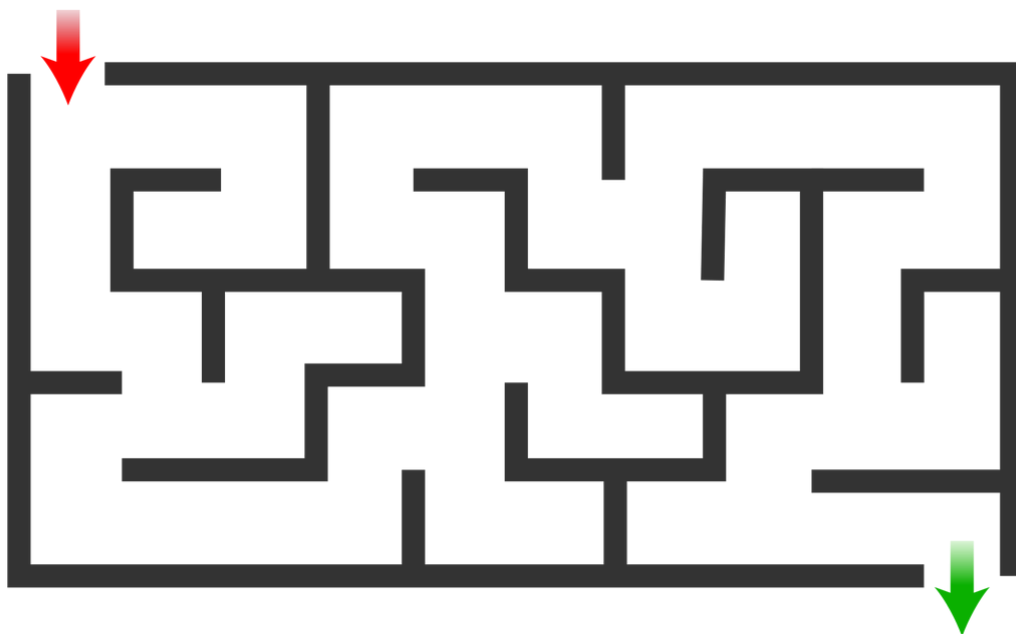


**Information and Communication
Technology
SBA - Game making**



Name: Yu Chung Yau
Class: 6C (18)
Game: 2d Maze

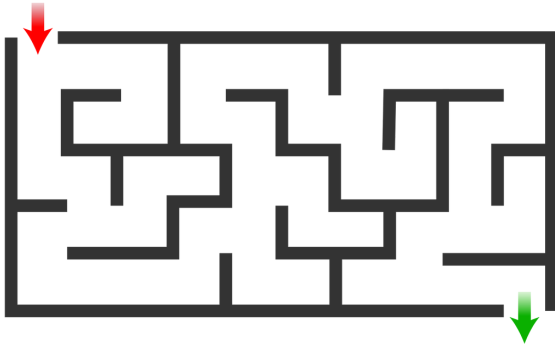
TABLE OF CONTENTS

PROPOSAL	2
DESIGN	2
IMPLEMENTATION	4
Flow Chart	4
Overview of Main Menu	5
Overview of Main Game	6
Overview of leaderboard	8
Overview of Guide	10
Algorithms	11
Variables	20

PROPOSAL

Game name: 2D Maze Game

Inspiration:



Inspired by simple maze games for kids. It is further enhanced by adding competitive elements to make the game more exciting.

Game description:

In the 2d maze, players explore around with WASD/arrow keys, heading towards the end point as quickly as possible while dodging the randomly generated walls, and competing for the no.1 on quickest completion on the leaderboard.

DESIGN

User Interface:



Fig.1: Main menu

- Players can use up/down arrow keys to navigate between different options. Spacebar to confirm selection.

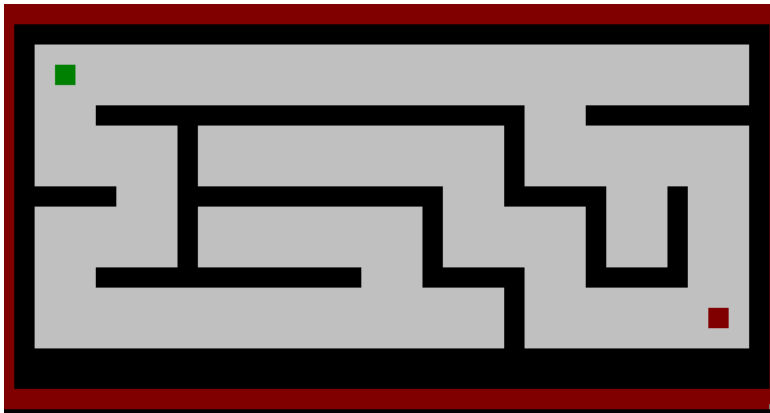


Fig. 2: Start game

- With WASD/arrow keys, players(green dot) can navigate through the random generated maze towards the end point(red dot).



Fig. 3: Leaderboard

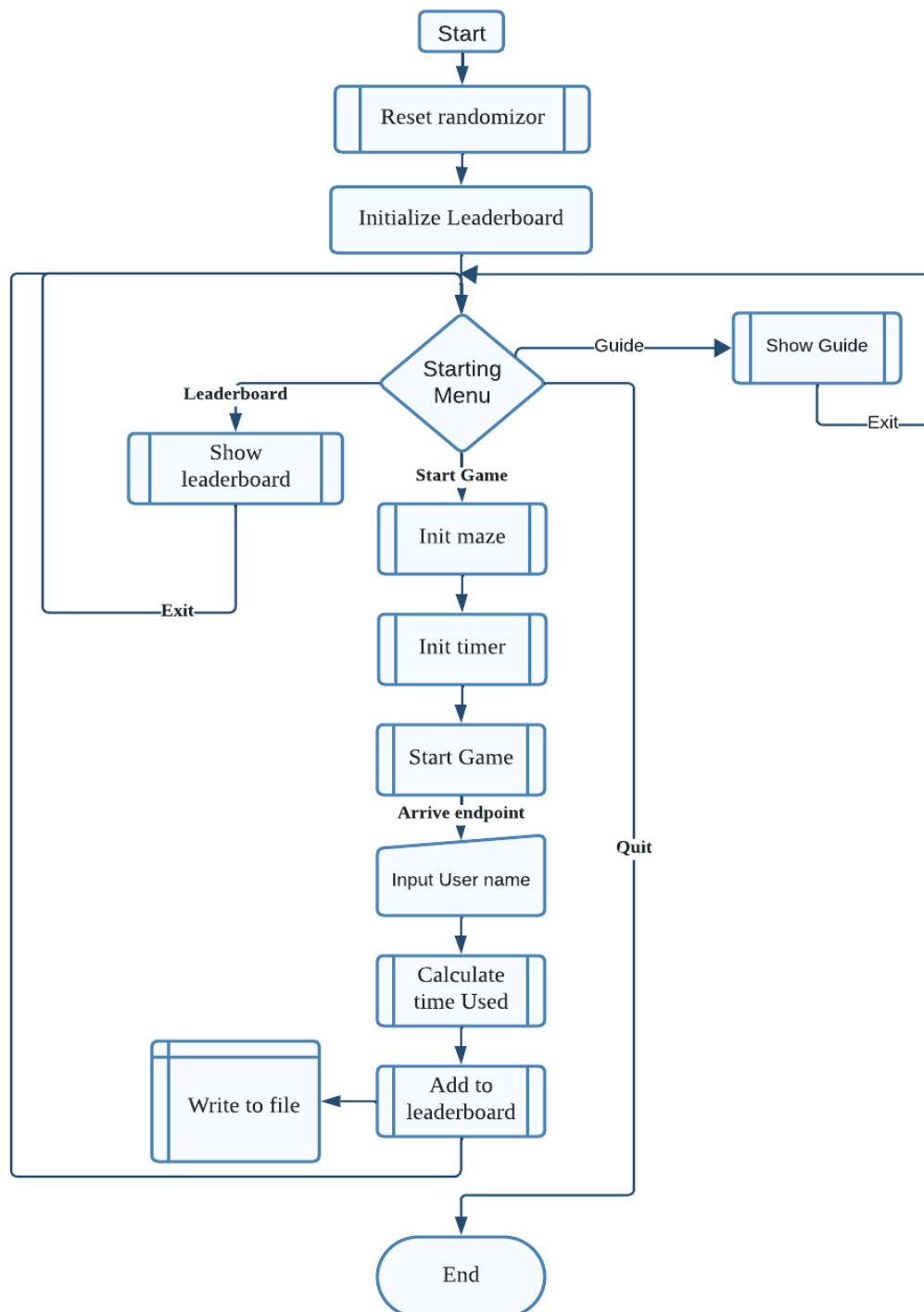
- Players can view the leaderboard within the game to see who has the quickest completion time.

File format:

- .board file for storing leaderboard
- .map file for storing the default map
- .exe file for executing the program
- .pas file for compiling into exe file with free pascal compiler.

IMPLEMENTATION

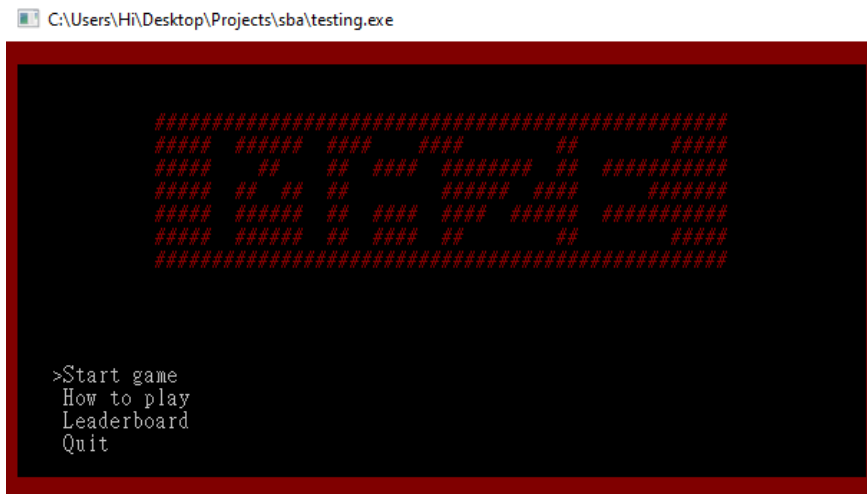
Flow Chart



Overview of Main Menu

1. Initialization

At the start of the program, a red border is generated around the window such that `WINDOW_SIZE_X = 77; WINDOW_SIZE_Y = 21`. Then the selection menu is printed at the bottom of the window and the name of the game is printed on the center of the window. As shown below:



2. While Loop

In a while loop, the user's input is taken by the `readkey` function, and handled accordingly. The `Selected` option is originally set to 1, then when the down arrow is pressed for example, the `Selected` option increases by one, being a pointer pointing towards the option below (next option in the array). Additionally, the small arrow: '`>`' moves downwards by one coordinate, indicating the currently selected option to the player.

3. Confirm selection

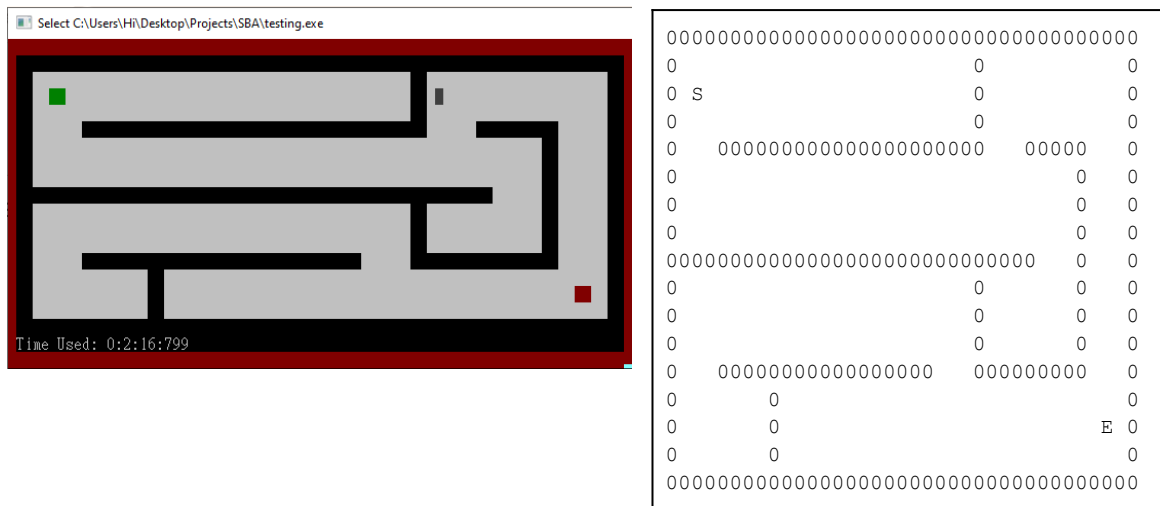
Once the spacebar is pressed, the selection is confirmed, thus triggering different functions in a case switch operation. Here's the pseudo code the operation:

```
case Selected option of
  1: start the game;
  2: generate the guide;
  3: generate the board;
  4: selectedKey := 0; //exiting the while loop, thus
terminating the program
end;
```

Overview of Main Game

1. Initialization

By using a depth first search algorithm implemented through recursion, walls of the default map are removed, thus generating a maze with 1 way to the exit. Then, it is stored in a 2 dimensional character array where 'S' represents the start point, 'E' represents the end point, '0' represents the wall. Here is an example of the 2d map array and the resulting map:



When the game runs, a while loop is created, taking in and processing input from the user continuously by Readkey function. For example, if 'w' is pressed, the y coordinate of the player will decrease by one. Then, a new green dot is drawn onto the new position on the board while the original position is cleared. Creating the effect of a moving entity.

3. Collision detection

Whenever a player updates its position, the new position is checked against the 2d map array. If the new position of the player is empty in the array, the player can move into it. However, if the new position of the player in the array contains '0', the player's coordinate should be reverted and remain unchanged. Thus, they cannot pass through the walls of the maze. Additionally, if the new position of player in the array contains 'E', the game ends, and the end game functions are triggered.

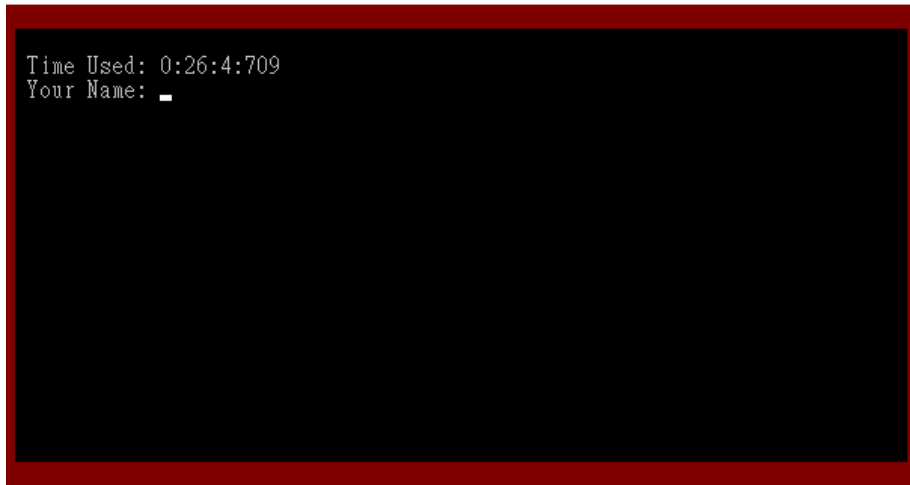
4. Timer

Within the for loop, the TimeUsed is updated. By calculating the difference between the current time in milliseconds and the time at the start of the game, the time used can be found. Then, it is converted into hours, minutes, seconds, and milliseconds by the function DecodeTime.

5. End Game

Once the game ends, the timer stops and a prompt appears, showing the time used by players and asking for the player's username.

C:\Users\Hi\Desktop\Projects\SBA\testing.exe



After entering the username, the data is validated until $3 \leq \text{length of username} \leq 20$, so that the username can be displayed correctly in the leaderboard. Then, the `userName` and `timeUsed` are added to the leaderboard. Lastly, the main screen reopens to allow players to replay the game.

Overview of leaderboard

1. Initialization/Reading file

After the start of the program, the leaderboard is read from the saved 'default.board' file. Then, the content of the table is saved into an array[1..10] of records, ready for further uses.

2. Initial Sorting

After retrieving the data from the file, the records are sorted once using bubble sort with a nested for loop. The timeUsed of each record is compared, thus sorting the array into ascending order of rankings according to maze completion speed.

3. Adding records

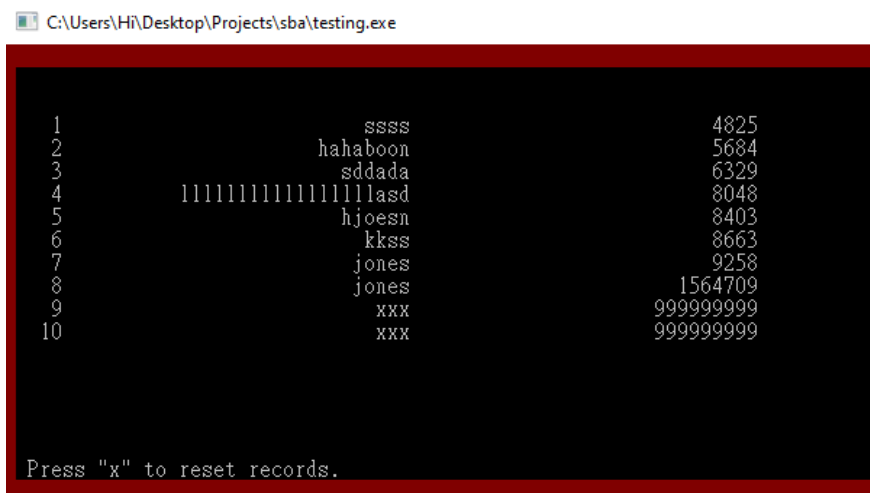
Once the game ends, the time and username is collected and compared with existing data. With insertion sort, the new data can be inserted into the already sorted array quickly and deleting the slowest record.

4. Writing file

After a new record is created, the array is written back into the file in order to save it from volatile memory into nonvolatile memory, thus allowing the users to reopen the leaderboard even after relaunching of the program.

5. Generate leaderboard

After choosing the 'Leaderboard' option in the main menu, the timeUsed of each record are decoded and written onto the window together with the username and rankings.



```

C:\Users\Hi\Desktop\Projects\sba\testing.exe

1          ssss          4825
2      hahaboon          5684
3          sddada          6329
4      1111111111111111lasd      8048
5          hjoesn          8403
6          kkss          8663
7          jones          9258
8          jones          1564709
9          xxx          999999999
10         xxx          999999999

Press "x" to reset records.
  
```

When any key is pressed, the leaderboard is closed and returns to the main menu.

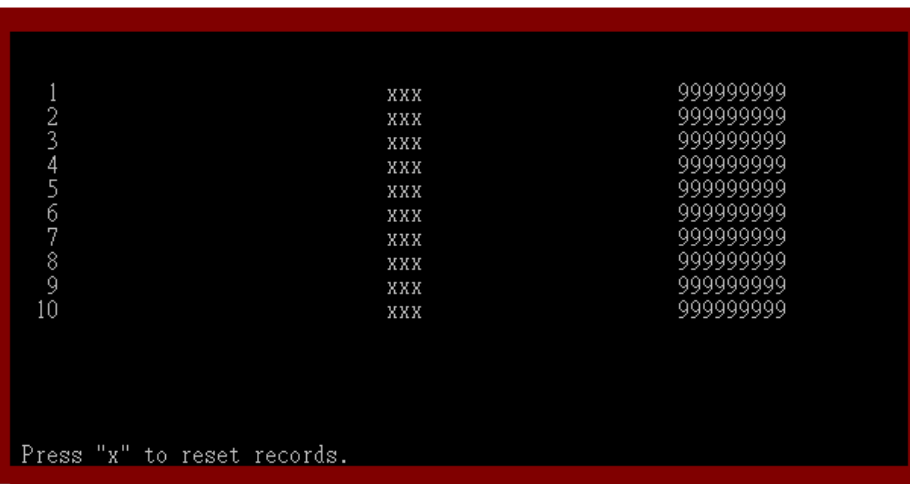
6. Reset leaderboard

Within the leaderboard window, you can press 'x' to clear the records. Then a confirmation pops up in order to prevent accidental deletion of records. After confirmation, all records in the array are set to

```
x.playerName := 'xxx';  
x.timeUsed := 999999999;
```

Then, the new array is written back into 'default.board', thus updating the leaderboard permanently. The new timeUsed 999999999 ensures that it sits at the back of the leaderboard when new records are created, thus not affecting the ranking of new records.

C:\Users\Hi\Desktop\Projects\sba\testing.exe



Here is the leaderboard after resetting.

Overview of Guide

1. Draw border

When 'How To Play' is selected from the main manu, the window is cleared and the border is drawn again to clear everything from the main menu.

2. Write text

Then, the follow guide with strategies and How to play is write to the screen:

1. Use WASD or arrow keys to navigate.
2. Use both to get double speed.
3. Hold keys to gain speed boost.
4. Head towards the red dot to win.
5. Try and get the quickest time possible!!!!

By skipping a line for each sentence, here is the resulting 'How To Play' screen:

C:\Users\HI\Desktop\Projects\sba\testing.exe

```
1. Use WASD or arrow keys to navigate.  
2. Use both to get double speed.  
3. Hold keys to gain speed boost.  
4. Head towards the red dot to win.  
5. Try and get the quickest time possible!!!!
```

3. Exiting

After the guides are written out, it stays in this window until any key is pressed and caught by the 'Readkey' function. If any key is pressed, the 'how to play' window is cleared, and the main menu is generated again.

Algorithms

1. Leaderboard Sorting I (Bubble sort)

After reading the array from the file, the records are sorted again to ensure the rankings are arranged according to the ascending order of time used.

```

procedure swapRecords(x:integer; y:integer);
var
    temp:recordsType;
begin
    temp := records[x];
    records[x] := records[y];
    records[y] := temp;
end;
procedure sortLeaderboard(var leaderboard);
var
    subI, subJ:integer;
begin
    for subI := 1 to length(records) do
        for subJ := 1 to length(records)-subI do
            if(records[subJ].timeUsed > records[subJ+1].timeUsed) then
                swapRecords(subJ, subJ+1);
        end;
    end;
end;

```

Here is the code for the bubble sort algorithm in the program. Two nested for loops are used to compare the time used of each record of the leaderboard. If the time used is higher, swap it with the other record in front of the array.

2. Leaderboard Sorting II (Insertion sort)

When a new record is generated after the player finishes the game, the time used is inserted into the leaderboard array with insertion sort.

```

function Leaderboard.addRecord(userName:String; timeUsed:TDateTime):integer;
var
    subI, subJ:integer;
    h1, m1, s1, ms1:word;
begin
    //Update the time, + insertion sort into records array
    DecodeTime(timeUsed, h1, m1, s1, ms1);
    timeUsed := ((h1*60+m1)*60+s1)*1000+ms1;
    if(timeUsed > records[length(records)].timeUsed) then
        exit();
    subI := 1;
    while timeUsed > records[subI].timeUsed do
        begin
            subI := subI + 1;
        end;
    for subJ := length(records)-1 downto subI do
        begin
            records[subJ + 1] := records[subJ];
        end;
    end;
end;

```

```

        records[subI].timeUsed := timeUsed;
        records[subI].playerName := userName;
        Leaderboard.writeFile();
        exit(subI);
    end;
end;
```

With a while loop, we compare the time used of the new record to the existing records from the first in the leaderboard towards the end. If the time used of a new record is greater than the time used of a record on the leaderboard, it means that the new record should be inserted after this record. However, when the time used of the new record is lower than that of the record on the leaderboard, it means that the new record should be inserted in front of this record.

With these two methods, we can locate the index where the new record should be appended into. Then, with a for loop, we push all records after the index backwards, and lastly replace the record with the correct index with the new record.

3. Default Maze Map File Handling

To prevent boilerplate code, a version of default map is saved in a text file named 'default.map' which contains the default maze map with all walls intact:

[illegible]

To read in this map from the file, a special algorithm is used:

```
procedure initMap;
var
    mapInput: text;
    subI, subJ: integer;
```

```

begin
    assign(mapInput, './maps/default.map');
    reset(mapInput);
    for subI := 1 to mazeSizeY do
        begin
            for subJ := 1 to mazeSizeX-1 do
                begin
                    read(mapInput, mazeMap[subJ, subI]);
                    case mazeMap[subJ, subI] of
                        'S':
                            begin
                                startPoint.x := subJ;
                                startPoint.y := subI;
                            end;
                        'E':
                            begin
                                endPoint.x := subJ;
                                endPoint.y := subI;
                            end;
                    end;
                end;
            end;
            readln(mapInput, mazeMap[mazeSizeX, subI]);
        end;
    close(mapInput);
end;

```

Using a nested for loop, each char is read and stored into the maze map 2d array. If the char is S or E, it marks the start point and the end point of the maze. Which can be changed if the default map is edited.

4. Maze Map Generation*

After the start game option is selected in the main menu, a new map is generated for each game using a depth first search algorithm and recursion demonstrated below:

1) Initiation of visited map

A two dimensional array, visitedMap, is initialized with all values false to remember if each room of the maze has been visited yet.

```

setlength(directionMap, 4, 9);
setlength(visitedMap, 4, 9);
for subI := 0 to length(visitedMap) do
    begin
        for subJ := 1 to length(visitedMap[subI]) do
            begin
                visitedMap[subI][subJ] := false;
            end;
        end;
    end;
end;

```

2) Create the direction map

```
visit(random(8)+1, random(3)+1, 9);
```

With a random X and Y coordinate, a random room is visited. After visiting the room, a random direction is chosen in order to decide the next room (one of the neighboring cells) to visit.

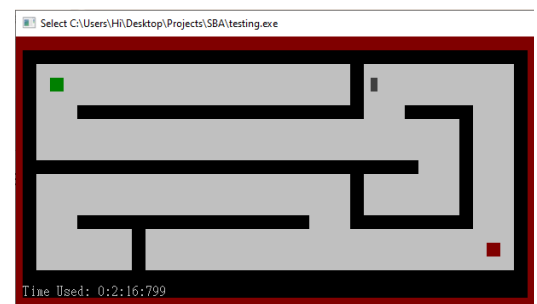
```
procedure visit(x:integer; y:integer; direction:integer);
var
  subI, subJ:integer;
  thisX, thisY:integer;
begin
  if((y > length(visitedMap)) or (y < 0))then exit;
  if((x > length(visitedMap[y])) or (x < 1))then exit;
  if(visitedMap[y, x] = true)then exit;
  visitedMap[y, x] := true;
  directionMap[y, x] := direction;
  {find neighbour cells}
  subJ := random(3);
  for subI := 1 to length(directionsList) do
  begin
    thisX := x+directionsList[((subI + subJ)mod 4)+1].x;
    thisY := y+directionsList[((subI + subJ)mod 4)+1].y;
    visit(thisX, thisY, changeDirection(((subI + subJ)mod
4)+1, 2));
  end;
end;
```

After a few iterations, the recursion might arrive in a room which all of its neighbors have visited. Then, it backtracks to the previous room which contains unvisited neighbors and visits them.

When a cell is visited, the direction of the room it comes from is saved in the direction map. With the help of the map, we can have an overview of walkalbe pathway of the maze.

```
S  ←  ←  ←  ←  ←  ↓  ←  ←
↑  ←  ←  ←  ←  ←  ←  ←  ↑
→  →  →  →  →  ↓  →  ↑  ↑
↑  ←  →  →  →  →  →  →  E
```

^Visualization of the idea of direction map.



Since every room of the map contains an arrow which will eventually point towards the same room (the first visited room), it ensures that every room is accessible by the player.

3) Removing the wall

After the direction map is created, all the walls that the arrows point towards are removed. Thus forming a map where every cell is reachable by the player.

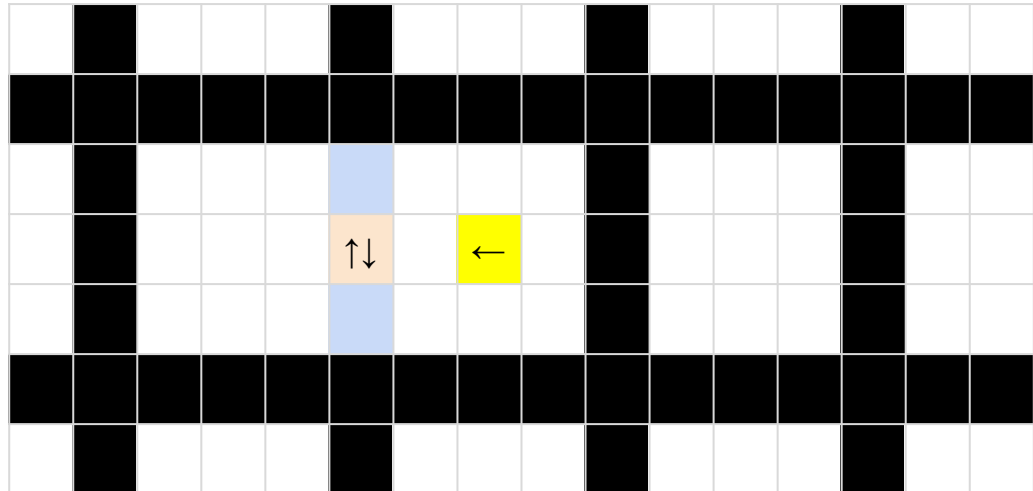
```

procedure processMap;
var
    subI, subJ:integer;
    mazeX, mazeY:integer;
    randomNo:integer;
begin
    for subI := 0 to length(directionMap) do
        begin
            for subJ := 1 to length(directionMap[subI]) do
                begin
                    if(directionMap[subI, subJ] = 0) or (directionMap
[subI, subJ] = 9)then continue;
                    {find center of this cell on maze map}
                    mazeX := (subJ-1)*4+3;
                    mazeY := (subI)*4+3;
                    {find the wall to be removed}
                    randomNo := random(2);
                    mazeX := mazeX+2*directionsList[directionMap[subI,
subJ]].x;
                    mazeY := mazeY+2*directionsList[directionMap[subI,
subJ]].y;

                    {Removing the wall}
                    if(randomNo = 0) or (difficulty = 0)then
                        mazeMap[mazeX, mazeY] := ' ';
                        mazeX := mazeX+1*directionsList[changeDirection(
directionMap[subI, subJ], +1)].x;
                        mazeY := mazeY+1*directionsList[changeDirection(
directionMap[subI, subJ], +1)].y;
                    if(randomNo = 1) or (difficulty = 0)then
                        mazeMap[mazeX, mazeY] := ' ';
                        mazeX := mazeX+2*directionsList[changeDirection(
directionMap[subI, subJ], -1)].x;
                        mazeY := mazeY+2*directionsList[changeDirection(
directionMap[subI, subJ], -1)].y;
                    if(randomNo = 2) or (difficulty = 0)then
                        mazeMap[mazeX, mazeY] := ' ';
                end;
            end;
        end;
    end;
end;

```


To find the wall needed to be removed, the (x, y) coordinates of the direction map is scaled to the (x, y) coordinates of the **center of the room**. Then, by adding 2, multiplied by the directions, the **center of the wall that should be removed** is found. Here is the visualization of one of the rooms pointing towards left:



Yellow: center of the room

Red: center of the wall to be removed

Blue: 2 sides of the wall to be removed

Example: the direction is pointing towards the left $(-1, 0)$.

After that, by rotating the direction by 90 degree clockwise and anti clockwise, the neighbor walls can be found.

With a nested for loop, it loops through the direction map, then scales it to find the corresponding cell coordinates of the room of the actual maze map. Then the wall which the direction points to is removed (set from '0' to ' '). Here is an visualization of the idea:

[illegible]

4) Draw maze

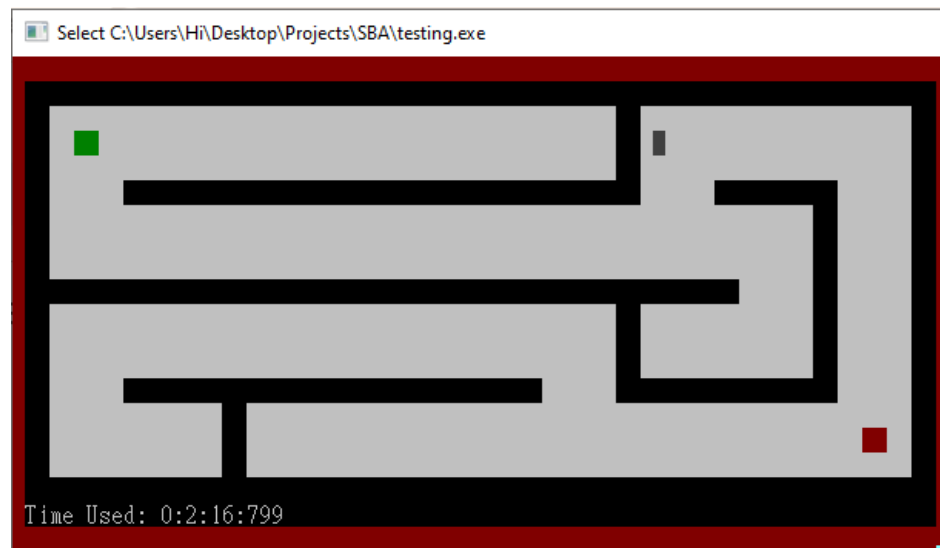
After creating the maze map with '0', 'S' and 'E', we have to convert it into a readable format and write it onto the window for the users to view it. Here's the function:

```
procedure drawMap;
var
  subI, subJ:integer;
begin
  TextBackground(black);
  for subI := 1 to mazeSizeY do
    begin
      for subJ := 1 to mazeSizeX do
        begin
          GotoXY(subJ*2, subI+1);
          TextBackground(textToColor(mazeMap [subJ, subI]));
          write(' ');
        end;
      end;
    end;
end;
```

This function uses a nested for loop which loops through the maze map and sets the text background of the corresponding coordinates of the window into the correct color. `textToColor` is a function which converts '0', 'S' and 'E' into the corresponding text background.

```
function textToColor(text:char):integer;
begin
  case text of
    '0':exit(black);
    ' ':exit(white);
    'S':exit(green);
    'E':exit(red);
  end;
end;
```

With a simple case switch operation, the char is converted into a suitable color for the text background. Finally, here is an example of the result of the maze.



5. User name Validation

In order to prevent extra long names or names too short, which will affect the display of the leader, a length validation of username input is used.

```
write('Your Name: ');
repeat
    GotoXY(14, 4);
    write(' ');
    GotoXY(14, 4);
    readln(userName);
until ((length(userName) >= 3) and (length(userName) <= 20));
```

A repeat until loop is used until the length of username entered by user is greater than or equal to 3 and less than or equal to 20. If the length is too long or too short, an empty string is written on top of the input, and asking for the user to input the username again.

6. Flush Key Buffer

Flush key buffer is a simple yet useful procedure which is reused multiple times throughout the program. During delays, the key pressed is stored in the key buffer which will be read once the delay is over. However, this could be a way to cheat by moving the player before the game starts (during the red yellow green light stage). Here is the procedure:

```
//other functions
procedure FlushKeyBuffer;
begin
    while KeyPressed do
        Readkey;
end;
```

Using a while loop, it reads all the keys in the key buffer until keypress is false (the key buffer is empty). Thus all the keypresses during delays are removed at once.

Variables

Here is a list of the major variables used in the program and its usage.

Variable	Var name	Type	Usage
Maze map	mazeMap	array of array of char	To store the map of the maze in the form of: ‘0’: wall ‘ ’: empty space ‘S’: start point ‘E’: end point
Direction map	directionMap	array of array of integer	To store the direction of walls that should be removed from each room during maze generation.
Direction list	directionsList	array[1..4] of twoDCoord	Store the direction in the form of a 2d vector. (X: 0; Y:-1),{up} (X: -1; Y:0),{left} (X: 0; Y:+1),{down} (X: +1; Y:0){right}
Visited map	visitedMap	array of array of boolean	To store if each room has been visited during the maze generation.
Leaderboard array	recordsRay	array[1..10] of recordsType recordsType = record rank:integer; playerName:string; timeUsed:TDateTime; end;	To store the records of the leaderboard. With each record containing rank, player name, and the time used.
Start and end time	startTime, endTime	TDateTime	To store the start time and end time of the game. In order to calculate the time used for the leaderboard.