Abraham Arocha

Abraham Arocha

Requirements for Simple Car Racing Game Prototype

FUNCTIONALITY

1. The game provides an animated or graphical representation of cars moving along their routes.
   1. Cars are visually displayed on a map or track that includes various stops/locations.
2. Each car travels a unique route consisting of a sequence of stops.
   1. For example:
      1. Car 1 might go from Stop A to B to C to D.
      2. Car 2 might start at Stop C, then go to D, A, and end at B.
   2. The start and end locations for each car are different.
3. The game introduces randomization to simulate realistic race conditions.
   1. Car speeds vary between stops based on random factors.
   2. Random events (like obstacles or weather conditions) may affect car performance.
4. The car that completes its assigned route in the shortest total time wins the race.
   1. The game calculates the total time for each car based on distances and speeds between stops.
5. The game displays detailed data for each car, including:
   1. The path taken (e.g., sequence of stops: A, B, C, D).
   2. The speed at each segment between stops.
   3. The total time taken to complete the race.
   4. At the end of the race, the game declares the winner.
6. The user can control the simulation through a graphical interface.
   1. Options to start, pause, and reset the race are provided.
   2. The user can select the number of cars participating in the race.
7. The user can customize certain aspects of the race.
   1. Choose different cars with varying performance characteristics.
   2. Select or generate different routes for each car.
8. The game maintains a leaderboard displaying past winners and fastest race times.
   1. Users can view race history and statistics.
9. Random obstacles or events can occur during the race, affecting car speeds or introducing delays.
   1. Examples include oil spills, tire punctures, or engine malfunctions.
10. The game updates the positions of cars in real time as they progress along their routes.
    1. Animations reflect the current state and speed of each car.

DATA

1. Input and output are managed through the graphical interface.
   1. Inputs:
      1. User commands from the graphical interface (start, pause, reset, car selection).
      2. Random data generated by the system (speeds, events).
   2. Outputs:
      1. Graphical display of the race, including car positions and movement.
      2. Textual data showing car paths, speeds, and total times.
      3. Notifications of events affecting cars during the race.
      4. Race results and winner announcement.
2. The game uses in-memory data structures to manage race data.
   1. Car objects with properties like speed, position, and route.
   2. Route objects containing sequences of stops and distances.
   3. Leaderboard data stored for the session; future versions may persist this data.
3. The game generates random values to simulate speed variations and events.
   1. Random number generators determine:
   2. Car speeds between stops.
   3. Occurrence and impact of obstacles/events.
4. Information about stops/locations, including names and distances between them.
   1. Possible predefined routes or the ability to generate them procedurally.

USABILITY

1. The interface is intuitive and easy to navigate.
   1. Clear buttons for starting, pausing, and resetting the race.
   2. Easy selection of cars and race parameters.
   3. Visual elements clearly represent cars and their progress.
2. Car paths, speeds, and total times are displayed prominently.
   1. Real-time updates keep the user informed of the race status.
   2. Events affecting cars are highlighted with visual and/or audio cues.
3. Users can interact with the simulation smoothly.
   1. Clicking on cars may provide additional details.
   2. Menus and dialogs are straightforward and provide helpful prompts.
4. The game prevents invalid actions.
   1. For example, if an action is invalid during a race, a message explains why.
   2. User inputs are validated, and informative error messages are provided as needed.
5. The interface design considers accessibility guidelines.
   1. Clear fonts, high-contrast visuals, and scalable interface elements.
   2. Users may adjust settings like animation speed or sound effects.

RELIABILITY AND AVAILABILITY

1. The game handles edge cases and unexpected inputs gracefully.
   1. Race calculations are accurate, ensuring fair determination of the winner.
2. The game runs smoothly without crashes or significant bugs.
   1. Memory management avoids leaks that could degrade performance over time.
3. Data integrity is maintained throughout the simulation.
   1. Randomization processes produce realistic and varied outcomes without causing instability.
4. The game is a standalone application that runs locally.
   1. It doesn't depend on external services, ensuring it's available whenever the user wants to play.

PERFORMANCE

1. The graphical interface and animations are optimized for smooth performance.
   1. Frame rates are maintained at a level that ensures fluid motion.
2. User inputs are processed immediately.
   1. The game responds quickly to commands like start, pause, and reset.
3. The game handles varying numbers of cars without significant performance degradation.
   1. The system is capable of managing additional complexity if more features are added.
4. Memory and CPU usage are kept within reasonable limits.
   1. Background processes, such as randomization and event handling, are efficient.

SUPPORTABILITY

1. The game is developed with a modular architecture.
   1. Components like the race logic, car mechanics, and user interface are separable.
   2. This design facilitates maintenance and future enhancements.
2. The codebase is structured to allow for easy addition of new features.
   1. New car types, obstacles, or race modes can be integrated without major rewrites.
3. The code is well-documented with comments explaining key functions and classes.
   1. User manuals or in-game help screens explain how to use the game.
4. The game includes unit tests for critical components.
   1. Regular testing ensures that new changes don't break existing functionality.

FURPS(PLUS)

1. Additional Considerations
   1. Portability: The game can run on multiple platforms (e.g., Windows, macOS, Linux).
   2. Localizability: The game is designed to support multiple languages in the future.
   3. Scalability: Architecture allows for the addition of networked multiplayer in future versions.
   4. Legal and Ethical: All assets used (graphics, sounds) are properly licensed or created in-house.