Project Report: Rush Hour Taxi Game (Assembly Language)

Student Name: Abdul Ghafoor

Roll No: i240118

Section: AI-B

Course: Computer Organization & Assembly Language

Submission Date: December 3, 2025

---

1. Introduction

The "Rush Hour Taxi" project is a console-based simulation game developed in x86 Assembly Language using the Irvine32 library. The objective of the game is to simulate the experience of a taxi driver navigating a busy city grid. The player must pick up passengers and drop them off at specific destinations while avoiding obstacles, traffic, and pedestrians.

The project demonstrates proficiency in low-level programming concepts, including memory addressing, array manipulation, procedure calls, stack management, and interacting with Windows API libraries (winmm.lib) for audio playback.

2. Game Features

The game implements the requirements specified in the project guidelines, offering a dynamic and interactive user experience:

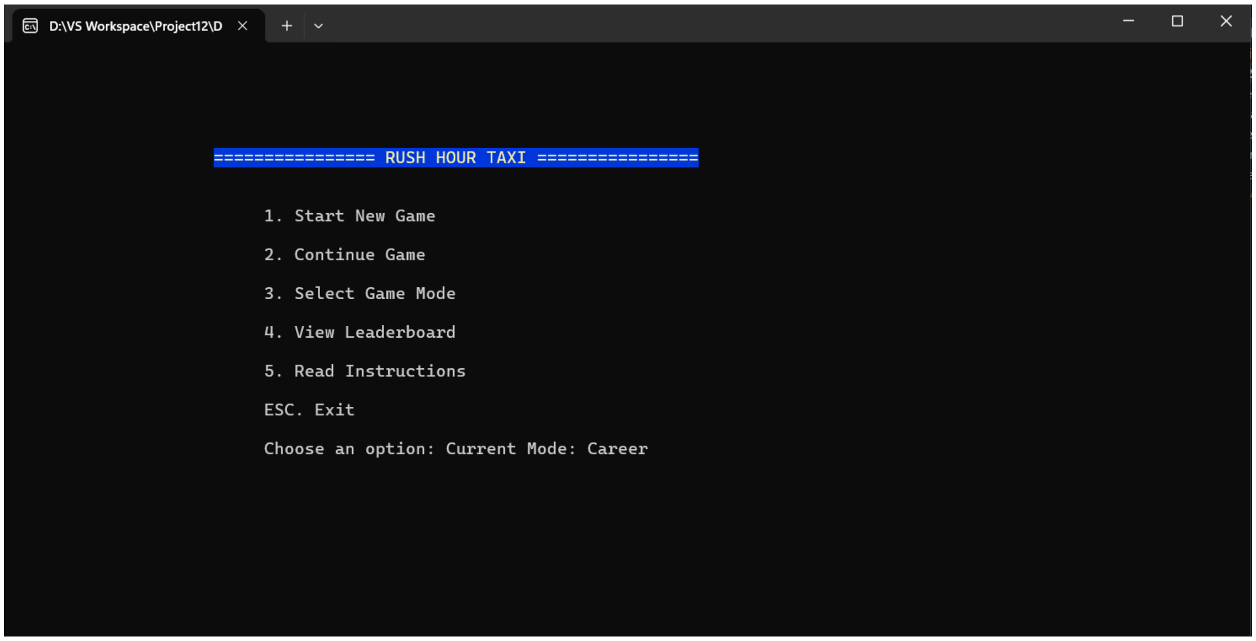
- **Game Modes:**
  - **Career Mode:** Standard gameplay focused on accumulating score.
  - **Time Mode:** A high-pressure mode with a 60-second countdown timer.
  - **Endless Mode:** Continuous play without a win condition based on passenger count.
- **Taxi Selection:**
  - **Yellow Taxi:** Moves faster (low delay) but incurs higher point penalties for collisions.
  - **Red Taxi:** Moves slower (high delay) but incurs lower penalties for collisions.
  - **Random Selection:** The computer randomly assigns a taxi type to the player.

- **Dynamic Traffic System:** NPC cars navigate the grid autonomously. If they encounter an obstacle, they reverse or change direction.
- **Scoring System:**
    - **+10 Points:** Successful drop-off.
    - **+10 Points:** Collecting Bonus Gems.
    - **Penalties:** Specific deductions for hitting trees, cars, or people (varying by taxi color).
- **Audio Effects:** The game utilizes PlaySound to provide auditory feedback for menu selection, pickups, drop-offs, crashes, and game-over states.
- **Difficulty Scaling:** Every 2 successful drop-offs, the game speed increases (delay decreases), making traffic and movement faster.

3. User Interface (UI) Design

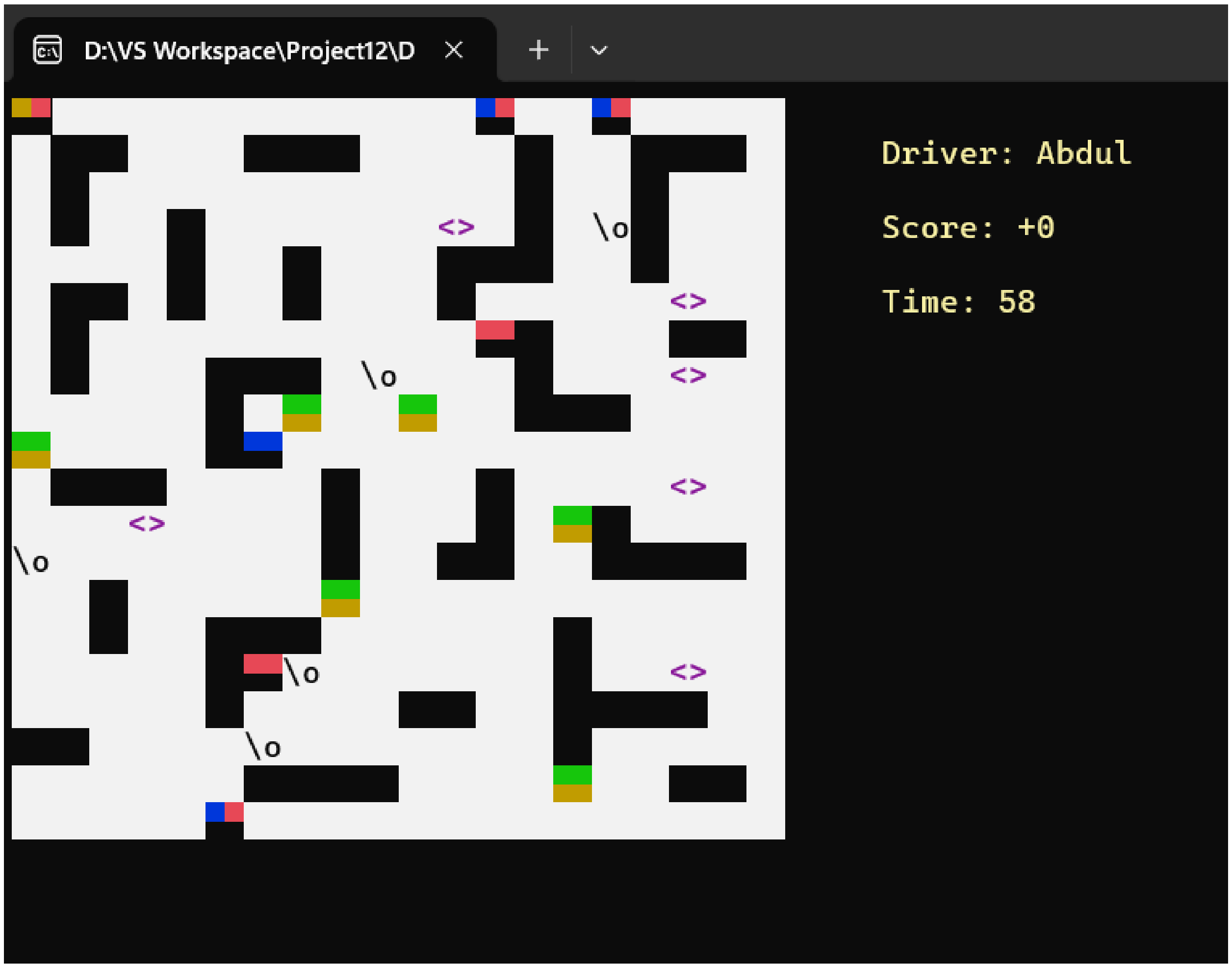
The game utilizes a 20x20 grid layout rendered using ASCII characters and color attributes.

- **The Board:**
    - **Roads:** White background (Safe for travel).
    - **Buildings:** Black blocks (Impassable walls).
    - **Trees:** Green symbols (Obstacles).
- **Entities:**
    - **Player Taxi:** Yellow or Red block (depending on selection).
    - **Passengers:** Represented by \o (waving figure).
    - **Drop-off Zone:** Highlighted in Green or Red indicating the target destination.
    - **Traffic:** NPC cars represented by colored blocks moving across the screen.
    - **Gems:** Bonus items represented by <>.



```
D:\VS Workspace\Project12\D

================ RUSH HOUR TAXI ================

    1. Start New Game

    2. Continue Game

    3. Select Game Mode

    4. View Leaderboard

    5. Read Instructions

    ESC. Exit

    Choose an option: Current Mode: Career
```

Figure 1: Main Menu Selection



```
D:\VS Workspace\Project12\D

Driver: Abdul

Score: +0

Time: 58
```

Figure 2: Active Gameplay showing Taxi, Traffic, and Passengers

4. User Manual (How to Play)

Controls

Key	Action
Arrow Keys	Move Taxi (Up, Down, Left, Right)
Spacebar	Interact (Pick up Passenger / Drop off Passenger)
P	Pause / Resume Game
ESC	Exit to Main Menu

Rules

1. **Objective:** Navigate the city to find passengers (waving figures).
2. **Pickup:** Stop adjacent to a passenger and press **Spacebar**.
3. **Drop-off:** A destination marker will appear on the map. Navigate to it and press **Spacebar** to earn points.
4. **Avoid:**
    - **Trees/Buildings:** Crashing causes damage (points deduction).
    - **Traffic:** Moving cars will crash into you.
    - **People:** Hitting a pedestrian results in a massive penalty (-5 points).
5. **Game Over:** The game ends if the score drops below zero or time runs out (in Time Mode).

5. Technical Implementation & Code Explanation

5.1 Memory Management

The city layout is stored in a 1-dimensional byte array (city_layout) of size 400 (20x20). The logic converts (Row, Col) coordinates to a memory offset using the formula:

Offset = (Row * 20) + Col

This allows for efficient collision detection by checking the byte value at the calculated offset (0 for road, 1 for building, 2 for tree).

5.2 Movement Logic

Movement is handled by the validate_move block. Before updating the taxi's position:

1. The code calculates the target_row and target_col.
2. It checks boundaries (0-19).
3. It checks the city_layout array for static obstacles.
4. It loops through traf_cols and traf_rows arrays to check for dynamic car collisions.
5. If the path is clear, the taxi coordinates are updated, and the screen is redrawn.

5.3 Passenger & Drop-off Logic

- **Data Structures:** Passengers are tracked using parallel arrays: ppl_cols, ppl_rows, and ppl_states (1=Waiting, 2=Picked Up).
- **Pickup:** The try_pickup_func checks all 4 adjacent cells for a passenger. If found, the passenger_onboard flag is set to 1.
- **Drop-off:** When a passenger is boarded, create_new_dropoff generates a random coordinate that is not a building or the current player position.

5.4 Traffic AI

The move_traffic procedure iterates through all NPC cars. It attempts to move them in their current direction (traf_dirs). If a car hits a wall, building, or another car, the code reverses the car's direction or assigns a random new direction, creating a semi-autonomous traffic flow.

5.5 Sound Implementation

The winmm.lib library is linked to play .wav files. A helper procedure play_audio invokes the Windows PlaySound function asynchronously, ensuring the game loop does not freeze while audio plays.

Code snippet

```
PlaySound PROTO STDCALL :PTR BYTE, :DWORD, :DWORD

; ...

INVOKE PlaySound, edx, 0, MODE_FILENAME OR MODE_ASYNC
```

6. Video Demo

A full video demonstration of the project, including gameplay of different modes and features, can be found at the following link:

https://drive.google.com/file/d/1_pdD36MbQZVJ72Spt3PZqJfU2tcaBmm1/view?usp=sharing

7. Conclusion

This project successfully replicates the mechanics of a "Rush Hour" taxi game. It meets the core requirements of grid navigation, collision detection, and dynamic game states. The implementation of varying taxi speeds, penalty calculations based on taxi color, and sound integration enhances the user experience, resulting in a complete and playable Assembly Language game.