Documentación Técnica - Need For Speed

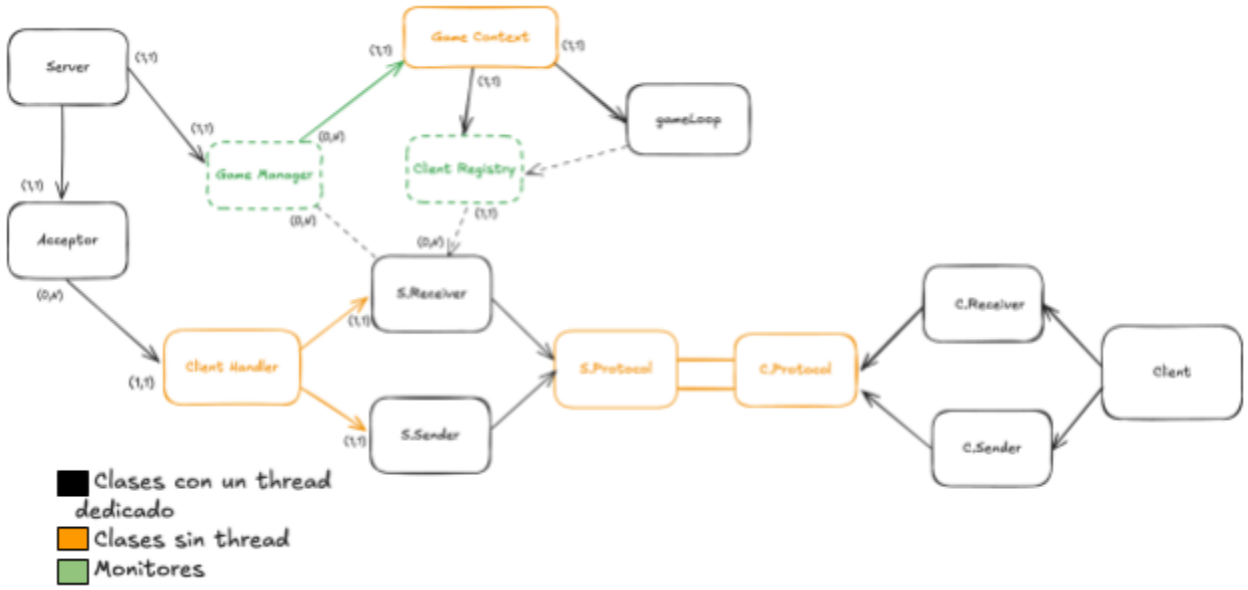
1. Arquitectura General

El sistema utiliza una arquitectura **Cliente-Servidor** robusta.

- **Modelo de Concurrencia:** El servidor utiliza un hilo aceptador (*AcceptorThread*) y un hilo por cada cliente conectado (*ClientHandler*), más un hilo principal para el *GameLoop* (bucle de juego).
- **Lógica:** Toda la lógica del juego (física, colisiones, estado de la carrera) reside en el servidor para evitar trampas.
- **Vista:** El cliente es "dummy"; solo renderiza el estado que recibe del servidor y envía inputs.

Diagrama de Hilos y clases principales:

El siguiente diagrama muestra cómo se manejan las conexiones y el ciclo de juego:




```
node "Main Server Process" {
    component [AcceptorThread]
    component [GameLoop]

    node "Client 1 Handler" {
       [Receiver]
       [Sender]
    }
```

```
    node "Client 2 Handler" {
       [Receiver]
       [Sender]
    }
}

[AcceptorThread] --> [Client 1 Handler] : Crea al conectar
[Client 1 Handler] <--> [GameLoop] : Lee/Escribe Colas Protegidas
```

2. Diagramas de Clase Principales

A continuación se detallan las clases core del dominio del juego separado en los principales módulos del trabajo.

Modelo del Mundo (Server Business-Logic)

El servidor está dividido en dos grandes módulos: world y game_logic:

- En world se concentra toda la parte física del juego. Allí se encuentra el WorldManager, encargado de crear y administrar los cuerpos de Box2D (autos, edificios, checkpoints) y de instanciarlos a partir de un archivo YAML del mapa mediante un parser. En este módulo también se ubica el WorldContactHandler, que actúa como listener de las colisiones de Box2D y las traduce a eventos que se encolan para ser procesados por la lógica de juego.

- En game_logic se implementa la lógica de alto nivel. El núcleo es el GameLoop, que ejecuta el loop principal del servidor y orquesta el resto de los componentes. Para la fase previa a la carrera utiliza un LobbyController, que gestiona el lobby (espera de jugadores y condición de inicio), y luego un RaceController, responsable de la lógica de la carrera (progreso por checkpoints, finalización, etc.).
  Además, se define un PlayerManager, que recibe las acciones de cada cliente y aplica esos inputs sobre los autos correspondientes, y un WorldEventHandler, que consume los eventos de colisión generados en world, actualiza el estado de la carrera y determina la información que se envía a los jugadores.

- Dentro de game_logic existe además una carpeta config, que contiene un archivo YAML con parámetros de configuración y un parser asociado. En este módulo se definen valores ajustables del juego (por ejemplo, constantes de física, tiempos y distintas variables de la partida), que luego son leídos al iniciar el servidor y utilizados por componentes de la lógica de juego, como GameLoop, y por aquellos que interactúan con el mundo físico. Esto permite modificar el comportamiento del juego sin cambiar el código ni recompilar.

Esta separación entre world y game_logic permite desacoplar la simulación física de la lógica de juego. El módulo world se limita a representar el estado del mundo y las colisiones, mientras que game_logic decide cómo interpretar esos eventos y cómo avanzar la partida. De esta forma se facilita el mantenimiento, la extensión de reglas de juego y la reutilización del mundo físico en distintos modos de juego.

Cliente y Renderizado

Al igual que del lado del servidor, la comunicación con el mismo se hace a través de dos hilos: Sender y Receiver. Para serializar o deserializar mensajes se utiliza de igual forma que del lado del servidor el protocolo.

El cliente está dividido en varios módulos con responsabilidades bien separadas

- La Camera es la encargada de seguir al auto del jugador y determinar qué parte del mundo debe mostrarse en pantalla en cada momento.
- El ClientWindow es la ventana principal creada al iniciar SDL y actúa como núcleo del renderizado, ya que allí se dibuja todo lo visible del juego.
- El EventManager procesa los eventos que llegan desde SDL, traduce las entradas del jugador y se las envía al sender, además recibe la información del Receiver y se la pasa al ClientWindow para actualizar lo que se muestra.
- El TextureManager y el resto de las clases en /textures administran todas las texturas y sprites, manejando su carga, almacenamiento y reutilización para optimizar recursos.
- Finalmente, dentro de /renderables están las clases que representan todos los objetos dibujables, cada uno con su propia lógica de render que utiliza las texturas gestionadas por el TextureManager y se integra con la ClientWindow para aparecer correctamente en pantalla.

3. Protocolo de Comunicación

La comunicación se realiza mediante Sockets. El protocolo es binario y se serializa byte a byte la información insertando previamente un Opcode para indicar el tipo de mensaje que está pasando por Socket.

Formato General del Mensaje:

```
|----------|------------|
| OPCODE |  PAYLOAD   |
```

Opcodes Definidos

Todos los diferentes Opcodes están definidos en la ruta:

```
/source/common_src/opcodes.h
```

5. Dependencias y Bibliotecas Externas

- **Box2D** : Para simulación física.
- **SDL2**: Renderizado de texturas y manejo de eventos.
- **Qt**: Interfaz gráfica del Lobby y estadísticas post-partida.