Project Report

A C++ Integrated Development Environment Software called Dev C++ in order to simulate the game: Pacman. The project's objective is first to create the game and then solve the maze using the search algorithms Brute Force Search (BFS) and Depth First Search (DFS). We will also figure out which algorithm is most efficient in terms of performance. The basis of performance measurement is program size, compilation time and the number of steps (score) required by the algorithm to solve the puzzle.

Objective

1. Create a maze for the Pacman game similar to Figure 1 shown below using C++.
2. Solving the maze using BFS and DFS and comparing its performance.

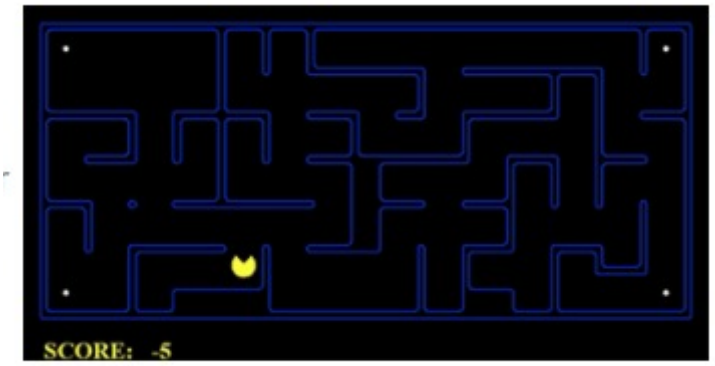
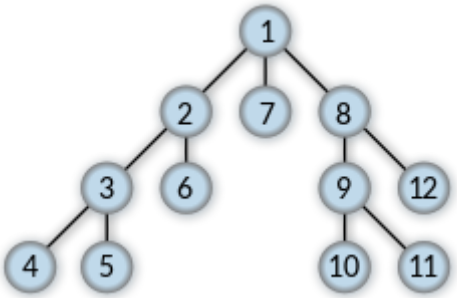


Figure 1: Pacman Maze

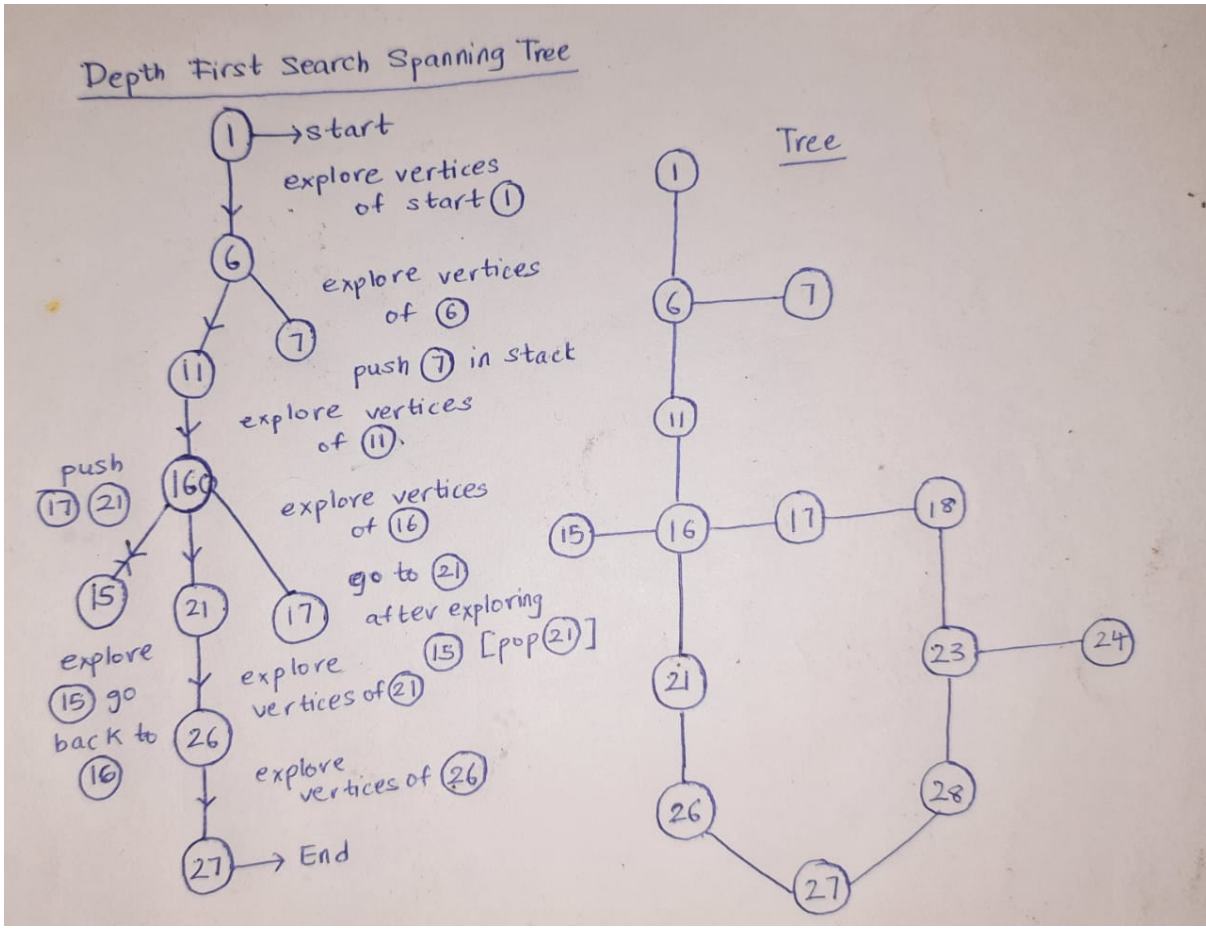
Algorithms

1. **Brute Force Approach/Search (BFS)** is a very general problem-solving technique and algorithmic paradigm that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

2. **Depth-first search** (**DFS**) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.



(a)



(b)

Figure 2 (a) and (b): Tree Diagrams

Activities

1. No changes were made to the intended project. We are working as per the plan i.e creating the game, implementing the BFS and DFS algorithms to compute the score and determining the performance of both the algorithms. However, there might be a change in the scoring system such that it can be used to determine the total steps taken by the agent to reach the food. The goal of the project is to solve the maze in the least number of steps.

2. Progress in determining the datasets: nil. We have to identify data sets yet. We have not yet reached this phase because the game is under development. Once the coding for the maze is finished, we will begin to implement the aforementioned algorithms. The number of nodes and edges is dependent on the design of the maze and the initial position of the agent in the maze (start node).

3. We are using **Brute Force Approach** and **Depth First Search** algorithms in this project. In the former case, we try every possible root in the tree and then determine the desired path. The depth first search explores edges that come out of the most recently discovered vertex. Only edges going to the unexplored vertices are explored.

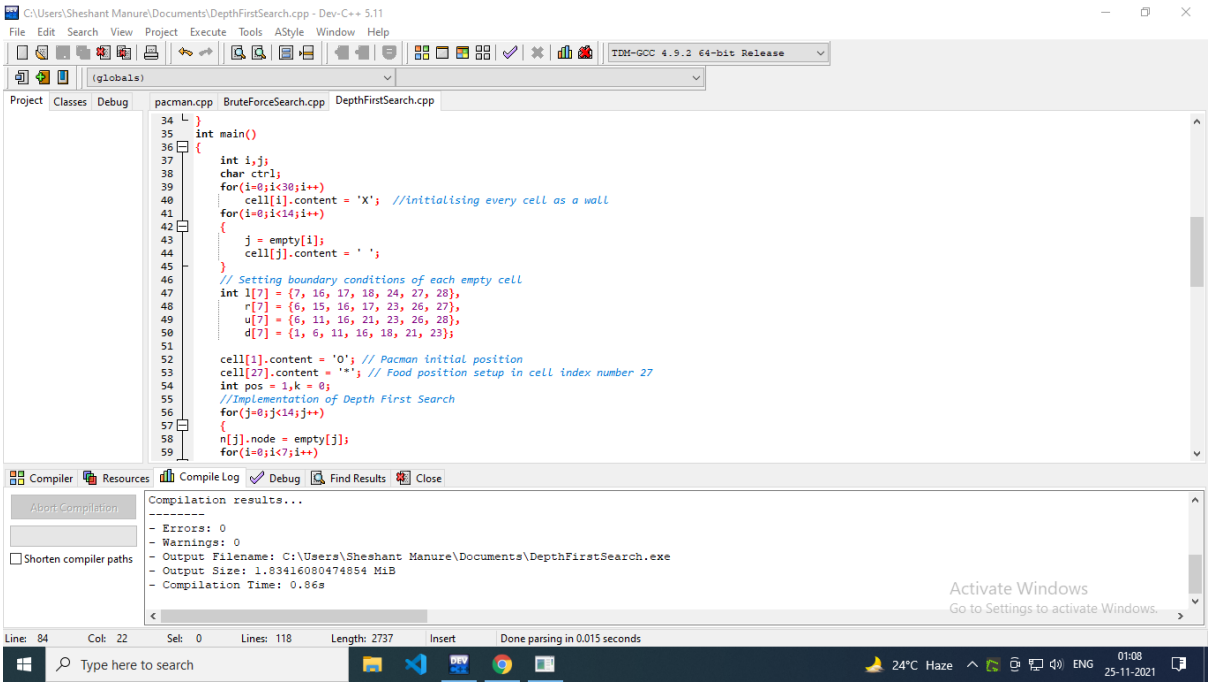
4. The experimental plan of the project implementation is to use an Integrated Development Environment (IDE) to simulate the pacman game on a console using a C++ IDE Software called Dev C++. The game development requires us to do coding, debugging, compiling and running the test of each module. It is an iterative and time taking phase. Once the game is created devoid of any error, we shall implement the algorithms. Thereafter, the performance will be computed based on the time it takes for execution of the algorithm.

Phase I: Creating the game: Pacman

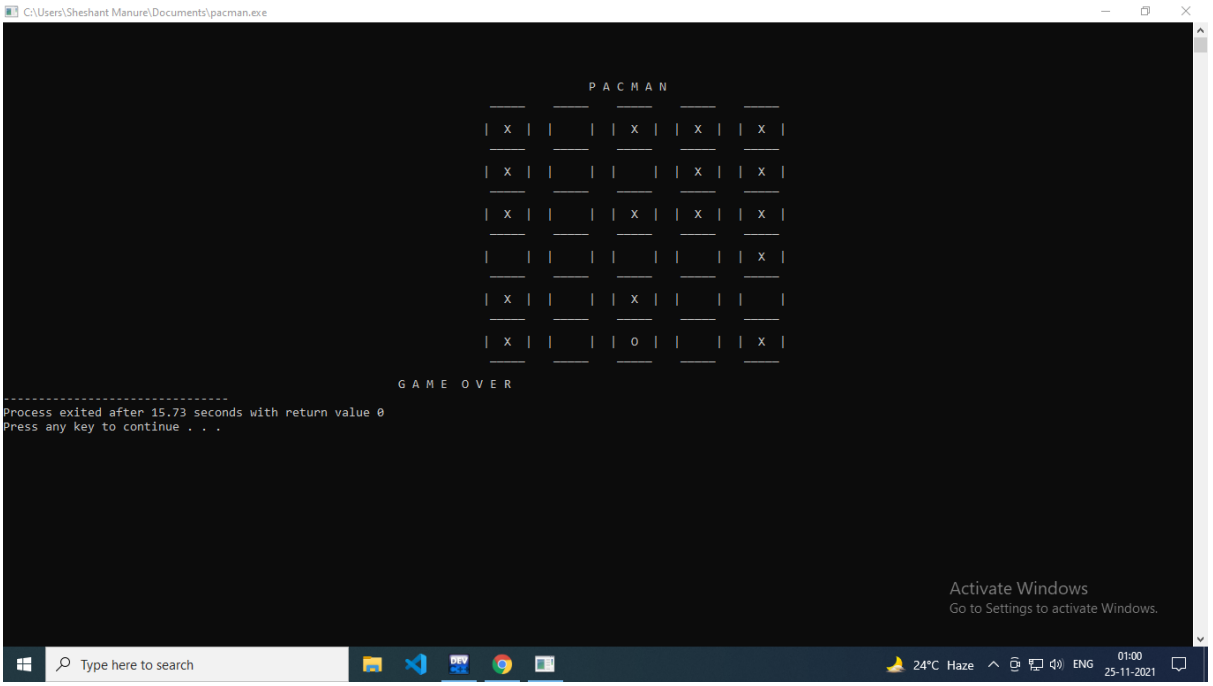
Software:

1. Dev C++ 5.11 IDE
2. Windows 10 OS

The maze comprises of 30 cells arranged in a 6 x 5 rectangular grid. Each cell in the maze can be empty or occupied by the food item or agent. Every cell is well defined by certain boundary constraints which determines the restrictions on agents' motion through the maze. Pacman (agent's name) can move through the maze in 4 allowed directions: UP (W), DOWN (S), LEFT (A) and RIGHT (D). Pacman is allowed to take only one step at a time.



(a)



(b)

Figure 3 (a) and (b): Dev C++ 5.11 IDE console

Phase II: Coding the game

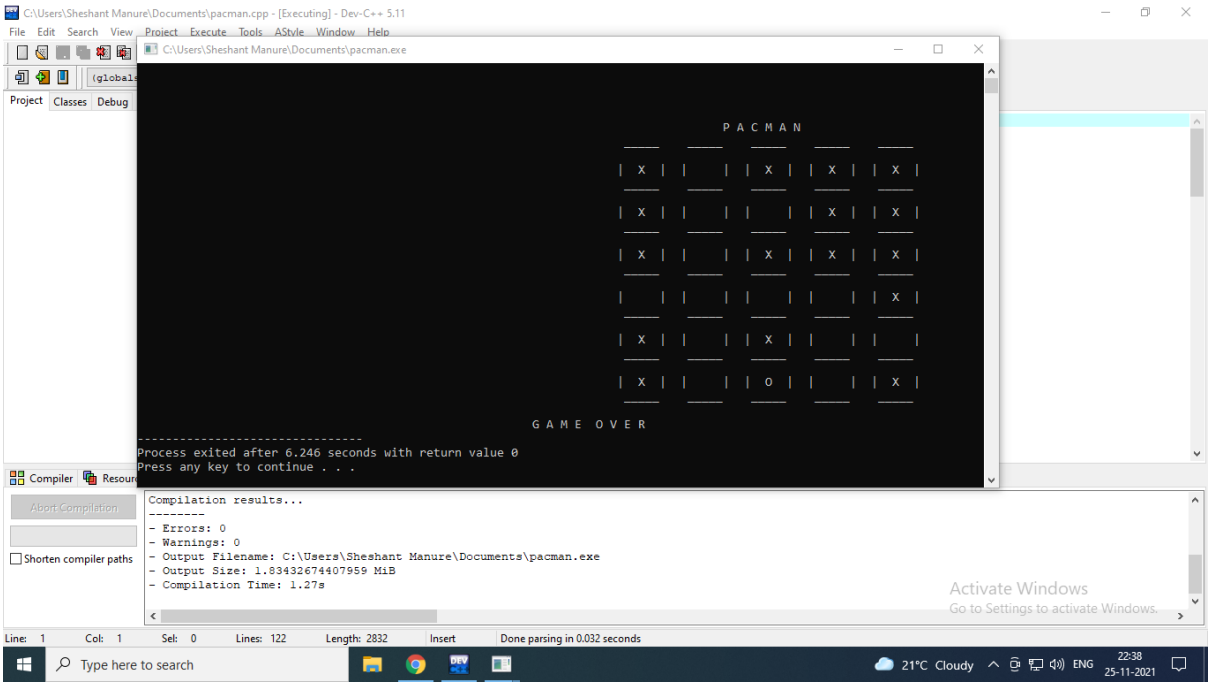
```
#include <iostream>
using namespace std;
int empty[14] = {1, 6, 7, 11, 15, 16, 17, 18, 21, 23, 24, 26, 27, 28};  //index of empty cells;
struct maze
        {
                char content; // ' ': empty, '*': Food, 'X': Wall, 'O': Pacman
        } cell[30];
void gameDisplay()
{
        int i,j,k=0;
        std::cout<<"\n\n\n\n                                                                P A C M A N\n";
        for(j=1;j<=6;j++)
        {
        std::cout<<"                                                ";
        for(i=1;i<=5;i++)
                std::cout << "  _____  ";
        std::cout <<"\n\n";
        std::cout<<"                                                ";
        for(i=1;i<=5;i++)
        {
           std::cout << " |  "<<cell[k].content<<"  | ";
        k++;
        }
        std::cout <<"\n";
   }
   std::cout<<"                                                  ";
   for(i=1;i<=5;i++)
                std::cout << "  _____  ";
        std::cout <<"\n";
}
int main()
{
        int i,j;
        char ctrl;
   for(i=0;i<30;i++)
        cell[i].content = 'X';  //initialising every cell as a wall
        for(i=0;i<14;i++)
        {
                j = empty[i];
                cell[j].content = ' ';
```

```
}
// Setting boundary conditions of each empty cell
int l[7] = {7, 16, 17, 18, 24, 27, 28},
   r[7] = {6, 15, 16, 17, 23, 26, 27},
        u[7] = {6, 11, 16, 21, 23, 26, 28},
        d[7] = {1, 6, 11, 16, 18, 21, 23};

cell[1].content = 'O'; // Pacman initial position
cell[27].content = '*'; // Food position setup in cell index number 27
int pos = 1;
while(1)
{
        if(pos == 27)
                {
                        system("cls");
                        gameDisplay();
                        std::cout<<endl<<"
G A M E  O V E R";
                        break;
                }
        else
                gameDisplay();
        std::cout<<"\n                                                Use W, S, A, D to move: ";
        std::cin>>ctrl;
        switch(ctrl)
        {
                case 'D':
                case 'd':
                        for(i=0;i<7;i++)
                                if(pos == r[i])
                                {
                                        cell[pos+1].content = 'O';
                                        cell[pos].content = ' ';
                                        pos = pos + 1;
                                        break;
                                }
                        break;
                case 'A':
                case 'a':
                        for(i=0;i<7;i++)
                                if(pos == l[i])
```

```
                                        {
                                                cell[pos-1].content = 'O';
                                                cell[pos].content = ' ';
                                                pos = pos - 1;
                                                break;
                                        }
                                break;
                        case 'W':
                        case 'w':
                                for(i=0;i<7;i++)
                                        if(pos == u[i])
                                        {
                                                cell[pos-5].content = 'O';
                                                cell[pos].content = ' ';
                                                pos = pos - 5;
                                                break;
                                        }
                                break;
                        case 'S':
                        case 's':
                                for(i=0;i<7;i++)
                                        if(pos == d[i])
                                        {
                                                cell[pos+5].content = 'O';
                                                cell[pos].content = ' ';
                                                pos = pos + 5;
                                                break;
                                        }
                                break;
                        default:
                                std::cout<<"Enter a valid input\n";
                                system("Pause");
                                break;
                }
                system("cls");
        }
        return 0;
}
```

Output



Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Programs\Documents\pacman.exe
- Output Size: 1.83432674407959 MiB
- Compilation Time: 1.27s

Phase III: Solving the Puzzle using Brute Force Approach

```
#include <iostream>
using namespace std;
int empty[14] = {1, 6, 7, 11, 15, 16, 17, 18, 21, 23, 24, 26, 27, 28};  //index of empty cells;
struct maze
	{
		char content; // ' ': empty, '*': Food, 'X': Wall, 'O': Pacman
	} cell[30];
struct tree
	{
		int node, vertex[4];
	} n[14];
void gameDisplay()
{
	int i,j,k=0;
	std::cout<<"\n\n\n\n                                                                P A C M A N\n";
	for(j=1;j<=6;j++)
	{
	std::cout<<"                                                ";
	for(i=1;i<=5;i++)
		std::cout << "  _____  ";
	std::cout <<"\n\n";
	std::cout<<"                                                ";
	for(i=1;i<=5;i++)
	{
	   std::cout << " |  "<<cell[k].content<<"  | ";
	k++;
	}
	std::cout <<"\n";
   }
   std::cout<<"                                                  ";
   for(i=1;i<=5;i++)
		std::cout << "  _____  ";
	std::cout <<"\n";
}
int main()
{
	int i,j;
	char ctrl;
   for(i=0;i<30;i++)
	cell[i].content = 'X';  //initialising every cell as a wall
```

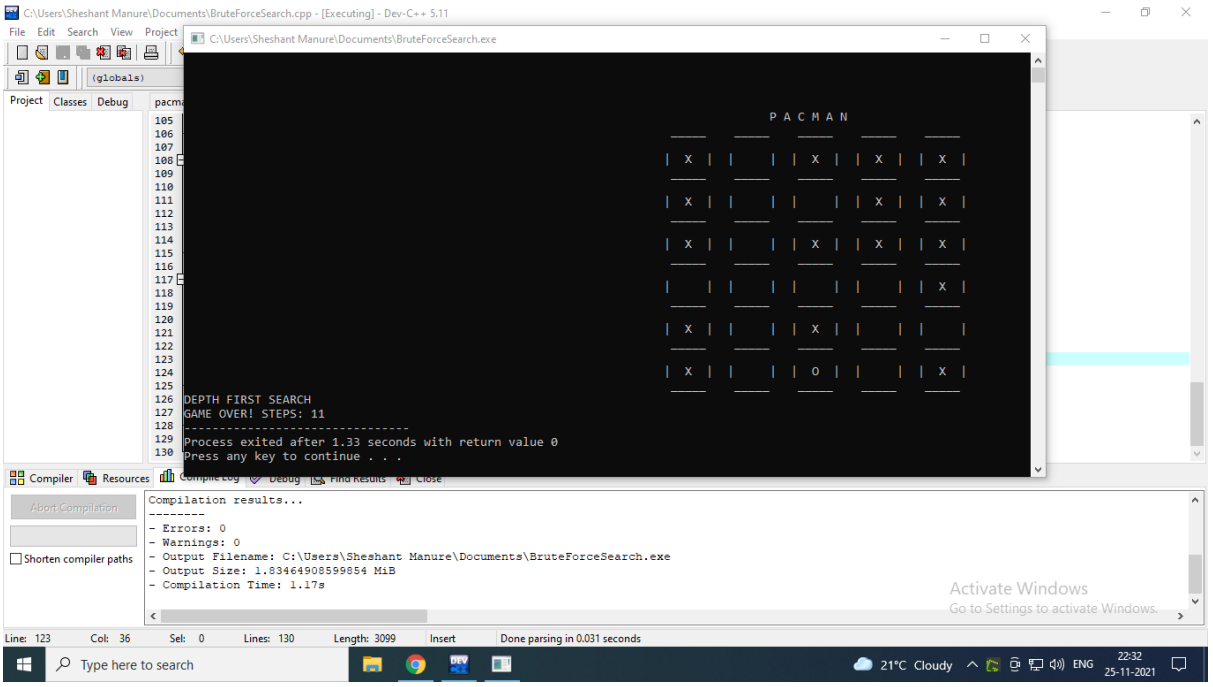
```
	for(i=0;i<14;i++)
	{
		j = empty[i];
		cell[j].content = ' ';
	}
	// Setting boundary conditions of each empty cell
	int l[7] = {7, 16, 17, 18, 24, 27, 28},
	   r[7] = {6, 15, 16, 17, 23, 26, 27},
		u[7] = {6, 11, 16, 21, 23, 26, 28},
		d[7] = {1, 6, 11, 16, 18, 21, 23};

	cell[1].content = 'O'; // Pacman initial position
	cell[27].content = '*'; // Food position setup in cell index number 27
	int pos = 1,k = 0;
	//Implementation of BRUTE FORCE SEARCH
	for(j=0;j<14;j++)
	{
	n[j].node = empty[j];
	for(i=0;i<7;i++)
	{
		if(n[j].node == l[i])
			{
				n[j].vertex[k] = n[j].node - 1;
				k++;
			}
		if(n[j].node == r[i])
		   {
				n[j].vertex[k] = n[j].node + 1;
				k++;
			}
		if(n[j].node == u[i])
		   {
				n[j].vertex[k] = n[j].node - 5;
				k++;
			}
		if(n[j].node == d[i])
		   {
				n[j].vertex[k] = n[j].node + 5;
				k++;
			}
}
```

```
}
//Brute Force Search Algorithm
int steps = 0;
	system("cls");
	gameDisplay();
	std::cout<<"BRUTE FORCE SEARCH\nSTEPS: "<<steps;
	system("pause");
for(i=0;i<14;i++)
{
	if(n[i].node == 27)
	{
		system("cls");
		gameDisplay();
		std::cout<<"BRUTE FORCE SEARCH\nGAME OVER! STEPS: "<<steps;
		break;
	}
	else if(cell[n[i].node + 1].content == '*')
	{
		cell[n[i].node].content = ' ';
		cell[n[i].node + 1].content = 'O';
		system("cls");
		gameDisplay();
		steps++;
		std::cout<<"BRUTE FORCE SEARCH\nGAME OVER! STEPS: "<<steps;
		break;
	}
	else if(cell[n[i].node - 1].content == '*')
	{
		cell[n[i].node].content = ' ';
		cell[n[i].node - 1].content = 'O';
		system("cls");
		gameDisplay();
		steps++;
		std::cout<<"BRUTE FORCE SEARCH\nGAME OVER! STEPS: "<<steps;
		break;
	}
	else
	{
		cell[n[i].node].content = ' ';
		cell[n[i+1].node].content = 'O';
		system("cls");
```

```
                    gameDisplay();
                    steps++;
                    std::cout<<"BRUTE FORCE SEARCH\nSTEPS: "<<steps;
                    system("pause");
                }
        }
        return 0;
}
```

Output



Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Programs\Documents\BruteForceSearch.exe
- Output Size: 1.83464908599854 MiB
- Compilation Time: 1.17s

Phase IV: Solving the Puzzle using Depth First Search Algorithm

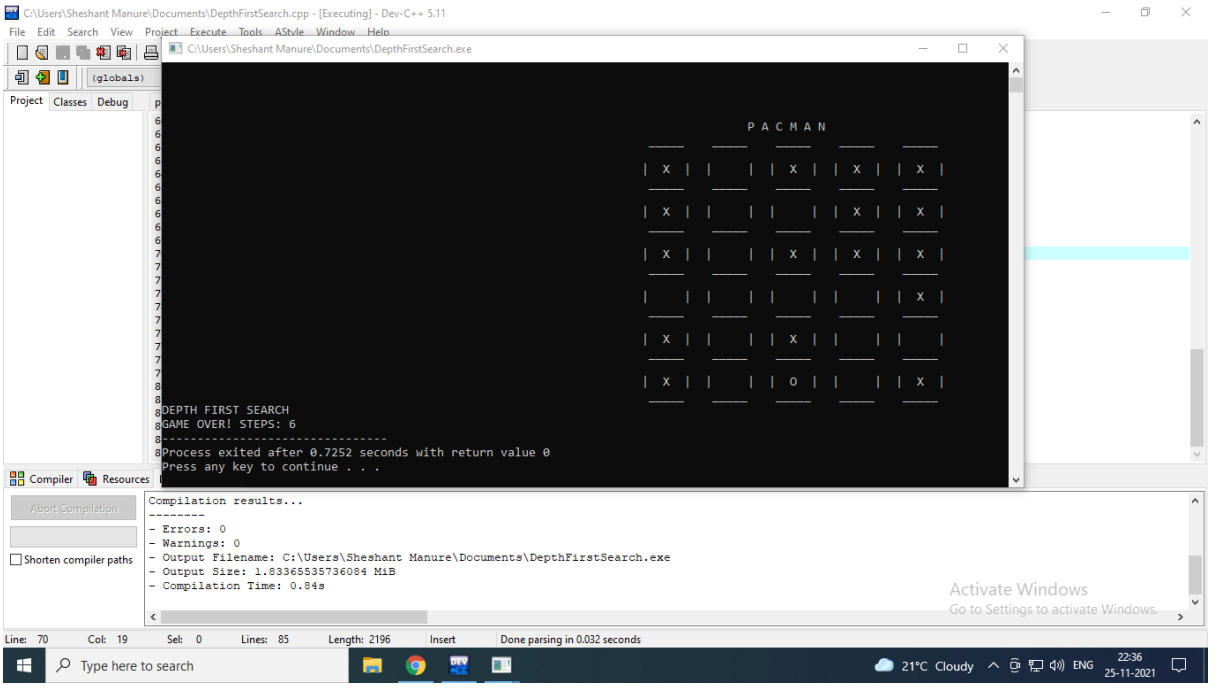
```
#include <iostream>
using namespace std;
int empty[14] = {1, 6, 7, 11, 15, 16, 17, 18, 21, 23, 24, 26, 27, 28};  //index of empty cells;
struct maze
	{
		char content; // ' ': empty, '*': Food, 'X': Wall, 'O': Pacman
	} cell[30];
void gameDisplay()
{
	int i,j,k=0;
	std::cout<<"\n\n\n\n                                                                    P A C M A N\n";
	for(j=1;j<=6;j++)
	{
	std::cout<<"                                                ";
	for(i=1;i<=5;i++)
		std::cout << "  _____  ";
	std::cout <<"\n\n";
	std::cout<<"                                                ";
	for(i=1;i<=5;i++)
	{
	   std::cout << " |  "<<cell[k].content<<"  | ";
	k++;
	}
	std::cout <<"\n";
   }
   std::cout<<"                                                  ";
   for(i=1;i<=5;i++)
		std::cout << "  _____  ";
	std::cout <<"\n";
}
int main()
{
	int i,j;
	char ctrl;
   for(i=0;i<30;i++)
	cell[i].content = 'X';  //initialising every cell as a wall
	for(i=0;i<14;i++)
	{
		j = empty[i];
		cell[j].content = ' ';
```

```
	}
	// Setting boundary conditions of each empty cell
	int l[7] = {7, 16, 17, 18, 24, 27, 28},
	   r[7] = {6, 15, 16, 17, 23, 26, 27},
		u[7] = {6, 11, 16, 21, 23, 26, 28},
		d[7] = {1, 6, 11, 16, 18, 21, 23};

	cell[1].content = 'O'; // Pacman initial position
	cell[27].content = '*'; // Food position setup in cell index number 27
	//Implementing the DEPTH FIRST SEARCH ALGORITHM
	int t=1,steps = 0;
	gameDisplay();
	system("pause");
	while(1)
	{
		if(t == 27)
			{
				system("cls");
				gameDisplay();
				std::cout<<"DEPTH FIRST SEARCH\nGAME OVER! STEPS:
"<<steps;
				break;
			}
		else if(cell[t + 5].content == ' ')
			{
				cell[t].content = ' ';
				cell[t + 5].content = 'O';
				steps++;
				system("cls");
				gameDisplay();
				system("pause");
				t = t + 5;
				if(t+1 == 27 || t-1 == 27)
				{
				system("cls");
				cell[t].content = ' ';
				cell[t+1].content = 'O';
				steps++;
				gameDisplay();
				std::cout<<"DEPTH FIRST SEARCH\nGAME OVER! STEPS:
"<<steps;
```

```
                                    break;
                                    }
                                }
        }
        return 0;
}
```

Output



Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Programs\Documents\DepthFirstSearch.exe
- Output Size: 1.83365535736084 MiB
- Compilation Time: 0.84s

Conclusion

1. **Performance**

	Brute Force Search	Depth First Search
Output Size	1.83464908599854 MiB	1.83365535736084 MiB
Compilation Time	1.17s	0.84s
Score	Steps: 12	Steps: 6

2. **Inference**
   I. The file size is practically the same in both the algorithms
   II. Brute force search consumes more than twice the time required for compilation by the depth first search algorithm.
   III. BFS took twice as many steps as was taken by DFS to finish the puzzle.
   IV. In a case wherein the location of the food was altered in a different position, the BFS surpasses DFS
   V. In conclusion, we can say that since BFS searches each and every possibility one after the other, the time required by the BFS algorithm to solve the puzzle is practically the distance of steps between the pacman and food. Whereas, in DFS, we first look vertically downward to the end of the branch to search for the food.
   VI. The advantage of DFS over BFS is that it consumes comparatively less space. It will reach the goal node in a shorter time period than BFS if it traverses in the right path. It may find a solution without examining much of the search because we may get the desired solution in the very first go.

3. **Shortcomings, drawbacks and scope**
   The algorithm implemented in BFS does not obey the game rule strictly. The diagonal movement was possible. Also, the rule of one step at a time may be broken in between. Also, coordinating the pacman's non-diagonal motion with the DFS algorithm was not achieved. Hence, the algorithm is limited to special cases only. Stack was not utilised (which is normally used) in the DFS program.

References

4. **Dev C++ 5.11** Integrated Development Environment

Source: Internet

Link: https://sourceforge.net/projects/orwelldevcpp/

Dev-C++ is a full-featured Integrated Development Environment (**IDE**). It uses GCC, Mingw or Cygwin as a compiler and libraries set.

5. Brute-force search

   Link: https://en.wikipedia.org/wiki/Brute-force_search

6. Depth-first search

   Link: https://en.wikipedia.org/wiki/Depth-first_search