|  |
| --- |
|  |
| **2D Maze**  **Software Design Document**  Release 1.0 |
|  |

Table of Contents

[1. Introduction 3](#_Toc502268038)

[1.1 Document Purpose 3](#_Toc502268039)

[1.2 Scope 3](#_Toc502268040)

[1.3 Definitions and Acronyms 3](#_Toc502268041)

[2. Overview 3](#_Toc502268042)

[2.1 Purpose of the software 3](#_Toc502268043)

[2.2 Description of the actors/software items/systems that interact with the software 3](#_Toc502268044)

[3. Description 3](#_Toc502268045)

[3.1 Description of Capabilities 3](#_Toc502268046)

[3.2 Interactions with software units 5](#_Toc502268047)

[3.3 Details of algorithms implemented 5](#_Toc502268048)

[3.4 Performance Analysis 7](#_Toc502268049)

[4. Environment 7](#_Toc502268050)

# Introduction

## Document Purpose

This Software Design Document (SDD) describes the 2D Maze Game.

## Scope

This document provides a description of the implementation of the 2D Maze Game.

## Definitions and Acronyms

|  |  |
| --- | --- |
| BFS | Breadth First Search |
| DFS | Depth First Search |

# Overview

## Purpose of the software

The 2D maze is a game application that provides a user interface to display the shortest path to the nearest exit from the user chosen point in maze.

## Description of the actors/software items/systems that interact with the software

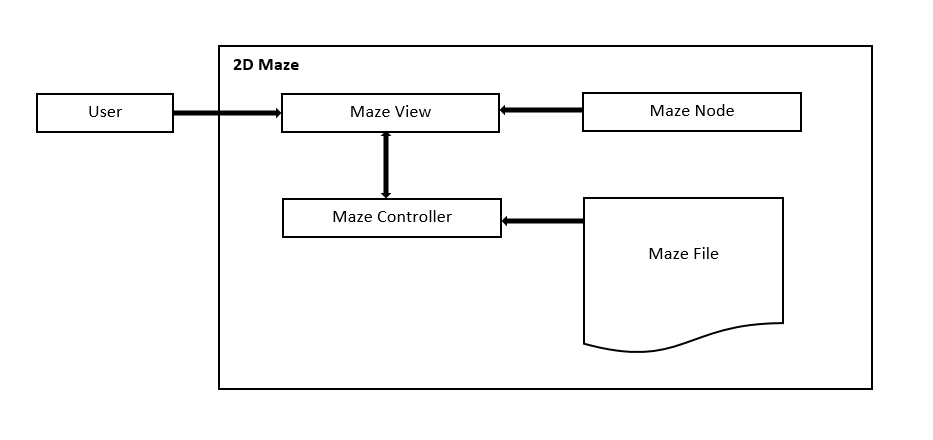
* User: Person who interacts with maze UI to play the game.

# Description

## Description of Capabilities

1. Allows the user to configure a maze input file.
2. Read the input maze file and display the maze in specified size.
3. Allows the user to click in any coordinate in maze.
4. Displays the shortest path from user chosen coordinate to nearest exit.

**Software Structure**



Maze View: A collection of objects that combine to implement the user interface. It provides GUI of the 2D maze that displays the maze matrix. It communicates to maze Controller based on user action. View handle the user events or actions and execute it.

Maze Controller: Maze Controller is responsible to start the main application. It creates and initialize the maze view. It read the maze input file and pass to view.

Maze Node: Implements the graph node that represents each point or coordinate in maze. It holds all the four neighbor nodes and the shortest distance from that node to the nearest exit.

Maze File: Maze file is the input file contains the maze structure or format.

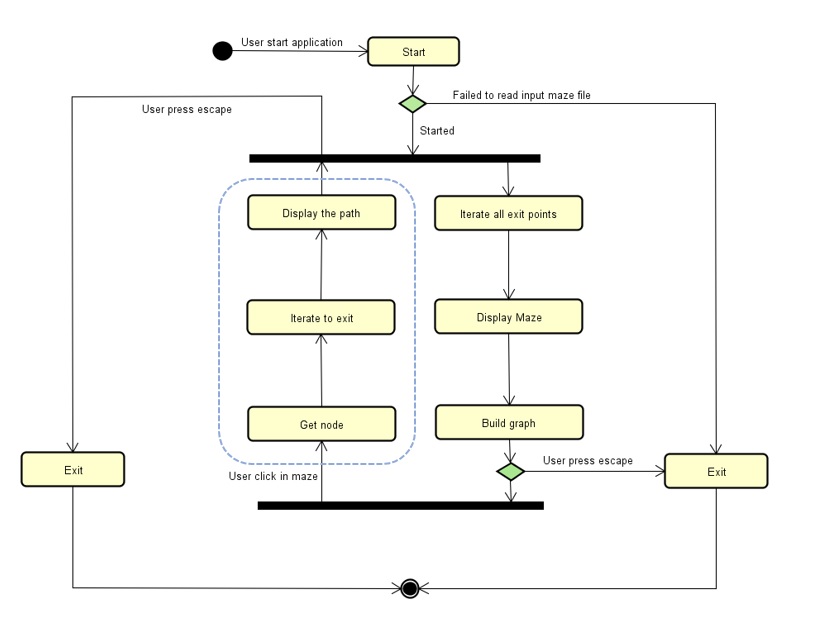
**File format:** All elements in the file are positive integers, commas, spaces and newlines. The maze is considered to be an N x M matrix, with (0,0) as the upper left and (N-1, M-1) as the lower right.

The first line of the file is: N, M\n Where N = number of columns and M = number of rows, followed by a newline character.

Following the maze dimensions are the wall coordinates. Each wall has the following format: X1, Y1, X2, Y2\n where 0 <= X1, X2 < N and 0 <= Y1, Y2 < M. Each wall line is terminated with a newline character.

## Interactions with software units

**Activity diagram of 2D Maze:**



## Details of algorithms implemented

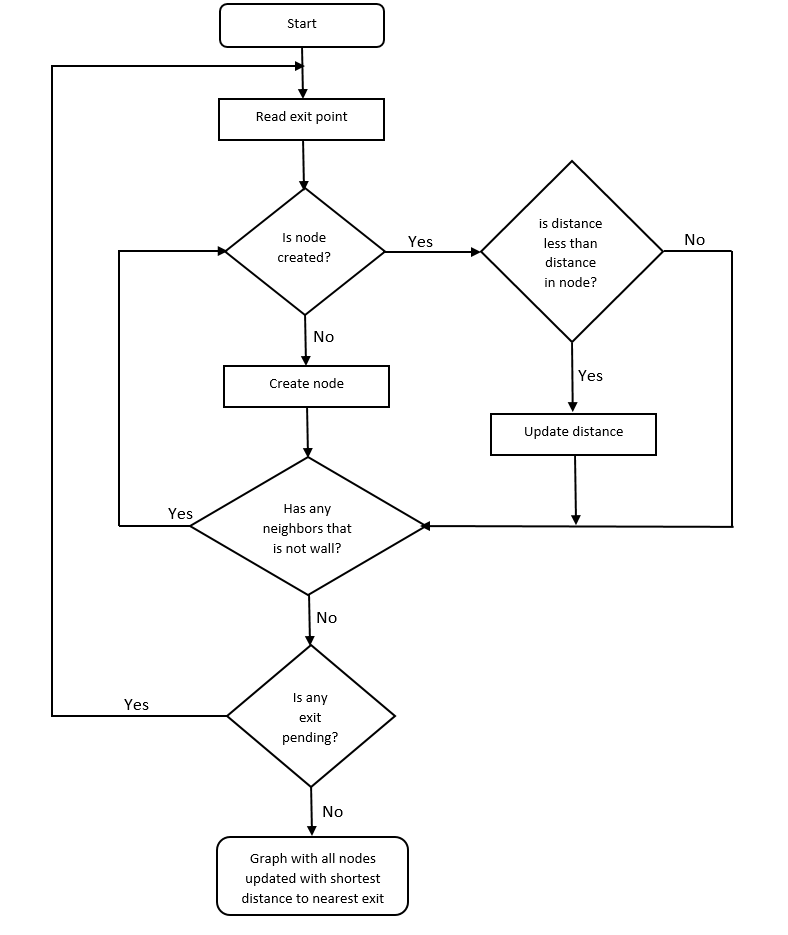
There are so many algorithms which can find paths between two points. There are three algorithms which are easy to implement and understand.

1. **Depth first search (DFS)** - DFS is an [algorithm](https://brilliant.org/wiki/algorithm/) for searching a [graph](https://brilliant.org/wiki/depth-first-search-dfs/(https:/brilliant.org/wiki/graphs/)) or [tree](https://brilliant.org/wiki/trees-basic/) data structure. This algorithm starts at the root node of a tree and goes as far as it can down a given path, and then backtracks until it finds an unexplored path, and then explores it. The algorithm does this until the entire graph has been explored. Depth-first search is a common way that many people naturally approach solving problems like mazes. First, select a path in the maze and follow it until hit a dead end or reach the exit of the maze. If a given path doesn’t work, then backtrack and take an alternative path from a past junction, and try that path. But DFS cannot guarantee an optimal solution.
2. **Breadth first search (BFS)** - BFS is an algorithm that traverses a graph in search of one or more goal nodes. The defining characteristic of this search is that, whenever BFS examines a maze node, it adds the neighbors of that node to a set of node which it will to examine later. In contrast to DFS, these nodes are removed from this set in the order in which they were visited. BFS maintains a queue of nodes which have been visited but not yet examined (an examination of a node consists of visiting all of its neighbors). In our scenario, based on user chosen point need to calculate shortest distance every time.
3. **Dijkstra's algorithm** - Dijkstra's algorithm is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for finding the [shortest paths](https://en.wikipedia.org/wiki/Shortest_path_problem) between [nodes](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) in a [graph](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)). It is similar to BFS, but it can have weights. In this application, the cost of moving from one node to another costs as 1 distance. In Dijkstra's algorithm, it allows us to use any positive integer as the cost, and it can be different for each and every node.

I selected an adaptive approach of Dijkstra's algorithm and BFS to find the shortest path from the user chosen point to nearest exit. Since we have lot of exit points, started from there.

Algorithm pseudo:

1. Iterate all exit points (All exit points start with distance 0 and should be increased by 1 for all neighbor nodes).
2. Check node for this point is already created or not.
   1. if node is not created, then create it and update its distance (weight of node), then recursively iterate neighbor nodes until it reaches a wall.
   2. If node is already created, then update it with shortest distance.
3. Iterate all the path nodes and update the shortest distance value to its nearest exit.
4. Traverse from user chosen node and check which neighbor is just one less than its distance and traverse recursively until the exit.



Node creation is only for points that has path to exit ('#' nodes and ' ' nodes surrounded by only walls are ignored during graph creation).

when user click on a wall ('#'), then will check neighbor nodes that has shortest path, then highlight that path. If any user chosen node is surrounded by four wall nodes, then won't display any path, since it has no path to exit.

## Performance Analysis

**Time Complexity:**

Display maze – O(MN) where M- Rows, N- Columns.

Build Graph – O(E Log N) where E – Exit nodes, N- Neighbor path nodes.

Find Nearest Path – O(D) where D – Distance value in that node.

**Space Complexity:**

Maze matrix – O(MN) where M- Rows, N- Columns.

Graph Nodes – O(N) where N is number of nodes that has at least one path to exit.

Exit Nodes – O(E) where E is number of exit nodes.

# Environment

The application is developed under windows platform using C++ technology.