AERSP 424 Final Project Report

Formula One Race Simulator

By: Nathen Realmuto and Jay Bhavsar

Overview:

In this project, we have created a Formula One race simulator. Formula One, commonly known as F1, is the highest international racing class. This program was designed to simulate one of these races. The code features popular F1 tracks, drivers, and their corresponding vehicles. Although you don't actually get to play as a driver, the program simulates the main aspects of a race, such as starting grid, lap times, and finishing order. Core features of this program include track and driver selection, race dynamics, starting grid and race progression, and finishing order and statistics.

Components

1. Track Class (Track.cpp and Track.h)

- Purpose: Manages the properties of the racetrack.
- Functionality:
    - Initialization: Initializes track with the name, location, number of turns, and number of straights, as well as the fastest lap.

2. Driver Class (Driver.cpp and Driver.h)

- Purpose: Defines drivers and stores them.
- Functionality:
    - Initialization: Initializes driver with name, team, driver age, speed, handling, and stamina skills.
    - Driver list and skills: Contains all drivers to choose from with their corresponding skillset.
    - Car assignment: Assigns each driver the car they will drive.

3. Car Class (Car.cpp and Car.h)

- Purpose: Defines car properties and stores them.
- Functionality:
    - Car list and performance: Contains each of the cars and their performance characteristics.

4. Main Function (main.cpp)

- Purpose: Creates and runs the simulation.
- Functionality:
    - Initialization: Initializes components and starts the simulator.
    - Simulator Inputs: Manages user inputs for track selection and driver grid order.
    - Simulator Logic: Calculates driver average lap times and total time.
    - Outputs: Displays race results with driver order and lap times, and on request detailed lap by lap times.

C++ features used

1. Object Oriented Programming

- **Classes and Objects:** Definition and use of classes like Track and Driver.
- **Encapsulation:** Encapsulating data within classes using private and public access modifiers.

2. Basic Language Constructs

- **Data types:** Use of multiple standard data types, such as 'int', 'float', 'double', and 'std::string'.
- **Control Structures:** Use of 'for' and 'while' loops, as well as conditional statements ('if', 'else'), and functions

3. Standard Template Library (STL)

- **Containers:** Use of std::vector, std::map, std::queue to manage collections of objects.
- **Iterators:** Used in conjunction with STL containers for data access and manipulation.
- **Algorithms:** Use of algorithms like std::sort and std::find_if.
- **Utilities:** Use of std::pair for handling tuple-like data.

4. Functions and Methods

- **Member Functions:** Class methods to perform operations specific to the objects.
- **Non-member Functions:** Global functions that operate outside of classes to perform tasks like data conversion, such as convertTime.

5. Input/Output

- **Stream I/O:** Utilization of std::cin and std::cout for input from and output to the console.

6. Multithreading

- **Threading:** Use of std::thread to perform parallel tasks.
- **Mutexes:** Use of std::mutex and std::lock_guard/std::unique_lock to manage concurrent access to shared resources.
- **Condition Variables:** Use of std::condition_variable for inter-thread synchronization.

7. Code Organization

- **Header Files:** Use of header files (Track.h, Driver.h) to define class interfaces.
- **Source Files:** Separate source files for implementation details.

8. Modern C++ Features

- **Auto:** Usage of auto for type inference.
- **Lambda Expressions:** Used for defining inline functions.
- **Chrono Library:** Used for handling durations and time points, such as std::chrono::seconds.

Code improvements

There are multiple code enhancements that could be made to increase the realism of the simulator. Implementing an advanced physics engine that considers linear and angular dynamics, as well as environmental factors such as track wetness or temperature. In addition, formulas for friction, drag, and tire slip can simulate the cars' interactions with the track. The addition of a dynamic weather system can also have an effect on the simulation, as the weather can affect track temperatures, traction, and driver strategy. Driver AI improvements can be made, allowing real time race strategy updates based on the race situation. Introducing a tire wear function can increase the depth of the simulation, affecting performance and requiring pit stops for replacements. A real time damage model can be developed where collisions and accidents affect the car's performance, including aerodynamic penalties, reduced speed, and handling difficulties.