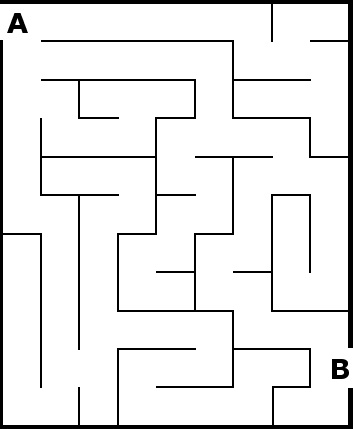
|  |
| --- |
| The Maze |
| Project Report |
| *Submitted by*  Vismay Parmar  Swapnil Shukla  Hemil Parmar |
| *Guided by*  Mr. Vishal Kanasagara |
| *Submitted to*  Department of Computer Science  Faculty of Technology & Engineering  The Maharaja Sayajirao University of Baroda  Year 2011 |





Faculty of Technology & Engineering

The Maharaja Sayajirao University

**Certificate**

This is to certify that **Vismay Parmar (Exam No. 561), Swapnil Shukla (Exam No.575) and Hemil Parmar (Exam No. 560)** have successfully completed their project work titled **“The Maze”** Under our supervision and guidance to our utmost satisfaction. This project was carried out during second semester of B.E.–II Computer Science & Engineering.

# 1. The Project

## 1.1 Project Profile

|  |  |
| --- | --- |
| **Project Title** | The Maze |
| **Aim** | To develop a computer game based on mazes |
| **Description** | Our objective is to develop a computer game based on maze solving that allows the player to solve different levels of labyrinths. |
| **Tools Used** | Borland Turbo C++ Compiler with Graphics Library |
| **Developed By** | Shukla Swapnil  Parmar Vismay  Parmar Hemil |
| **Project Guide** | Mr. Vishal Kansagara |
| **Submitted To** | Department of Computer Science  Faculty of Technology & Engineering  The Maharaja Sayajirao University of Baroda |

## 1.2 Project Overview

Maze is one of the most popular types of puzzles. It generally consists of a geometrical design with several pathways. The goal of the maze solver is completing the maze i.e. travelling along the pathways and reaching the final destination. There might be obstacles such as loops and dead ends, which make you unable to go further. You have to trace the shortest path and reach the finishing point. There can also be a stopwatch that records the elapsed time, or a countdown timer. Some advanced designs can have rewards, black holes that terminate the game, wormholes that allow the player to skip a part of the labyrinth, and conditions like checkpoints and no reverse travel.

Being a puzzle, solving a maze involves use of logical and observational skills. By a good observation of the design, one can decide which path to choose and how much time would it take to trace it. The player can develop a strategy and complete the maze in short time, while it might take longer with trial and error. A good puzzle solver would also use various advantages offered by the game only as per needed to solve the maze.

A maze can have various difficulty levels depending on the design. Easy level mazes are composed of fewer and/or shorter lines and the path is easily traceable, while medium and hard level mazes can have longer lines or a large number of shorter lines resulting in tricky turns and loops. Sometimes mazes are created using a symmetrical design but implementing different pathways. Such an example is shown in the figure. Here, a simple family of concentric circles has been converted into a maze.

In this computer game, the programming consists of designing various mazes, implementing the conditions for roaming only through open pathways and not through the walls, a timer mechanism that acts as a stopwatch and a file saving mechanism for high scores, and a menu layout for execution.

In designing portion, various lines are drawn using line function. The lines are then given particular blocking conditions to avoid passing through the wall. Every line is given two blocking conditions for its two faces. Starting and ending points are determined for every design. At the start of the game, only first level is accessible and the others are locked. After completing a level, the next level is unlocked.

A timer runs throughout the game. When the player reaches the endpoint, timer stops and the score is written to a separate file. When opened through High Scores option, the scores are sorted and displayed in ascending order of time elapsed.

Different segments of the entire program are explained in detail in next chapter.

# 2. The Programming

The source code is divided in three files:

1. MAZEMENU.CPP: The code in this file is used to construct the root menu.
2. MAZEFIN1.CPP: This is the core program containing the maze designs and submenu layout.
3. THEGAME.CPP: This is the driver program. It uses the definitions from the previous two files and contains the code for actual gameplay and high scores.

## 2.1 MAZEMENU.CPP

This file contains the coding for animated root menu. The coding consists of six functions in total: (i) menuinit(), (ii) rootmenu(), (iii) transleft(), (iv) transright(), (v) transmenu() & (vi) mazemenu(). The last function is the driver program for this particular file.

## menuinit()

The functions setbkcolor(), setcolor() and setfillstyle() are used to set the graphical properties of the functions used. The function pieslice(320,240,0,i,99) will draw a filled circular sector(“pie slice”) with a variable end angle. This function is put in a ‘for’ loop with a 10 ms delay to give an initializing or loading effect until the sector becomes a full circle.

After the loop ends, we enter another ‘for’ loop. This loop contains a blue circle at center and smaller light red circles on both sides. The function pieslice(100,240,0,360,i) draws a filled circle with variable radius on the left side of the blue one. Another similar circle is drawn on the right side. The code is put in the ‘for’ loop with a 10 ms delay that gives two circles popping out of the background.

## rootmenu(int f)

As suggested by the name, this function contains the coding for the root menu and is called after the initialization. The starting lines contain code for the design, depicting three orbs in rotation.

The function takes a flag value ‘f’. Value of ‘f’ can be 0, 1 or 2. ‘f=0’ gives us the option ‘START’, ‘f=1’ gives option ‘HIGH SCORES’ and ‘f=2’ gives option ‘EXIT’.

## transleft(), transright() and transmenu()

The function transmenu() is used for transition effects during navigation. The root menu uses two keys, left and right, for navigation. The ‘if’ condition in transmenu() identifies whether the key pressed is left arrow or right arrow and calls the appropriate transition function.

Transition functions (transleft() and transright()) are similar in coding. After a cleardevice() in transmenu(), the program draws filled circles that depict the intermediate positions of the orbs when moved. Without use of delay, the transition occurs almost immediately, giving a nice sliding effect.

## int mazemenu()

This function controls the execution of the previous functions. A flag value is defined here with initial value ‘f=0’ and rootmenu(f) is called. The program now enters a ‘while’ loop that can’t be terminated until the global variable ‘key’ gets the value 27 (Esc key) or break is encountered. If left or right arrow key is pressed, the program enters an if condition that changes the current flag value, performs the transition and calls rootmenu(f) again. This way, player can navigate in the root menu. When Enter key is pressed, ‘key’ receives the value 13 and breaks the loop. At the end of the function, it returns the flag value ‘f’ that is given to a variable ‘M’ declared in THEGAME.CPP.

MAIN MENU SNAP SHOTS

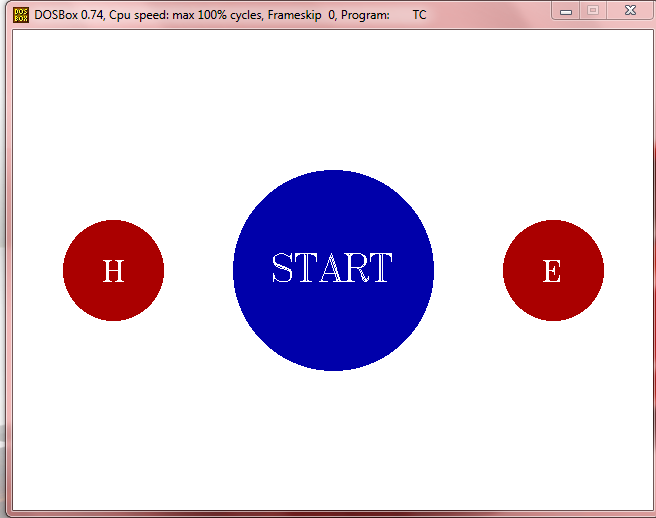
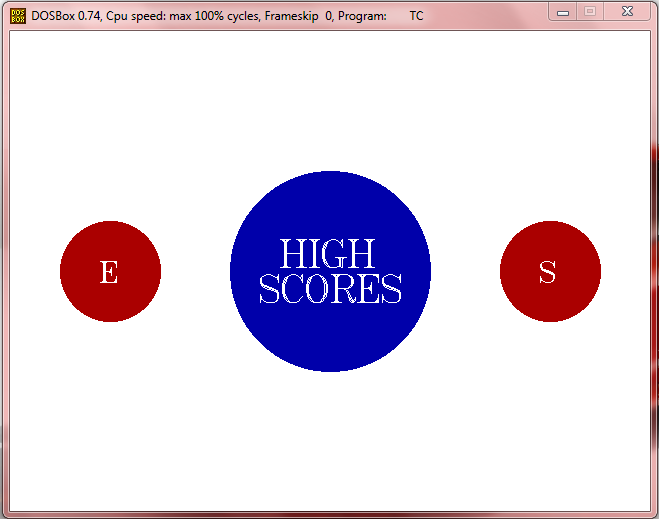
 

Figure 1 Figure 2

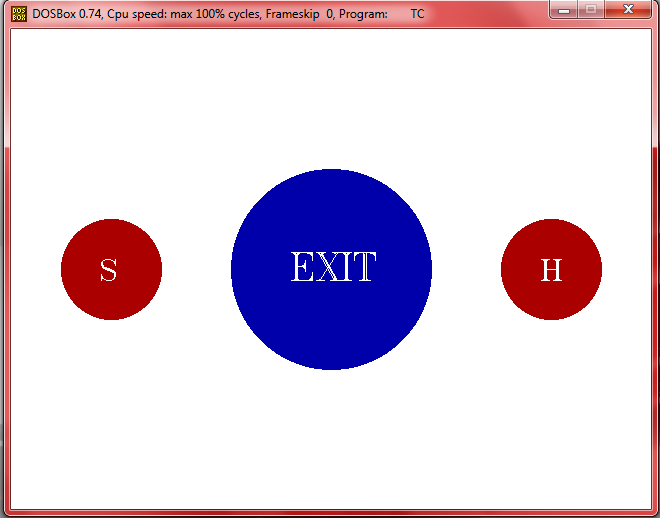


Figure 3