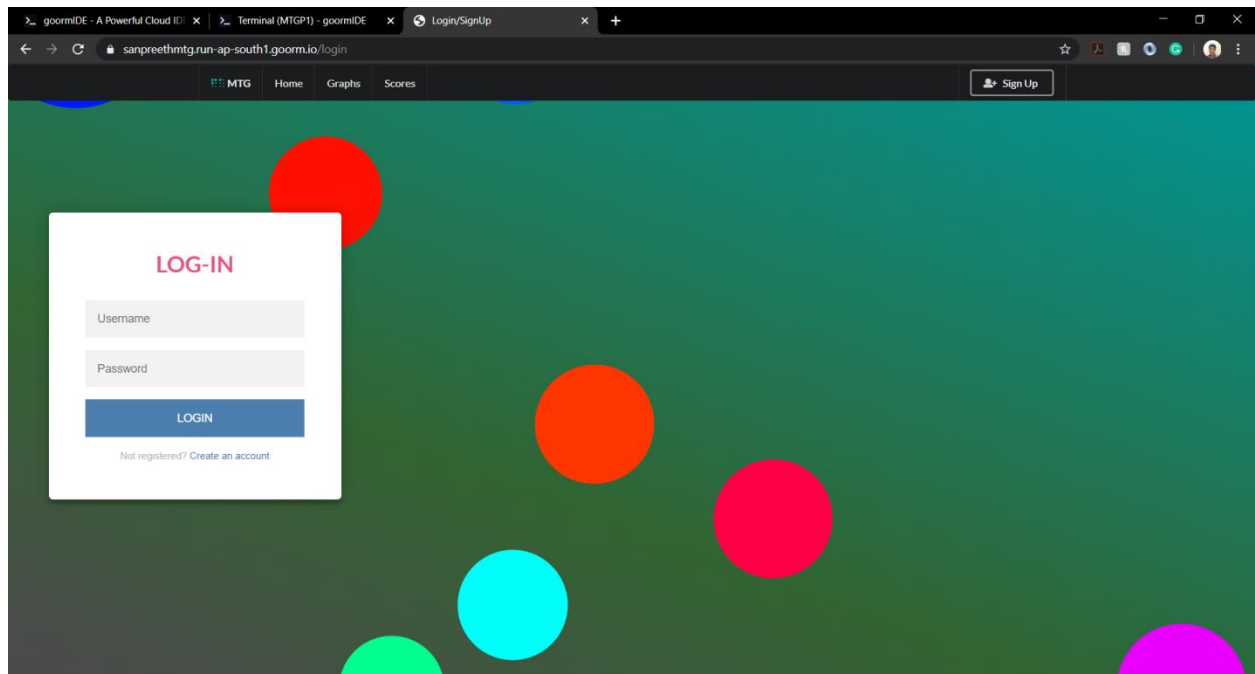
MAIN PAGE AFTER STARTING PROGRAM



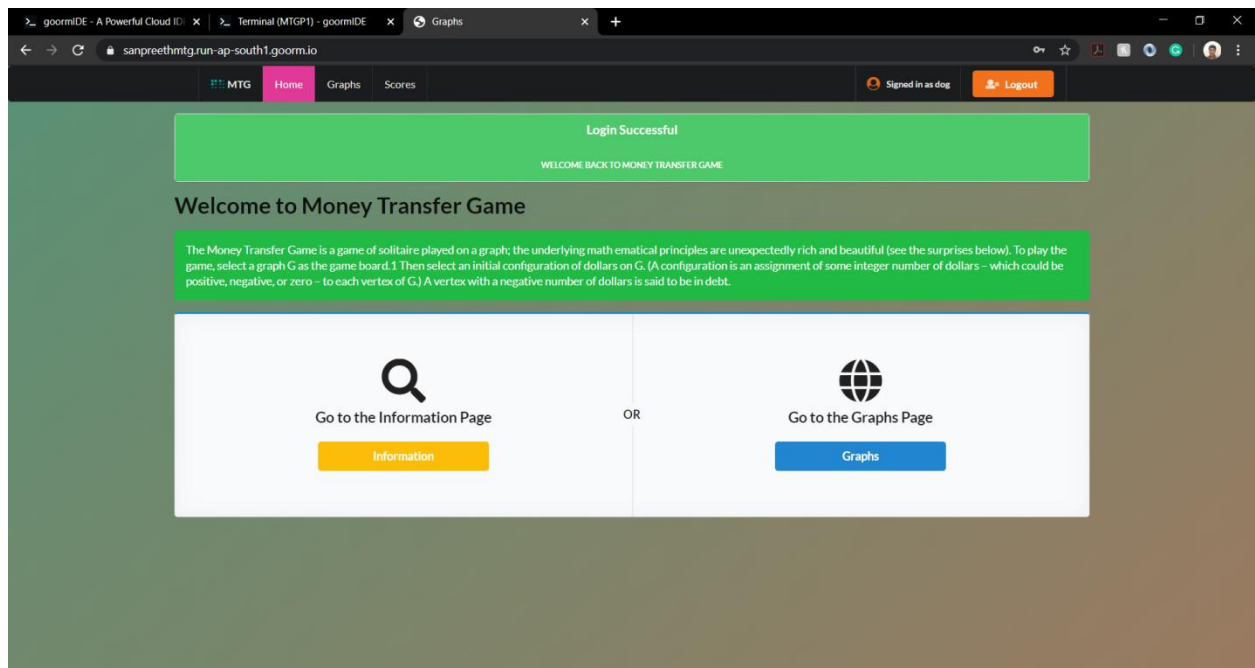
CREATION OF AN ACCOUNT



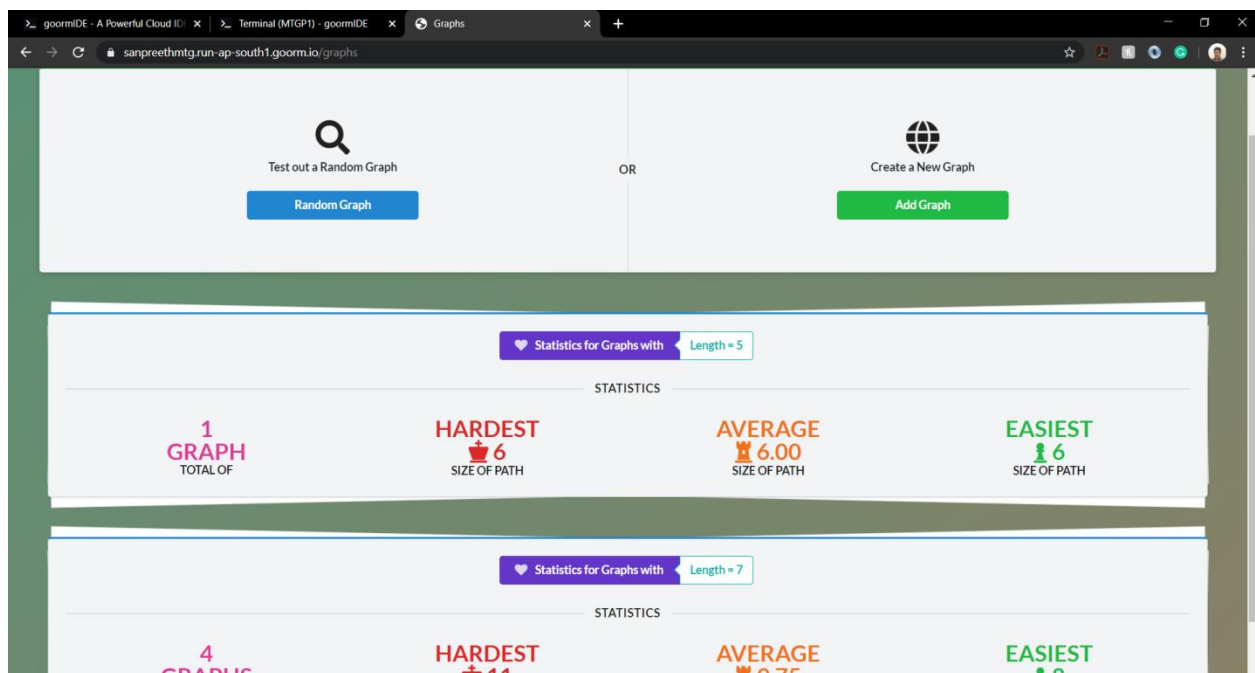
LOGIN BY THE CREDENTIALS



MESSAGE DISPLAY AFTER SUCCESSFUL LOGIN



There are two options of playing. The user can create his/her own game or play from existed game from database.



Creation of graph by user:

If user wants to create the graph, the user must select the number of nodes. For each nodes user must select the neighbours to which the nodes what to connect. Next Step is to assign values to each nodes. If for given values if game is not possible then it can not be added to data-base and displays the type of error in window. If for the given values if graph is possible then graph is stored in data base and will be showed in the user account only.

In Database the graph is stored in **Adjacency Table** format along with the **Vertices** and the **Positions** of all the nodes on the graph . Also the **Best Possible Path** is stored in database.

Selection of Nodes and there neighbor nodes

The screenshot shows a web browser window with the URL `sanpreethmtg.run-ap-south1.goorm.io/graphs/new`. The interface has a dark theme. At the top, there is a form with a text input labeled "Number" and a red error message "Enter the number of Nodes". Below the input is a teal "Submit" button. A navigation bar contains three icons: a blue grid, a mail icon, and a book icon. The main content area features a "Table" tab and a grid of nodes labeled "Node A" through "Node E". Each node has a dropdown menu and a set of colored buttons (orange, green, grey) representing potential neighbors. The buttons are arranged in a 5x5 grid, with the diagonal elements being grey and the off-diagonal elements being orange or green.

	Node A	Node B	Node C	Node D	Node E
Node A	Grey	Orange	Green	Orange	Orange
Node B	Orange	Grey	Green	Orange	Orange
Node C	Green	Green	Grey	Orange	Green
Node D	Orange	Orange	Orange	Grey	Green
Node E	Orange	Orange	Green	Green	Grey

Selecting values and if not works with the values error displays

Number Enter the number of Nodes

Submit

A is	<input type="text" value="-4"/>		<input type="button" value="C is"/> <input type="button" value="E is"/>
B is	<input type="text" value="1"/>		<input type="button" value="D is"/> <input type="button" value="E is"/>
C is	<input type="text" value="-5"/>		<input type="button" value="A is"/> <input type="button" value="E is"/>
D is	<input type="text" value="1"/>		<input type="button" value="B is"/>
E is	<input type="text" value="2"/>		<input type="button" value="A is"/> <input type="button" value="B is"/> <input type="button" value="C is"/>

Test Cases Failed

- *Sum of Values < Graph Number or 0. Unsolvable Game.*

If values are correct then it is added in data base

Number Enter the number of Nodes

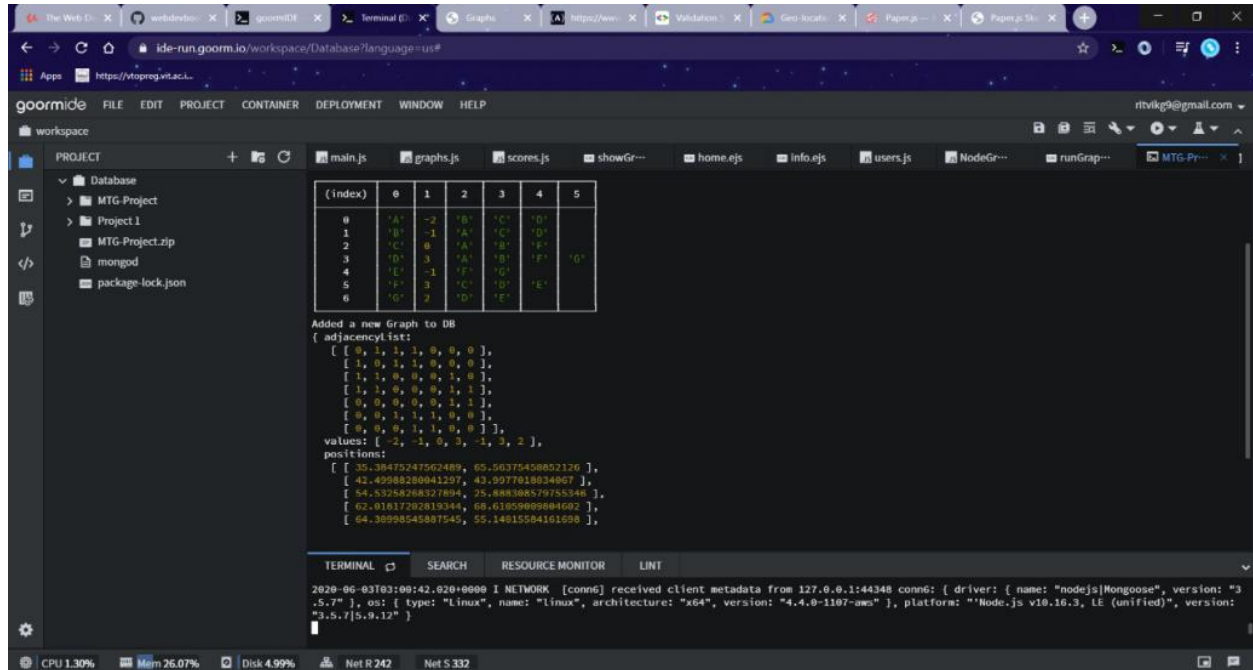
Submit

A is	<input type="text" value="-2"/>		<input type="button" value="C is"/>
B is	<input type="text" value="-1"/>		<input type="button" value="C is"/>
C is	<input type="text" value="2"/>		<input type="button" value="A is"/> <input type="button" value="B is"/> <input type="button" value="E is"/>
D is	<input type="text" value="5"/>		<input type="button" value="E is"/>
E is	<input type="text" value="-2"/>		<input type="button" value="C is"/> <input type="button" value="D is"/>

Test Cases Passed

Normal Game.

Example of how a graph is stored in the Database



The screenshot shows the goormide IDE interface. The left sidebar displays a project structure with folders like 'Database', 'MTG-Project', and 'Project 1'. The main editor area shows a file named 'main.js' containing a graph representation. The graph is an adjacency list with 7 nodes (0-6) and their connections. The graph is stored in a database as an adjacency list.

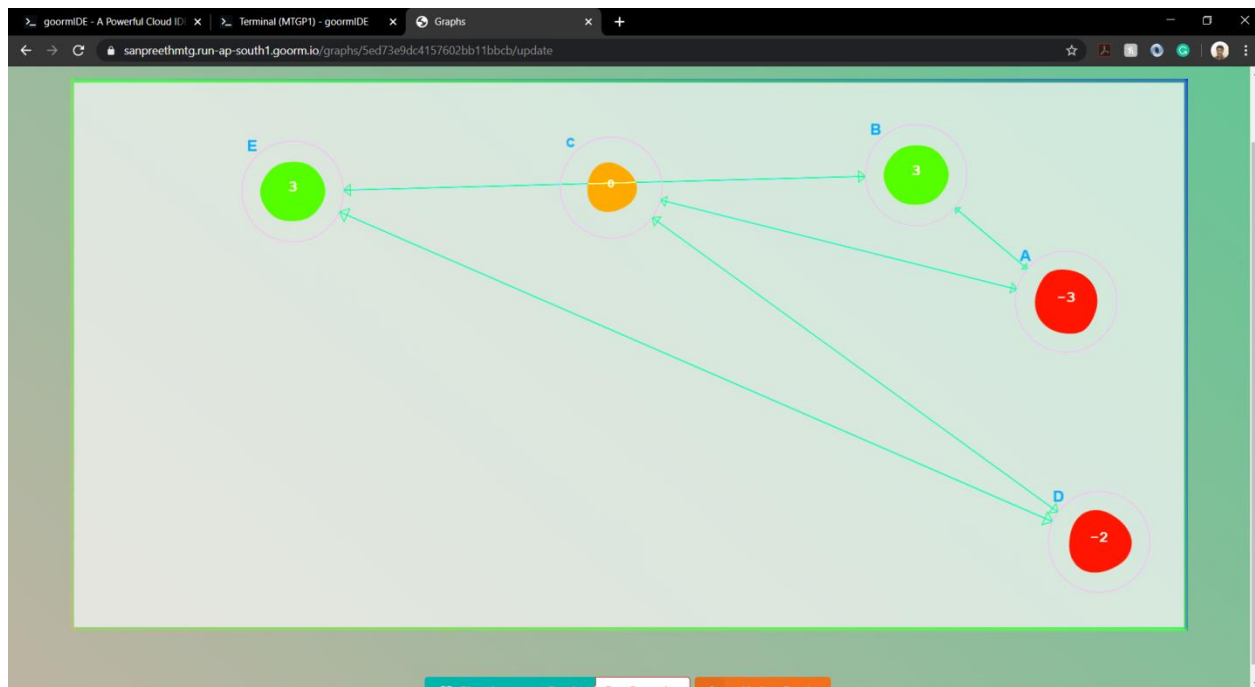
(index)	0	1	2	3	4	5
0	'A'	-2	'B'	'C'	'D'	
1	'B'	-1	'A'	'C'	'D'	
2	'C'	0	'A'	'B'	'E'	
3	'D'	3	'A'	'B'	'E'	'G'
4	'E'	-1	'F'	'D'	'E'	
5	'F'	3	'C'	'D'	'E'	
6	'G'	2	'D'	'E'		

Added a new Graph to DB
{ adjacencylist:
[[0, 1, 1, 1, 0, 0, 0],
[1, 0, 1, 1, 0, 0, 0],
[1, 1, 0, 0, 0, 1, 0],
[1, 1, 0, 0, 0, 1, 1],
[0, 0, 0, 0, 0, 1, 2],
[0, 0, 1, 1, 0, 0, 0],
[0, 0, 0, 1, 1, 0, 0]],
values: [-2, -1, 0, 3, -1, 3, 2],
positions:
[[35.398472247562489, 65.56375459852126],
[42.49988280941297, 42.9077018624067],
[54.53258268327894, 25.883388579755348],
[62.01817202819344, 68.6105989884682],
[64.38998545887545, 55.14815584161698],
[64.38998545887545, 55.14815584161698]] }

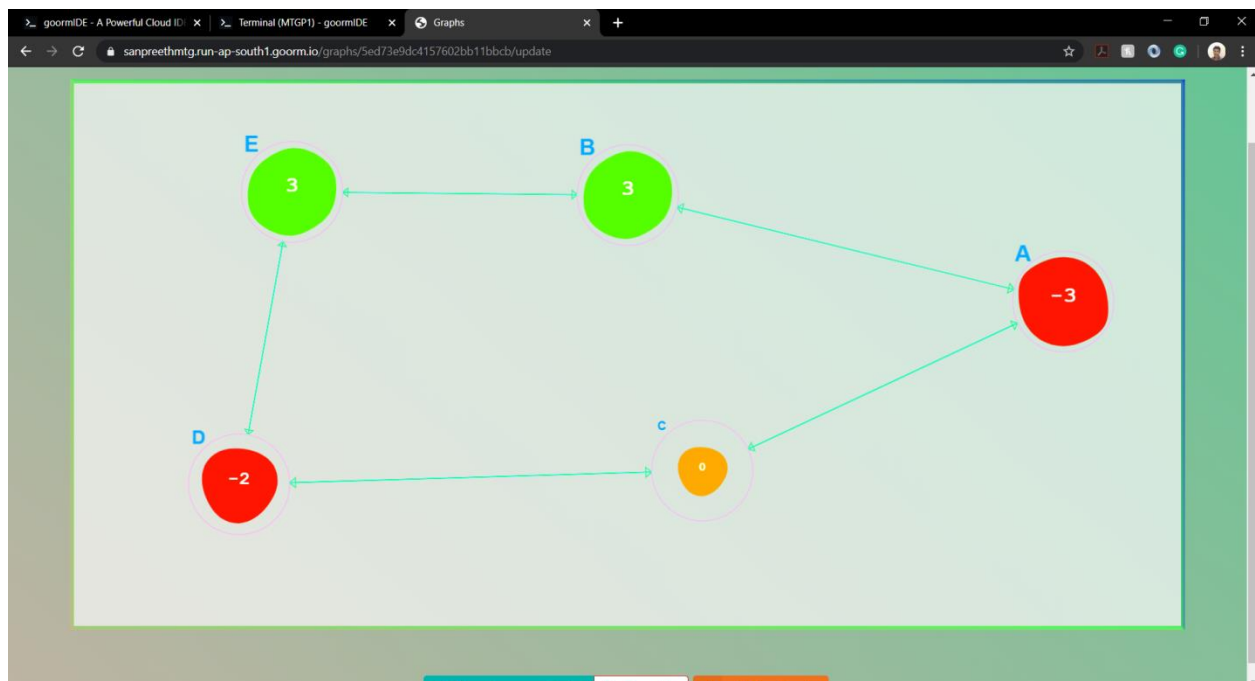
2020-06-03T03:00:42.020+0000 I NETWORK [conn6] received client metadata from 127.0.0.1:44348 conn6: { driver: { name: "nodejs|mongoose", version: "3.5.7" }, os: { type: "Linux", name: "linux", architecture: "x64", version: "4.4.0-1107-aws" }, platform: "Node.js v10.16.3, LE (unified)", version: "3.5.7|5.9.12" }

Redirecting to Update graph.

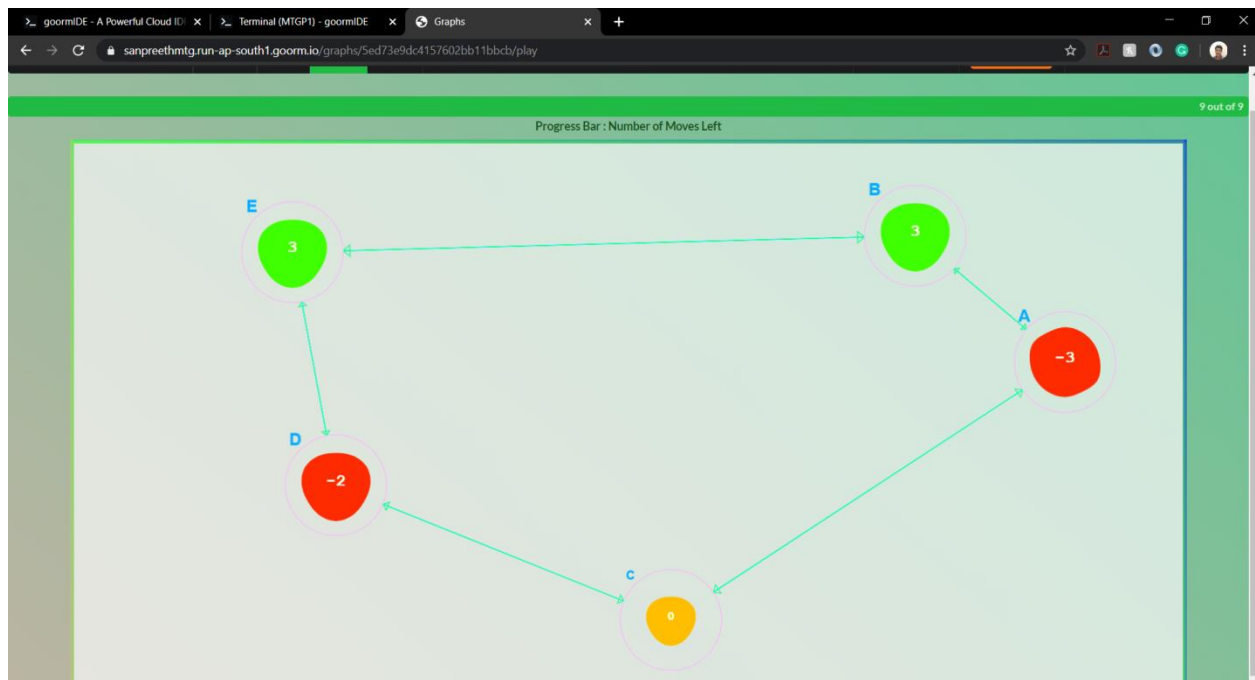
In this user can change the positions of nodes in the graph.



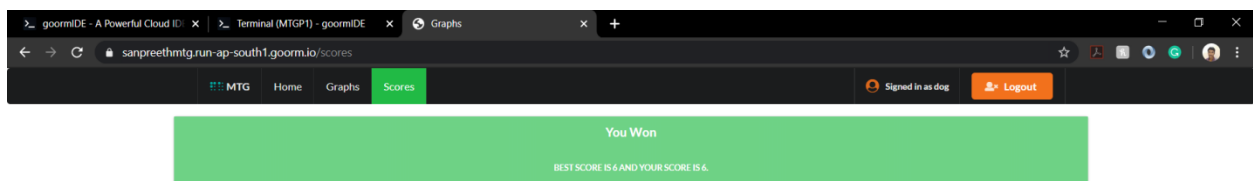
After Changing postions of nodes, Press play game.



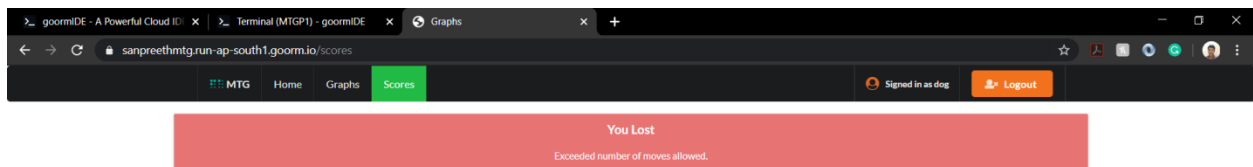
Game starts and shows the number of moves left on top of the screen



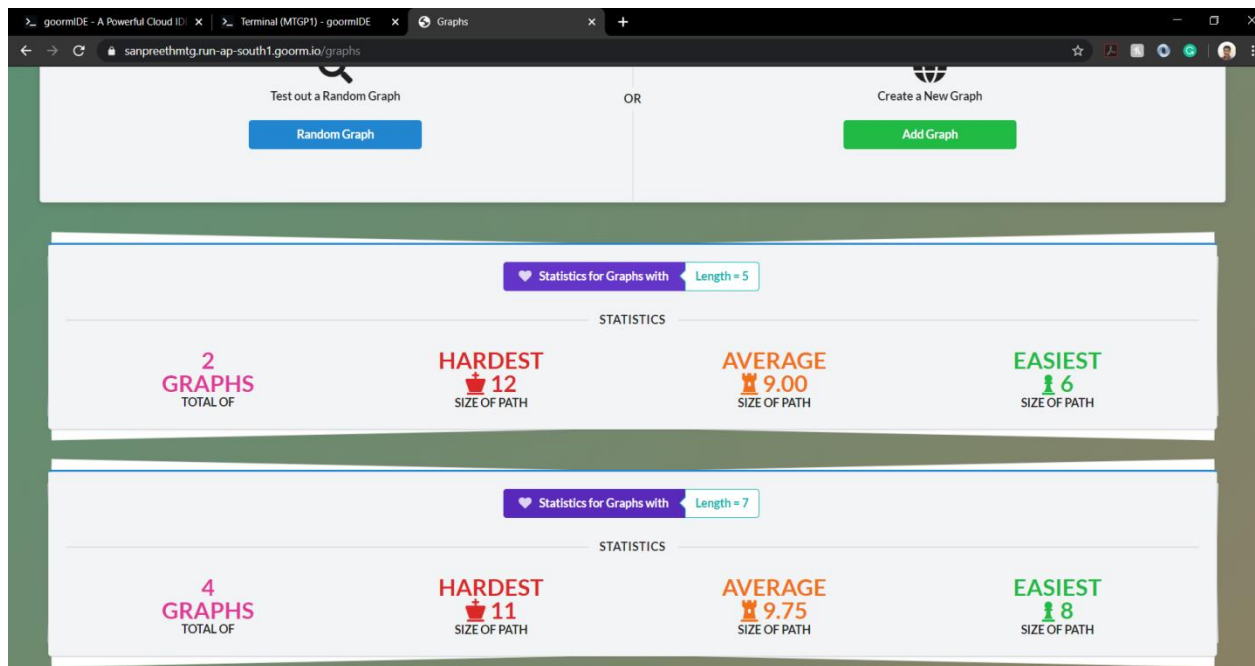
If player wins the game in less than or equal to assigned moves then he wins .



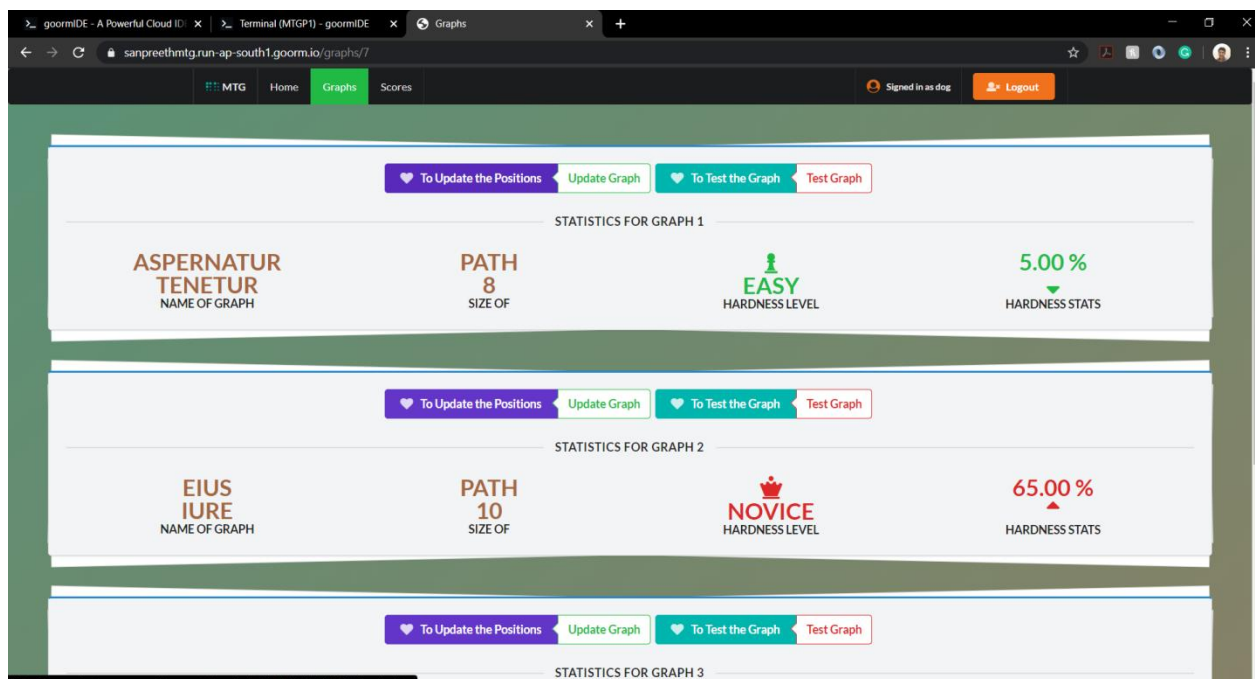
If player can not win in assigned moves , then player loses the game.



If User plays from existed game . The player must select the number of nodes for the game .

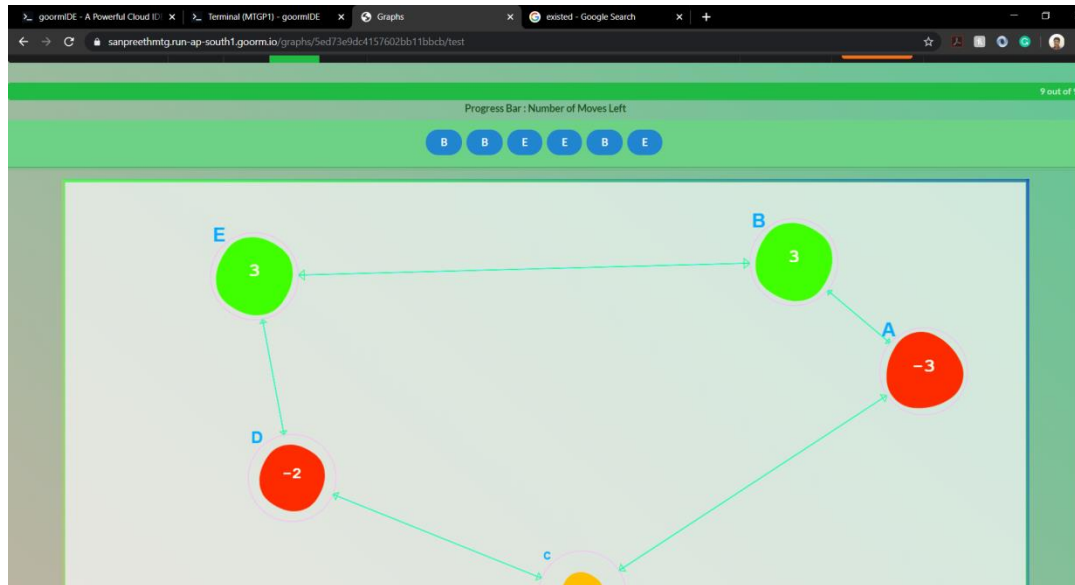


The player will select the toughness in game by seeing the path values and start the game directly.



To get solution of game press Test graph

And can get solution from top of game and by pressing the respective nodes.



CONCLUSION

Our Graph structure can be applied in many fields. Facebook uses this Graph Structure to keep track of its Users and their Connections (Friends). In many ways the Graph structure depicts how our society is connected where a Node can be considered as a Person and its Connections as the Person's close ones or friends and Facebook utilizes just this to construct a Graph structure for its Database.

REFERENCES

1. <https://thedollargame.io/>
2. <http://people.math.gatech.edu/~mbaker/pdf/g4g9.pdf>

