Data Structures Project

Project Group Members

Affan Wajid
Zubair Khalid

Fall 2023

Department of Computer Science

FAST – National University of Computer & Emerging Sciences

Table of Contents

Introduction

In our Data Structures 2023 semester project, we applied our acquired knowledge of data structures and algorithms to create a console-based race-car game. Our goal was to showcase a comprehensive understanding of the concepts we had learned throughout the semester. As a collaborative effort, we as a pair developed the logic and code for a complete console-based game, incorporating various data structures such as an integrated graph with Dijkstra algorithm optimization, linked lists, queues, and binary search trees.

Features and Functionality

In accordance with the project's expected features, the finished deliverable includes a randomized map that changes every time a new game is played, two modes of play i.e automatic and manual, collectables and obstacles for an added layer of challenge, three difficulty settings, a scoring system that calculates score based on factors such as the time taken by the player and the number of collectables obtained, a leaderboard for saving user's playing history, appealing aesthetic and design choices to improve the playing experience and increase the readability of the text based user interface style of the game, and user friendly menus for easy navigation of the game's features and functions. All implemented using appropriate data structures.

Challenges

In the process of working on this project, we faced difficulties with figuring out how to divide work evenly between us and how to collaborate on coding in the most optimal way. A solution that both group members agreed on was to work individually on the project, and then combine different parts of the logic of each others' work into a cohesive, best-case implementation.

Creating a graph that is both processable and visually representable posed a significant challenge. The implementation of the graph, along with programming algorithms for finding the shortest paths within it, proved to be the most difficult aspect of the coding process, requiring considerable time and effort to develop

the logic. Affan played a pivotal role in crafting this solution. A unique challenge we encountered was designing a comprehensive maze-like structure for the map, ensuring it appeared cohesive with clear paths between nodes while incorporating engaging and entertaining game elements to make traversal between these nodes a fun experience. These issues took considerable effort to resolve, requiring a lot of problem-solving centric thinking.

Explanation of Classes

1. Class int_Node

This class was used to make nodes for the List class. Main Purpose of this was to have data,weight be stored in the Node.

2. Class List

This class is used mainly for creating adjacency List for our Nodes in our graph by storing them in a list. It also consists of functions for randomizing weights which are used to make a random path using Dijkstra's Algorithm.

3. Class Node

This class is used to store the data of the Node in the graph and also their respective adjacency lists and also their type since each Node of the graph represents either a powerup,obstacle,wall,points,player,etc. It also has a function to print the Node with respect to its type representation like wall is printed as # .

4. Class QueueNode

This class is used to make a queue for the Node object in our game. By using this Queue, we placed the obstacles and powerups in random locations for our game.

5. Class ListNode

This class is used to create a list of nodes for our graph. Functions are provided to change the weights of nodes in the adjacency list in this class.

6. Class Graph

This class contains the List of Nodes in the graph,the size of the map to be made for the game, the prev string which is used to store the prev object type which player stepped on (*Used to replace the original value when player go in another Node. For example when the player is at start, he goes to the right of the start and then the previous value where the player was at is replaced with its original type "start*"). Graph is used for the main representation for the whole maze.

Main functions for class graph are as follow:

- Reset() | It resets the map to its original NULL values
- Generate() | The generate function is used to generate the graph by giving each node an int representation (*1,2,3….. to n*n where n is the size of the map*) which can be used to print the graph like an array and display it on the screen easily.
- randomize_weight() | It randomizes the weights for all of the nodes of the graph. It is used to help dijkstra make random paths for our graph/map.
- print_map() | It is used to print the map to the user for the game.
- add_obstacles() | Adds obstacles in the graph in random positions using a queue.
- add_powerup() | Adds powerups(50 points) in the graph in random positions using a queue.
- add_powerup2() | Adds powerups(2 x Score) in the graph in random positions using a queue.
- create_path() | Used to create a path using the path which dijkstra provides
- set_end_point() | Sets end point of the map which is the bottom left side
- set_start_point() | Used to set the start point of the map to top left side

- player_movement() | Contains the Logic for collision detection and also increase/decrease of score based on collision with type of object and input by user for movement.
- path_finder() | Used to ask user for start and stop positions for automatic mode and prints the shortest path automatically using dijkstra's algorithm

7. Class BSTNode

It is used to store the player name and player score in the BST.

8. Class BST

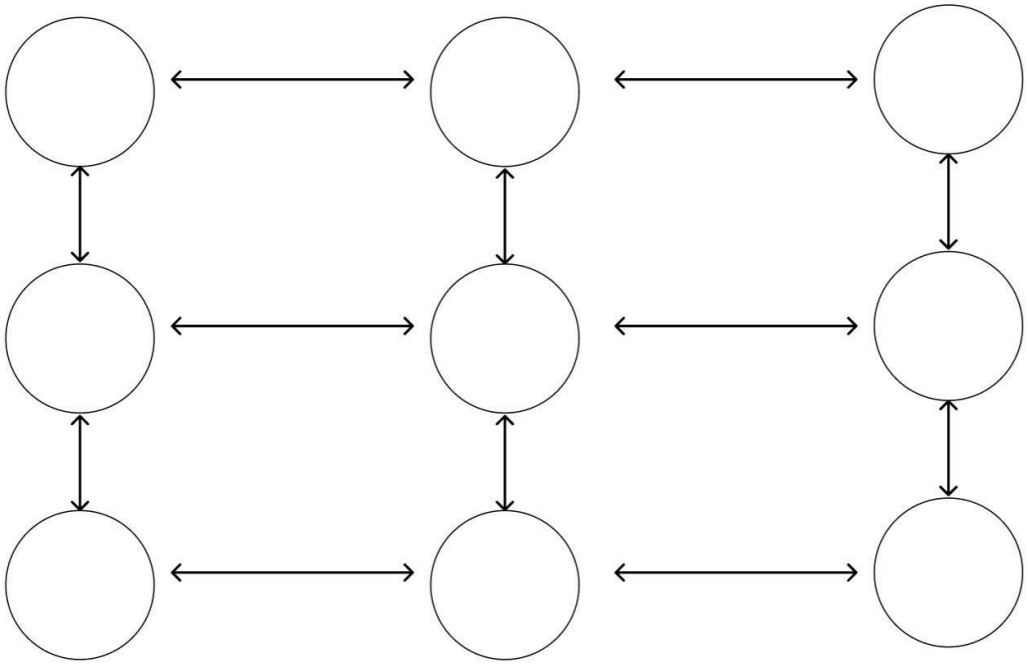
It is used to store the leaderboard of the game in a BST so that players' scores could be stored efficiently.

9. Class Game

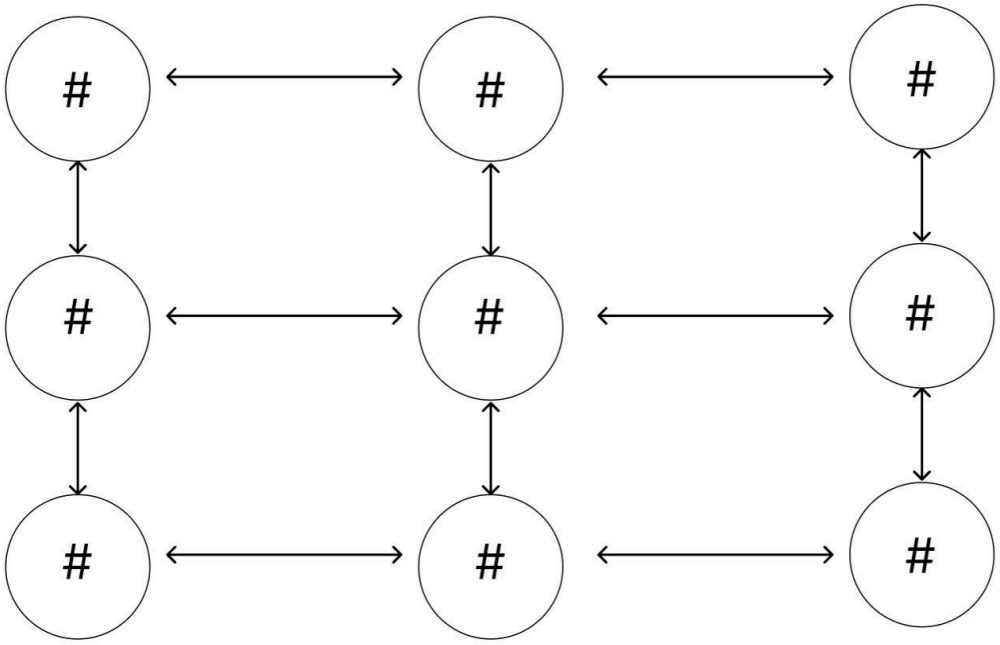
This class contains the graph which represents the map and the BST which represents the leaderboard of the game. It is used for making the game loop, saving the leaderboard, updating the leaderboard and giving the player the ability to use auto mode.

Explanation for Map Generation

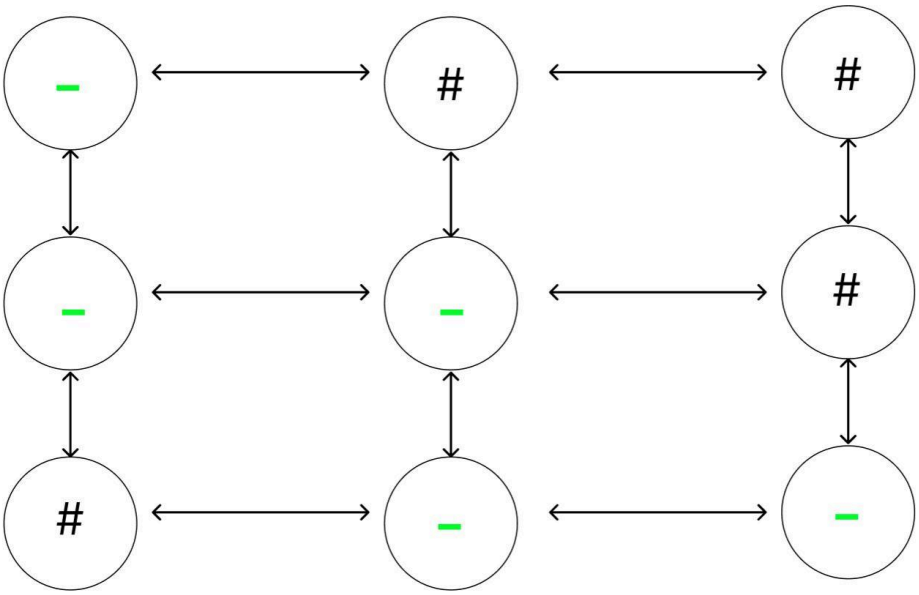
So for the map representation we have used a graph. First we give the relevant map size of the graph. Let's say for this case we give the map size 3. Then we make a map which is 3x3 but in the graph we just created 9 nodes which are connected like this:



We have not given them weights yet and their default types are a wall as follows:

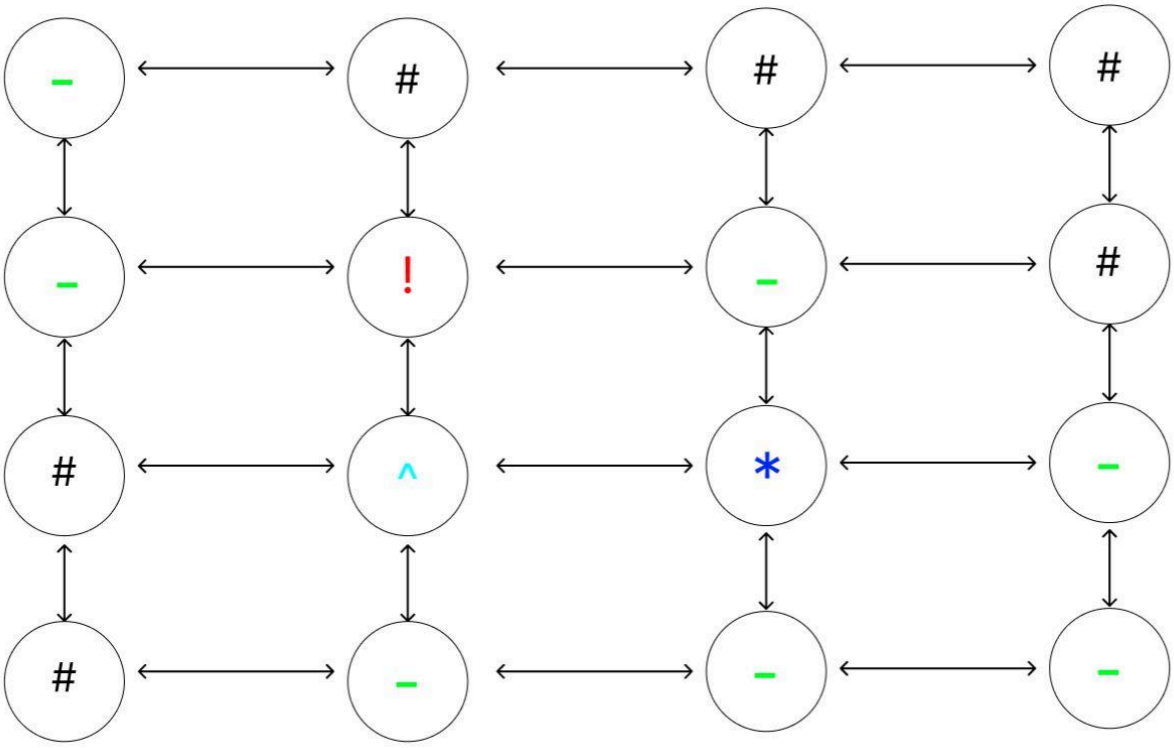


Then random weights are given to the graphs and dijkstra is applied 5 times to create a path from top left to top bottom of the map.

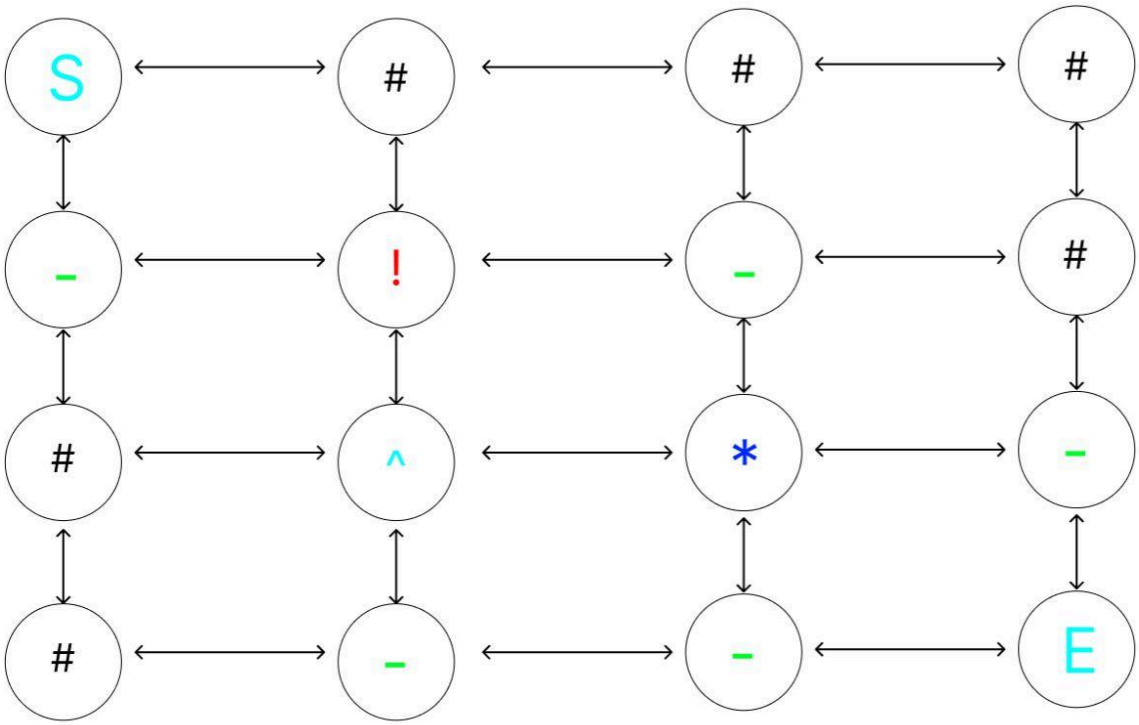


As an example above, let's say we applied a dijkstra to find a path from top left to bottom once. After doing that we set every node that was in that path to the type path which is represented by “-”.

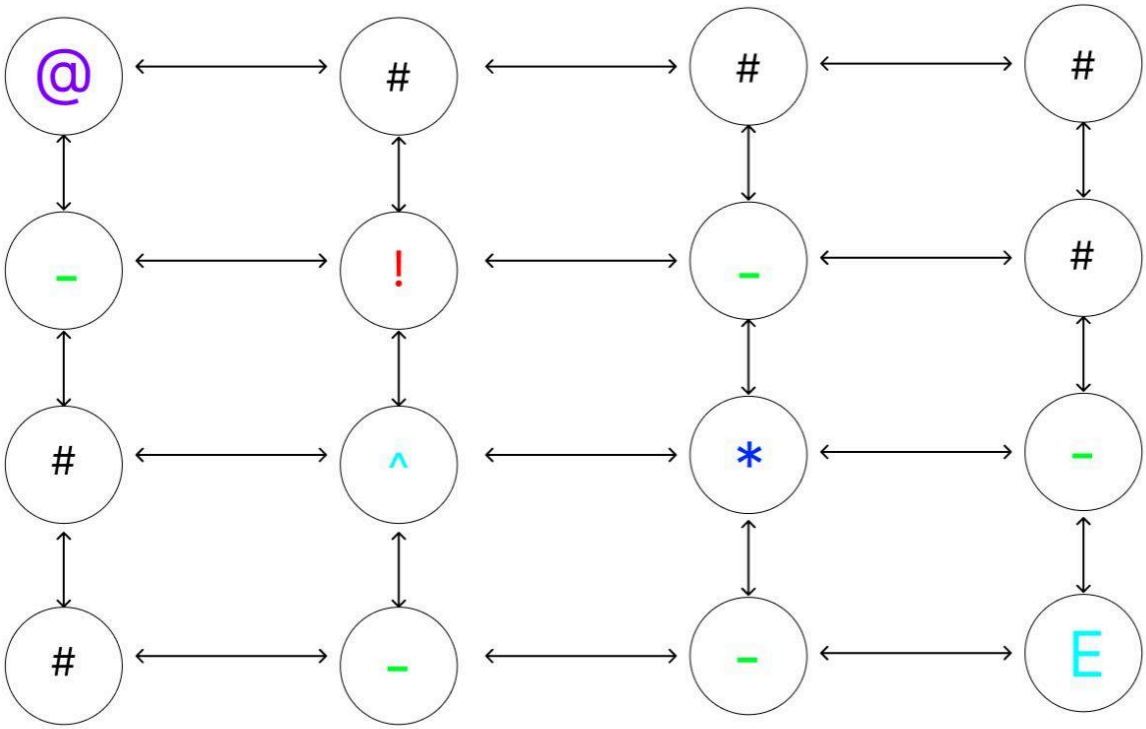
After that, by using a queue we add powerup,points,obstacles into our game only in those points where there exists a path and we have also ensured that the elements that we add don’t get placed in the position of the start and end points of our game. After adding the obstacles,powerups,points,obstacles the graph would look something like this:



After that we add the start and end points in the top left and bottom right respectively:



Now we place the player at the starting position:



After this the game loop starts and a map is shown to the player.

Project Overview

This project served as a great learning experience to help us clarify our concepts of data structures. It encouraged team-based thinking and collaboration, and posed a strong technical challenge. Overcoming this challenge helped us strengthen our coding skills and develop a better understanding of the practical usage of data structures in real-world coding. All of these qualities serve as significant tools in developing our software engineering expertise for the future.