# **Data Structures Semester Project**

Race Car Game

Course Instructor

**Ma’am Shahela Saif**

**By**

Mohammad Ibraheem (22i-2603)

Ibrahim Ahmed (22i-8781)

5th December, 2023

Fall 2023



**Department of Software Engineering**

F.A.S.T. – National University of Computer & Emerging Sciences, Islamabad

# 

[**Data Structures Semester Project 1**](#_j48j3cyaoqb4)

[**Document for Code Review and Documentation 3**](#_vdoxq5siafoa)

[**Problem Statement 3**](#_n6bm49k1dech)

[**Our Approach 3**](#_u3ghrdsreea4)

[**Data Structures Used 3**](#_ezibmnzatwu5)

[● List: 3](#_xstyatjf8dx)

[● Queue: 4](#_mdrm1mckoi1y)

[● Binary Search Tree: 4](#_tgxdakjgyfnn)

[● Graphs: 4](#_tj26q9btwfc3)

[**Introduction to our Code 4**](#_ag6mmoglyn6a)

[Part One: Map Class 4](#_oqvkxn6u1zoe)

[Class MapVertex 4](#_56ihtcmqyhx3)

[Class Map 5](#_mba5hu3e4atf)

[Part Two: Game Class 5](#_mkiflb133ro5)

[Class Collectible 5](#_nwotx9b769sw)

[Class Game 6](#_khkowrmbp68m)

[**Conclusion 6**](#_ezfxt4j0ay6b)

# 

# Document for Code Review and Documentation

# 

# Problem Statement

Developing a 2D console-based race car game using C++ while utilizing Data Structure components. Implement a text-based user interface within the console (A text-based user interface (TUI) relies on characters and text to convey information instead of graphical elements. In a console-based game, a TUI uses ASCII characters and text to represent various game elements). The game also features a player-controlled race

cars, obstacles, tracks, and a scoring system.

# Our Approach

* Create a racetrack out of graphs by utilizing GraphNode and LinkedList.
* Using the random edge-making function, we created tracks.
* The graph denoted by "#," "S," and "E" . Their symbolic key is “ S: Start, E: End, #: Obstacle, P: Player $: Coin \*: Normal Path“
* The Binary Search Tree is used to maintain the leaderboard and score system.
* The user is presented with a menu in Game Class, where the menu is created and maintained.
* For scoring passing through obstacles you score will be divided by 2 and passing through coins will get you 100 points added to the total score and passing through normal paths will get you 10 scores.

# Data Structures Used

## List:

A list is used to maintain an adjacency list of nodes in graphs.

## Queue:

A queue in the map to maintain obstacles.

## Binary Search Tree:

BST to maintain the score and handle it. Write into the file read from the file and show the leaderboard. (Not completely implemented, but constructed classes)

## Graphs:

To maintain nodes and vertices of the map to make a track for the car.

# Introduction to our Code

This document comprehensively outlines the structure, organization, and functionality of the provided C++ code for a 2D console-based race car game. The code is divided into multiple parts, each addressing specific aspects of the game development, including map representation, player controls, obstacles, scoring, user interface, and file handling using a Binary Search Tree (BST).

## Part One: Map Class

### Class MapVertex

**Purpose:**

Represents a vertex in the game map.

Stores information about the vertex, including its type, visit status, and adjacency list.

**Attributes**:

ver: Vertex number.

type: Type of the vertex ( 'S', 'E', '$', '\*', '#', 'P').

visited: Boolean indicating whether the vertex has been visited.

adjLst: Adjacency list storing pointers to connected vertices.

**Methods:**

getVer(), getType(), getVisit(): Getter methods for attributes.

setVer(), setType(), setVisit(): Setter methods for attributes.

getAdjLst(): Returns the adjacency list.

addAdjLst(): Adds a vertex to the adjacency list.

### Class Map

**Purpose:**

Represents the game map.

Handles the creation of paths, display of the map, and player movement.

Attributes:

**Vertices:**

The Array of pointers to MapVertex represents the game map.

width and height: Dimensions of the map.

obstacles: Queue for managing obstacle generation.

score: Game score.

**Methods:**

makePaths(): Creates paths between vertices.

showConn(): Displays adjacency lists for each vertex.

makeRand(): Randomly assigns types to vertices.

displayMap(): Displays the game map.

manGame(): Handles manual gameplay.

addPath(), checkAdjLst(), checkFood(): Helper methods.

getTotalObs(), getScore(): Accessor methods.

## Part Two: Game Class

### Class Collectible

**Purpose:**

Represents a collectible item in the game.

Stores information about the item, including its value, location, and type.

**Attributes:**

value: Value of the collectible.

location: Pointer to the map vertex where the collectible is located.

type: Type of the collectible ('$', '\*', '#', 'P').

**Methods:**

getValue(), getLoc(), getType(): Getter methods.

setValue(), setLoc(), setType(): Setter methods.

### Class Game

**Purpose:**

Manages the overall game, including manual and automatic gameplay, player history, and score tracking.

**Attributes:**

pName: Player name.

collection: List of Collectible objects.

score: Player score.

lvl: Game level.

Map: Instance of the Map class.

TreeObject: Instance of the BST class for file handling.

**Methods:**

Menu(), Manual(), Auto(), history(), showScoreBoard(): Game management methods.

ReadFromFile(), WriteToFile(): File handling methods.

# Conclusion

This document comprehensively overviews the code structure, organization, and functionality. Using a Binary Search Tree, it covers the game map representation, player controls, scoring, user interface, and file handling. The code is well-commented, making it easy to understand and maintain.