Elabyrinth Game Application

Group 9

CONTENTS COVERED

1. Introduction
2. Data flow diagrams
3. Functions
4. Screenshots
5. Non functional requirements
6. Applications
7. Challenges Faced
8. Conclusion

Image by storyset

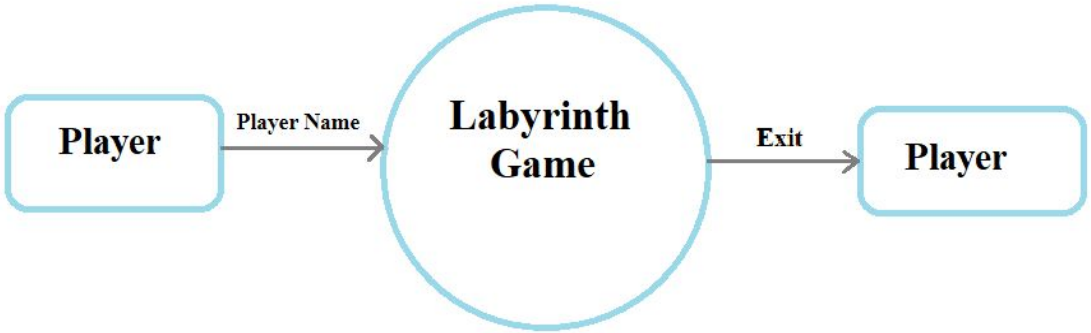
What is the actual game and its purpose ?

A Labyrinth is a tree-based maze in the Royal House of England where the royals often enjoyed themselves with their servants who would get in and find their way out of a labyrinth maze.

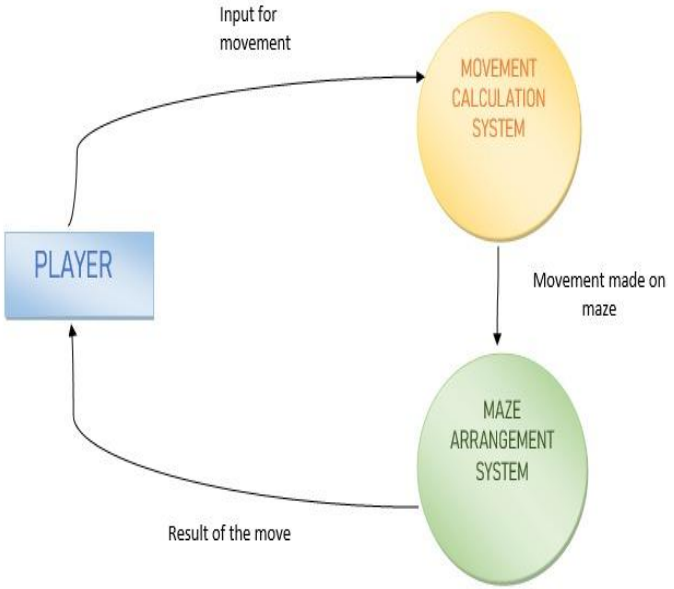
The purpose of this project is to implement a maze game that involves multiple mazes to play from. The game involves varied walls that restrict player movement and bombs that kills the player.

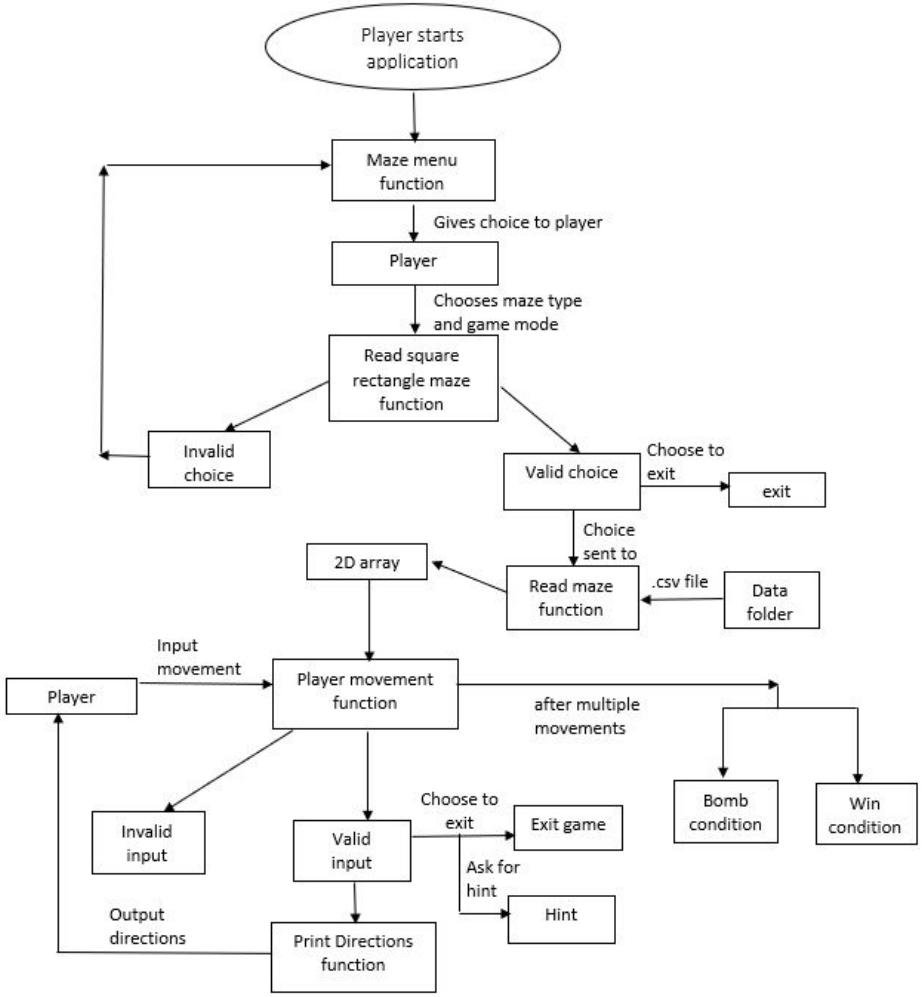
The main purpose of such puzzles is not only entertainment but also teaching by increasing our knowledge and stimulating curiosity.

Data Flow Diagram 0



Data Flow Diagram 1





Header files ,Macros and Global declaration of Variables.

- The header files contained the set of predefined standard library functions used for the game.
- The C preprocessing directive “#include”. All the header file have a '. h' an extension.
- The macros used here to define the constant value and variables with its value in the entire program that will be replaced by this macro name.
- There are some variables to be changed everywhere in program. Hence global declaration of variables is required

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*****************************************************************************
 * File    : project.c
 * Author  : sandeep,aditya,anadi mishra,harsh vardhan,shivant
 * Date    : 27-Oct-2022
 * Purpose : design of maze game
 *         : An example of eylabyrinth game
*****************************************************************************/

#define MAX_ROW 25
#define MAX_COL 25
#define LINE_BUFFER 400

int ans[MAX_ROW][MAX_COL];
int var=0, temp=0;
int row=0, col=0;
int mode=0, count=0, run=1, lock=0;
int a, b, type;
FILE *reportfile;
```

main()

- The main function serves as the starting point for program execution.
- It controls program execution by directing the calls to other functions in the program.
- We use this function to control the overall files in the game.

```c
int main()
{
    report();
    mazetype();
    options();
    printf("\nEnter the level you want to play \n");
    fprintf(reportfile, "\nEnter the level you want to play \n");
    scanf("%d",&mode);
    readfile();
    printmaze();
    rungame();
    score();
    fclose(reportfile);
    return 0;
}
```

mazetype()

- It asks the dimension of maze the user wants to play and shows the options the user can select.
- This function uses a switch case to print the maze the user choose.
- If the user choose a other input . It again asks the user to choose till he choose a correct input.

```
void mazetype(){

        printf("\nChoose the mazetype u want to play\n");
        printf("\nSquare maze - 1\n");
        printf("Rectangular maze - 2\n");
        scanf("%s",&type);

if(type>='1'&&type<='9'){

    switch (type)
        {
        case '1':
            printf("\n You choose square maze");
        break;

        case '2':
            printf("\n You choose rectangular maze");
        break;

        default:
            printf("Invalid input");
            printf("   \n\n\n                ---------    \n\n    ");
            mazetype();
            break;
        }
}
else{
    printf("Invalid input");
    printf("   \n\n\n                ---------    \n\n    ");
            mazetype();

}
return ;
}
```

readFile()

- In readFile function switch case is used, if case 1 then square mazes are loaded and if case 2 then rectangular mazes are loaded.
- We set break as a default, if player doesn't enters a value.

```
void readfile(){
      printf("\nEnter the level you want to play \n");
       scanf("%d",&mode);

       switch (type)
          {
          case '1':
          square();
          break;

          case '2':
          rectangle();
          break;

          default:
          break;
          }
}
```

options()

In options function, player is prompted to select the level of maze he wants to play.

Like :

- Press 1 for EASY MAZE.
- Press 2 for MEDIUM MAZE.
- Press 3 for HARD MAZE.

```
void options(){

        printf("\n1-Easy maze\n");
        printf("2-Medium maze\n");
        printf("3-Hard maze\n");
        printf("9-Quit the game and exit\n\n");
}
```

Press 9 for QUIT AND EXIT THE GAME.

square()

- Square function calls another function that is responsible for reading CSV maze files.
- Square function is used for printing the square maze.
- If player wants to play easy maze then it print easy square maze.
- If player wants to play medium maze then it print medium square maze.
- If player wants to play hard maze then it print hard square maze.

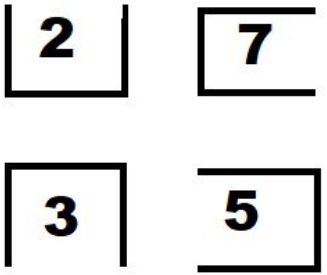
```
void square(){
            scanf("%c",&mode);
        switch (mode)
            {
             case '1':
             if (EXIT_FAILURE ==  readMazeCSV("../data/square_easymaze.csv")){
             printf(" Easy maze.\n\n");
             }
                 printf("\nYou choose easy level , can do better \n");
                 break;

             case '2':
             if (EXIT_FAILURE ==  readMazeCSV("../data/square_mediummaze.csv")){
             printf("Medium maze.\n\n");
             }
                 printf("\nYou choose medium level (keep going:)\n");
                 break;

             case '3':
             if (EXIT_FAILURE ==  readMazeCSV("../data/square_hardmaze.csv")){
             printf("Hard maze.\n\n");
             }
                 printf("\nYou choose hard level wow\n");
                 break;

             case '9':
             EXIT_SUCCESS;
             exit(0);
             break;

             default:
              printf("invalid move\n enter the level u want to play\n");
             square();
             break;
            }
}
```

rectangle()

- Rectangle function is used for printing the Rectangle maze.
- If player wants to play easy maze then it print easy Rectangle maze.
- If player wants to play medium maze then it print medium Rectangle maze.
- If player wants to play hard maze then it print hard Rectangle maze.

```
void rectangle(){
            scanf("%c",&mode);
        switch (mode)
            {
                case '1':
                if (EXIT_FAILURE ==  readMazeCSV("../data/rectangle_easymaze.csv")){
                printf(" Easy maze.\n\n");
                }
                    printf("\nYou choose easy level (can do better:)\n");
                    break;

                case '2':
                if (EXIT_FAILURE ==  readMazeCSV("../data/rectangle_mediummaze.csv")
                    printf("Medium maze.\n\n");
                }
                    printf("\nYou choose medium level (keep going:)\n");
                    break;

                case '3':
                if (EXIT_FAILURE ==  readMazeCSV("../data/rectangle_hardmaze.csv")){
                printf("Hard maze.\n\n");
                }
                    printf("\nYou choose hardmaze level (keep it up:)\n");
                    break;

                case '9':
                EXIT_SUCCESS;
                exit(0);

                default:
               printf("invalid move\n enter the level u want to play\n");
                rectangle();
                break;
            }
}
```

Maze creation logic



Cell value = 7*2 =14

Additional multiple
BOMB : 11
KEY : 13
ALERT: 19

4*4 maze

21	35	35	15
6	91	105	10
114	55	42	5
14	35	70	5

easy maze

5*8

399	385	1995	105	15
6	0	6	14	30
42	105	1330	385	570
42	5	0	21	30
42	15	21	70	30
42	570	6	7	15
798	110	21	35	15
54	65	2	7	10

easy maze

4*4

21	35	35	15
6	91	105	10
114	55	42	5
14	35	70	5

Medium maze

8*8

21	105	5	3	21	35	105	5
42	70	105	210	210	15	6	285
2	7	210	70	210	70	3990	330
21	15	2	21	30	7	10	38
14	210	35	70	70	105	105	195
21	2730	105	105	105	210	70	30
2	6	14	70	70	70	15	2
7	70	35	35	5	7	70	5

Medium maze

5*10

21	15	0	21	156
14	3990	1155	570	6
3	14	1330	210	30
14	35	35	30	6
21	105	105	210	10
6	14	210	30	33
42	15	42	70	30
42	70	30	0	6
42	35	210	285	6
182	35	1330	770	190

readMazeCSV(char* fileName)

- This function is the core part of our game it is responsible for reading all the different maze file and maintaining the records of it.
- For reading the file we use the File handling and also using the string function ‘strtok( )’ for retrieving the data from the files.

```
int readMazeCSV(char* fileName)
{
    char buffer[LINE_BUFFER] ;
    char *record,*line;

    FILE *fstream = fopen(fileName,"r");
        if(fstream == NULL)
            {
                printf("\n File opening failed ");
                return -1 ;
            }

    row=1;
    col=0;

    while((line=fgets(buffer,sizeof(buffer),fstream))!=NULL)
        {
            record = strtok(line,",");
            col++;

            while(record != NULL)
            {
                ans[row][col] = atoi(record) ;
                record = strtok(NULL,",");
                temp=max(col,temp);
                col++;
            }

            col=0;
            var=max(row,var);
            row++;
        }
        fclose(fstream);

    return 0;
}
```

rungame()

The function rungame() is responsible for running the program and it also contain other functions like : key(), bombsuggets(), showdirection() etc that help to run our program smoothly.

```
void rungame(){
    a=1;
    b=1;

    while(1){
        printf("\nYou are currently at position of %d,%d",a,b);
        key();
        bombsuggest();
        printdirections();
        showdirections();
        move();
        dead();
        win();
        printf("   \n\n\n     ---------     \n\n    ");
        count++;
       }

}
```

key()

- The key() function is basic and small function that describes that player have got the keys or not from the mazes.
- If the player does not have the key and it reaches
- the end than player not win the game instead he
- got message “You not get the keys”.
- Player cannot win the game without having all the keys.

```
void key(){

    if(ans[a][b]%13==0&&lock==0){
        printf("You have got the key");
        lock=1;
    }
    return;
}
```

showdirections()

This function is responsible to guide the player how to navigate in our Elabyrinth

This function shows the main set of keys that are used to

- Navigate inside the maze
- Ask the game for hints
- Exit the game

```
void showdirections(){
    printf("0-Top\n");
    printf("1-Bottom\n");
    printf("2-Left\n");
    printf("3-Right\n");
    printf("4-Exit\n");
    printf("5-Hint\n\n");
}
```

printdirections()

Print direction is the most important function from the players perspective because this is the function which shows the player which directions he can possibly make a move and which directions he cannot move.


```c
void printdirections(){

    printf("\nYou have the following ways\n");
    printf("\n");
    printf("  ");
        if(ans[a][b]%2==0){
            printf("   ^ ");
        }
        else{
        printf("   o ");
        }

    printf("  \n");
    printf("\n");
    printf("     O\n");
```

bombsuggest()

Bomb suggest function is a helpful function to the player. This function warns the player whenever the player is nearby a bomb to make the player choose his/her next move with caution

```
void bombsuggest(){
    int run=ans[a][b];
        if(run%19==0){
        printf("\n Cautious !!!!! Bomb ahead");
        }
}
```

move()

This function is responsible for movement, we use switch case along with if statements to control the movement of player, the if walls are present on any sides we restrict the movement in that directions.

```
void move(){
    int choice;
    scanf("%d",&choice);

    int x=a;
    int y=b;
    int value=ans[x][y];

    switch (choice)
        {
        case 0:
            if(value%2==0){
            x=x-1;
            }
            else{
                printf("Invalid move");
            }
        break;

        case 1:
            if(value%3==0){
            x=x+1;
            }
            else{
                printf("Invalid move");
            }
        break;

        case 2:
            if(value%5==0){
            y=y-1;
            }
            else{
                printf("Invalid move");
            }
        break;

        case 3:
            if(value%7==0){
            y=y+1;
            }
            else{
                printf("invalid move");
            }
        break;

        case 4:
            exit(1);


        case 5:
            hint();
            printf("use this hint to go further\n");
        break;

        default:
                printf("wrong choice\n");
        break;
}
        a=x;
        b=y;

}
```

hint() && printmaze()

This function invokes another function as hint for the user. If the user is stuck at any point and want to see the maze then he can use this hint function.

```
void hint(){
    printmaze();
}
```

This function prints out the maze for the player.

```
void printmaze(){
       printf("\nIt's a %d*%d maze\n\n",var,temp) ;

   for(int h=1;h<=var;h++){
   for(int q=1;q<=temp;q++){

    printf("%d ",ans[h][q]) ;
   }
   printf("\n") ;
  }

  printf("\n") ;

}
```

dead()

This function is called when a player steps into the block with bomb. The game exists as soon as this function is called.

```
void dead(){
    if(ans[a][b]%11==0){
        printf("\nYou have touched the bomb - You failed the game");
        exit(0);
    }
}
```

win()

Once you reach the last block the game finishes and so the player wins the challenge and the game exits.

```
void win(){
    if(a==var&&b==temp&&lock==0){
        printf("You have reached the end block but you dont have any key. Go back and collect the key \n");
        }
    else if(a==var&&b==temp&&lock==1){
        printf("Congrats you have won the game\n");
        exit(1);
        }
}
```

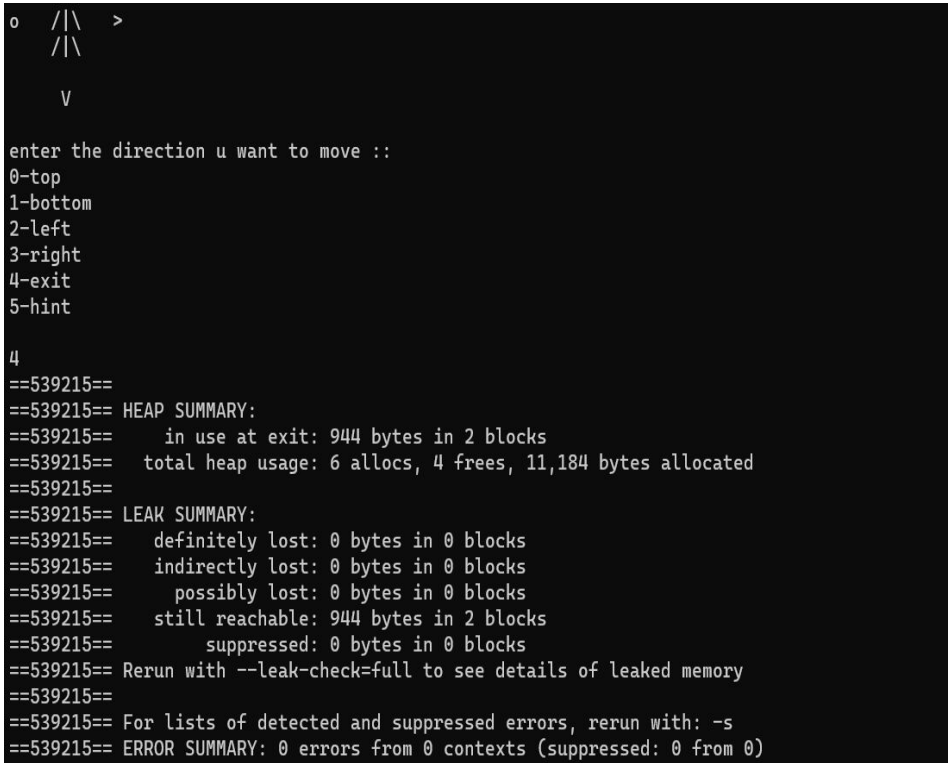
Non Functional Requirements

Makefile - It is a way of automating software building procedure and other complex tasks with dependencies. Makefile contains: dependency rules, macros and suffix(or implicit) rules.

```
cgb2-user41@instance-1:~/ProjectElabyrinth/code/Make$ cat Makefile
#####################################################
# Created by: Anadi Mishra
# Created on: 26 October 2022
#####################################################

SRCDIR = ../SRC
BINDIR = ../bin
OBJDIR = ../obj
REPORTDIR = ../reports

all:
        gcc -c ${SRCDIR}/project.c -o ${OBJDIR}/project.o
        gcc ${OBJDIR}/project.o -o ${BINDIR}/project.bin

        # Aditya's comment
        #splint -quiet ../SRC/project.c >> ../reports/splint_report.txt
        # parser error in splint
        #dynamic valgrind issue ?

#memcheck:
        #valgrind ../bin/project.bin  > ../reports/valgrind_report.txt 2>&1
clean:
        rm ${OBJDIR}/*.o ${BINDIR}/project.bin



###################### End of MakeFile ###############
cgb2-user41@instance-1:~/ProjectElabyrinth/code/Make$
```

Non Functional Requirements

Valgrind - Valgrind is a programming tool for memory debugging, memory leak detection, and profiling.



Application

1. These games emphasize long term planning, analytics, and skillful thinking in order to achieve victory. A player's decisions are important in determining the outcome of the game, and players are required to weigh the potential impact of multiple decisions in order to win.
2. Logic puzzles are an integral part of entertainment which find place in various newspapers, magazines and even mobile applications or web pages. These types of tasks are to find a solution or answer by using reasoning based on knowledge or intuition.

Image credits to safekidgames.com

Challenges Faced

1. Code Integration among teammates.
2. File Handling.
3. Continuous code modification
4. Lots of documentations.
5. Validation of all the easy and hard mazes for all possible cases.
6. Using the numbers to differentiate file for compiler to read.

Conclusion

We were successful in implementing the project within the time specified keeping the requirements in minds.

Tha game was created using 2D Arrays concept, with a mixture of conditional statements, file handling utilities and error detection code.