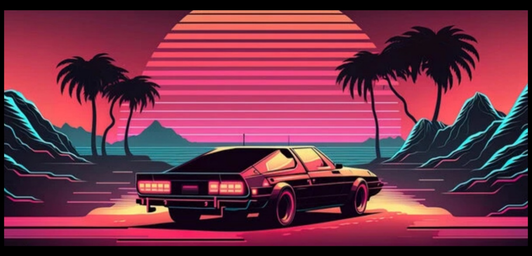
RushHour – Dynamic Passenger Pickup Game in C++



A real-time arcade-style game in C++ where players navigate through traffic, pick up passengers, and drop them off before time runs out.

Overview.

RushHour is a fast-paced arcade game developed in C++ where the player drives a car to pick up and drop off randomly spawned passengers. The game emphasizes real-time decision-making, path navigation, and dynamic obstacle avoidance. With each successful drop-off, the challenge intensifies as more traffic and obstacles appear.

Objective.

Develop a real-time 2D arcade-style driving game using C++. Incorporate random events like spawning of passengers and drop-off zones. Introduce increasing difficulty mechanics based on game progression. Encourage player skill development through a timed scoring system.

Tools.

- **Language:** C++
- **Graphics Library:** SFML / SDL (or your specific lib, if used)
- **Data Structures:** Queues for passenger management, arrays for map layout
- **Randomization:** rand() for dynamic passenger and destination generation
- **Timers:** For countdown and difficulty modulation

Process.

1. **Map Layout Design:** Defined 2D grid layout with designated road paths, spawn zones, and static obstacles.
2. **Vehicle Control Logic:** Arrow key-based input for car movement and collision detection.
3. **Passenger System:** Implemented queue or array to handle pickup/drop mechanics with score tracking.
4. **Difficulty Scaling:** With each drop-off, time slightly reduces, more cars appear, and their speed increases.

Core Features.

- **Random Spawning:** Passengers and destinations are generated at random locations for each session.
- **Obstacle Navigation:** Moving cars and static barriers force players to think and react quickly.
- **Progressive Difficulty:** Traffic speed and obstacle density increase after each successful drop-off.
- **Time-Based Pressure:** A countdown timer keeps the player on edge, encouraging fast and efficient play.
- **Collision Detection:** Real-time checks for vehicle-to-vehicle and vehicle-to-wall collisions.
- **Score System:** Points awarded for successful pickups and drop-offs.
- **🏁 Top 10 Leaderboard:** Final scores are saved using file handling in a persistent text file, maintaining a list of the top 10 players across sessions.

Challenges

- Ensuring smooth collision logic while increasing traffic speed.
- Avoiding overlapping spawns for passengers and obstacles.
- Balancing game difficulty without overwhelming the player.
- Managing frame rate stability under heavy game states.
- Designing a fair but challenging timer system.

Flowchart

Results, Learnings, and Impact

- Gained experience in efficient memory usage and object pooling.
- Explored balancing randomness with structured gameplay.
- Applied classical OOP principles to game development logic.
- Strengthened debugging and profiling of time-critical code.