

Car Race

Game

PROJECT

Submitted by:

Umar mehmood 22I-2617

SULEMAN AHMADZAI 22I-8807

Table of Contents

[1. Overall Description: 1](#_Toc152712034)

[2. Overview 1](#_Toc152712035)

[3. Source Code Structure 1](#_Toc152712036)

[4. Gameplay Mechanics 1](#_Toc152712037)

[4.1. Manual Mode: 2](#_Toc152712038)

[4.2. Automatic Mode: 2](#_Toc152712039)

[5. Gameplay Features 2](#_Toc152712040)

[5.1. Scoring System 2](#_Toc152712041)

[5.2. Coin Collection 2](#_Toc152712042)

[5.3. Milestones 2](#_Toc152712043)

[5.4. Leaderboard 2](#_Toc152712044)

[5.5. File Handling 2](#_Toc152712045)

[6. Dependencies 2](#_Toc152712046)

[6.1. Platform Dependency: 3](#_Toc152712047)

[6.2. Limited User Interface: 3](#_Toc152712048)

[7. Future Enhancements 3](#_Toc152712049)

[8. Classes: 3](#_Toc152712050)

[8.1. MazeGame Class 3](#_Toc152712051)

[8.1.1. Role and Functionality: 3](#_Toc152712052)

[8.1.2. MazeGame(int choose): 3](#_Toc152712053)

[8.2..2. playGame(): 3](#_Toc152712054)

[8.2.3. printMaze(): 3](#_Toc152712055)

[8.2. CoinsAndMileStoneManagement Class 3](#_Toc152712056)

[8.2.1. Role and Functionality: 3](#_Toc152712057)

[8.2.2. Methods: 3](#_Toc152712058)

[8.3. Vertex, Edge, EdgeList 4](#_Toc152712059)

[8.3.1. Vertex: 4](#_Toc152712060)

[8.3.2. Edge: 4](#_Toc152712061)

[8.3.3. EdgeList: 4](#_Toc152712062)

[8.4. Data Structures 4](#_Toc152712063)

[8.4.1. Stack: 4](#_Toc152712064)

[8.4.2. Queue: 4](#_Toc152712065)

[8.5. CustomList: 4](#_Toc152712066)

[8.6. Node Classes (e.g., TreeNode): 4](#_Toc152712067)

[9. Conclusion 4](#_Toc152712068)

[9.1. Integration of Complex Data Structures and Algorithms 4](#_Toc152712069)

[9.2. Advanced Game Mechanics and User Interaction 5](#_Toc152712070)

[9.3. Score and Milestone Management 5](#_Toc152712071)

[9.4. Dynamic Maze and Pathfinding Implementation 5](#_Toc152712072)

[9.5. Challenges and Future Enhancements 5](#_Toc152712073)

[9.6. Educational and Practical Implications 5](#_Toc152712074)

[9.7. Final Thoughts 5](#_Toc152712075)

# Overall Description:

The Project was to make a Text Based Car Racing Game for entertainment. We have implemented several features to make the Car Racing Game interactive and properly working according to the requirements and constraints defined by the instructors. The Car Race Game starts by showing a “Main Menu” having four options including “Play Game”, “Instructions”, “MileStones”, and “To Exit”.

The Selection of Play Game option takes the user (player) to another menu which gives the user two more options to choose, “Manual” and “Automatic”. If the user chooses the Manual option, a randomly generated maze is printed and the user can now move the car across the maze while the score and the distance are printed on the console until one of the following conditions are met:

* Player reaches the destination and the game is over.
* Player’s score decreases and becomes less than zero and the game is over.
* Player presses q to quit and the user comes back to main menu.

In either of the above cases the player comes back to the main menu.

Now, if the user chooses Play Game and then chooses to play automatically, again a randomly generated maze is printed on the console but this time the program will not wait or take any input from the User instead the program starts to move the car from the source to the destination using Dijkstra algorithm to find shortest path and then move the car over it.

Now, if the user selects to see the instructions, the game shows the basic representations and the keys to move the car.

Next the user selects to see the MileStones, MileStones are like Trophies or Achievements of our Game; so the program will show all the MileStones and tell that which one is achieved and which is not.

The User selects to exit and the program ends.

# Overview

This document provides details about a car racing game developed in C++. The game implements a maze-like race track where the player navigates a car through various obstacles to reach a destination. It features manual and automatic play modes, scoring, coin collection, and milestone achievements.

# Source Code Structure

The source code consists of multiple classes representing different aspects of the game, such as the game environment, obstacles, scores, coins, and milestones. Key classes include:

# Gameplay Mechanics

The maze is represented as a grid where each cell can have different properties ('X' for obstacles, 'P' for power points, $ for coins, and 'D' for the destination).

The player moves a car through the maze, aiming to reach the destination while avoiding obstacles and collecting power-ups and coins.

The game uses a basic physics model for movement and collision detection.

User Interaction and Controls.

* 1. Manual Mode:

Players use keyboard inputs (W, A, S, D) to move the car in different directions.

* 1. Automatic Mode:

The game automatically calculates the shortest path to the destination and moves the car along this path.

The main menu offers options to start the game, view instructions, check milestones, and see the leaderboard.

# Gameplay Features

## Scoring System

Players earn points by moving towards the destination and collecting power points. The score is reduced for hitting obstacles or making invalid moves.

The CoinsAndMileStoneManagement class keeps track of the scores and provides functionality to increment or decrement scores based on the player's actions.

## Coin Collection

Coins are scattered throughout the maze. The player can collect these to increase their score.

The class manages the coin count and updates it as the player collects coins in the game.

## Milestones

Specific achievements or goals are set as milestones (e.g., achieving a certain score within a limited number of moves).

The game tracks these milestones and updates their status when achieved by the player.

## Leaderboard

A leaderboard is maintained to rank players based on their scores.

This feature uses a tree data structure to store and sort the scores efficiently.

Technical Details

## File Handling

The game uses file I/O operations to save and load game data like scores, coins collected, and milestones achieved.

This ensures that player progress is retained between sessions.

# Dependencies

The game requires the C++ Standard Library and standard input/output facilities.

It uses specific functions from <termios.h> and <unistd.h> for reading key presses, making it platform-dependent (primarily for UNIX-like systems).

Limitations and Known Issues.

* 1. Platform Dependency: The game is currently tailored for UNIX-like systems due to specific system calls.
  2. Limited User Interface: The game uses a console-based interface, which might not be as visually appealing as GUI-based games.

# Future Enhancements

Porting the game to multiple platforms.

Introducing a graphical user interface for enhanced player experience.

Adding more levels, obstacles, and power-ups to increase game complexity.

# Classes:

## MazeGame Class

### Role and Functionality:

This is the central class of the game, acting as the primary controller for game mechanics.

It orchestrates the overall game flow, including initializing the maze, processing player inputs, and managing game states.

Methods:

### MazeGame(int choose):

The constructor sets up the game environment. It accepts an integer choose to determine the game mode (manual or automatic). It calls initializeMaze to create the maze and dijkstra for pathfinding in automatic mode.

### 8.2..2. playGame():

Contains the game loop, where it processes user inputs for movement and updates the game state, including checking for collisions and updating scores. It also renders the maze after each move.

8.2.3. printMaze(): This method handles the visual representation of the game state on the console, showing the maze, player position, and scores.

## CoinsAndMileStoneManagement Class

### Role and Functionality:

Manages scoring, coin collection, and milestone tracking.

It is responsible for updating and maintaining the player's current score and the number of coins collected.

### Methods:

Scoring Methods (incrementscore(), decrementscore()): These methods adjust the player's score based on game events, like collecting power points or hitting obstacles.

Milestone Tracking (checkMilestone()): This method checks if specific milestones, such as reaching a certain score within a limited number of moves, have been achieved.

## Vertex, Edge, EdgeList

### Vertex:

Represents a single cell in the maze.

Holds details like the cell's type ('X' for obstacles, 'P' for power points, etc.) and its position in the maze.

### Edge:

Represents a connection between two Vertex objects, essentially a pathway in the maze.

Contains information like the destination vertex and weight (cost) of moving through this edge.

### EdgeList:

A collection of Edge objects for a given Vertex.

It is used to keep track of all possible moves from a particular cell in the maze.

* 1. Data Structures: Stack, Queue, CustomList, and Node Classes

### Stack:

A traditional stack data structure implemented using linked lists.

Used in pathfinding algorithms to backtrack the path from the destination to the source.

## Queue:

A standard queue data structure also implemented using linked lists.

Utilized in various parts of the game for managing elements in a first-in, first-out manner.

## CustomList:

A specialized list used primarily in Dijkstra's algorithm for maintaining a list of vertices sorted by their distance from the source.

## Node Classes (e.g., TreeNode):

These are supportive classes used to create nodes for the data structures.

TreeNode, for instance, is used in the binary tree for managing the leaderboard, allowing efficient insertion and retrieval of player scores

# Conclusion

## Integration of Complex Data Structures and Algorithms

The car racing game demonstrates an intricate integration of data structures and algorithms to create a compelling gameplay experience. The use of stacks, queues, trees, and custom lists underscores the game's complexity and the thoughtful design behind its mechanics. These structures not only facilitate efficient game state management but also enhance features like pathfinding, leaderboard tracking, and dynamic maze generation.

## Advanced Game Mechanics and User Interaction

At the heart of the game is the MazeGame class, which serves as the central orchestrator. It successfully combines the logical aspects of the game with user interaction, balancing the challenge of navigating through a maze with intuitive control mechanisms. The game's dual modes—manual and automatic—offer varied gameplay experiences, catering to different player preferences and skill levels.

## Score and Milestone Management

The CoinsAndMileStoneManagement class adds a layer of depth to the game, introducing an element of strategy and goal orientation. By managing scores, coins, and milestones, it provides players with objectives beyond merely reaching the destination. This class enhances the replay value of the game, encouraging players to improve their scores and achieve new milestones in subsequent playthroughs.

## Dynamic Maze and Pathfinding Implementation

The implementation of the maze using Vertex, Edge, and EdgeList classes showcases the game's dynamic nature. Each game session presents a unique maze configuration, making every playthrough distinct and challenging. The incorporation of Dijkstra's algorithm for pathfinding in the automatic mode is a testament to the sophisticated level of programming involved, offering players an insight into the underlying complexity of game development.

## Challenges and Future Enhancements

While the game stands as a robust and engaging project, it also opens avenues for future enhancements. The current platform dependency and console-based interface present opportunities for development into a cross-platform, graphical interface game. Such advancements would not only widen the game's accessibility but also enrich the user experience.

## Educational and Practical Implications

From an educational perspective, this game serves as an excellent example of applying theoretical computer science concepts in a practical, entertaining product. It provides a real-world application scenario for data structures and algorithms, making it a valuable learning tool for students and aspiring game developers.

## Final Thoughts

In conclusion, this C++ car racing game is a blend of technical prowess, creative game design, and user-centric interaction. It reflects a deep understanding of programming concepts and their application in creating enjoyable and challenging gameplay. As an evolving project, it holds the potential for further enhancements, setting the stage for more advanced features and an even more immersive gaming experience.

