

Documentación Técnica



Manuel Pato Ibáñez	110640
Lucas Nicolás Pagani	110777
Federico Zanor	112097
Joaquín Schapira	112459

3 de Diciembre de 2025

Índice

1. Introducción	3
2. Protocolo de comunicación	3
3. Mensajes Cliente → Servidor	4
3.1. Input de teclas y cheats	4
3.2. Crear lobby	4
3.3. Unirse a lobby	5
3.4. Iniciar partida desde lobby	5
3.5. Enviar mejora del auto	5

3.6. Salir del lobby / Desconexión	6
4. Mensajes Servidor → Cliente	6
4.1. Snapshot del juego	6
4.2. Asignación de ID al cliente	7
4.3. Error al unirse a lobby	7
4.4. Snapshot del lobby	8
4.5. Inicio de partida del lobby	8
4.6. Snapshot pre-carrera (pre-game)	8
4.7. Resultados de carrera	9
4.8. Confirmación de unión OK	10
4.9. Cambio de fase	10
5. Comunicación Cliente–Servidor	10
6. Servidor	12
7. Cliente	16

1. Introducción

En este software se implementa una reencarnación del Need for Speed en una versión 2D. Se permite la creación de varias partidas simultáneas con un número fijo de hasta 8 jugadores por partida.

Cada partida, constará de varios mapas elegidos por el creador de la partida. Cabe destacar que, aunque hayan varios mapas creados por defecto, se permite mediante el editor agregar cuantos mapas sean necesarios para garantizar infinitos recorridos posibles, evitando así, el aburrimiento del usuario.

El diseño del software se realizó mediante una arquitectura cliente-servidor que se comunican mediante **sockets bloqueantes**. No existe comunicación entre estos y el editor, ya que el editor crea mapas jugables y los almacena en un directorio determinado desde donde se leerán los mapas. El servidor es **único**, mientras que existe un cliente por usuario. Cada cliente se comunica a través de su **propio socket** con el servidor, por lo que, para evitar que se bloquee el juego, fue fundamental la creación de **threads** a lo largo del desarrollo, tema del que se hablará más adelante en este mismo documento.

2. Protocolo de comunicación

El protocolo de comunicación entre el cliente y el servidor es **binario**. Los códigos de acciones se detallan en las siguientes tablas. Por protocolo, todos se envían en un byte con el valor literal.

Eventos Servidor → Cliente

Acción	Descripción	Código
EVENT_SEND_SNAPSHOT	Snapshot de juego	0x01
EVENT_SEND_ID	Enviar ID único de jugador	0x15
EVENT_LOBBY_JOIN_ERROR	Error al unirse a un lobby	0x20
EVENT_LOBBY_SNAPSHOT	Snapshot del lobby	0x21
EVENT_START_LOBBY	Inicio de partida	0x22
EVENT_PRE_GAME_SNAPSHOT	Info previa a carrera	0x23
EVENT_RACE_RESULTS	Resultados de la carrera	0x24
EVENT_JOIN_SUCCESS	Confirmación de unión	0x30
EVENT_PHASE_CHANGE	Cambio de fase	0x32

Eventos Cliente → Servidor

Acción	Descripción	Código
INPUT_KEY	Input de teclado	0x12
SEND_CREATE_LOBBY	Crear lobby	0x16
SEND_JOIN_LOBBY	Unirse a lobby	0x17
SEND_START_GAME	Iniciar partida	0x22
SEND_CAR_UPGRADE	Enviar mejora	0x33
LEAVE_LOBBY	Salir de lobby	0x34

El protocolo es binario y todos los enteros multi-byte se envían en **big endian**. Los strings se envían precedidos de su longitud y sin byte \0. Los tiempos son enviados en segundos.

3. Mensajes Cliente → Servidor

3.1. Input de teclas y cheats

Formato:

- INPUT_KEY = 0x12
- INPUT_KEY <key_code>

Mapeo de códigos:

0x00	UP_PRESSED
0x01	LEFT_PRESSED
0x02	DOWN_PRESSED
0x03	RIGHT_PRESSED
0x04	UP_UNPRESSED
0x05	LEFT_UNPRESSED
0x06	DOWN_UNPRESSED
0x07	RIGHT_UNPRESSED
0x08	COMMAND_WIN
0x09	COMMAND_LOSE
0x10	COMMAND_INFINITE_LIFE
0x11	COMMAND_GHOST

3.2. Crear lobby

Formato:

- SEND_CREATE_LOBBY = 0x16

- <car_model>(uint8)
- <name_len>(uint16)
- <player_name>(bytes)
- <map_count>(uint16)

Por cada mapa:

- <map_len>(uint16)
- <map_name>(bytes)

3.3. Unirse a lobby

Formato:

- SEND_JOIN_LOBBY = 0x17
- <lobby_code>(uint32)
- <car_model>(uint8)
- <name_len>(uint16)
- <player_name>(bytes)

3.4. Iniciar partida desde lobby

Formato:

- SEND_START_GAME = 0x22
- <lobby_code>(uint32)

3.5. Enviar mejora del auto

Formato:

- SEND_CAR_UPGRADE = 0x33
- <upgrade_code>(uint8, 1-9)

3.6. Salir del lobby / Desconexión

Formato:

- CMD_DISCONNECT = 0x34

4. Mensajes Servidor → Cliente

4.1. Snapshot del juego

Formato:

- EVENT_SEND_SNAPSHOT = 0x01
- <time_remaining>(uint32)
- <player_count>(uint16)

Payload por jugador (N jugadores):

Por cada jugador:

- <player_id> (uint32)
- <is_ghost> (uint8, 0x01 = ghost)
- <car_life> (uint16)
- <car_model> (uint16)
- <car_animation> (uint8)
- <sound_code> (uint8)
- <x_px> (uint32)
- <y_px> (uint32)
- <z> (uint8)
- <angle> (uint32)
- <checkpoint_count_1> (uint16)

Para cada punto del primer checkpoint:

- <cp1_x_px> (uint32)
- <cp1_y_px> (uint32)

- <is_finishline_1> (uint8)
- <has_second_checkpoint> (uint8)

Si hay segundo checkpoint ('flag = 0x01'):

- <checkpoint_count_2> (uint16)

Para cada punto del segundo checkpoint:

- <cp2_x_px> (uint32)
- <cp2_y_px> (uint32)

- <is_finishline_2> (uint8)

- <npc_count> (uint16)

Payload por NPC (M NPCs):

Por cada NPC:

- <npc_model> (uint16)
- <npc_animation> (uint8)
- <npc_x_px> (uint32)
- <npc_y_px> (uint32)
- <npc_z> (uint8)
- <npc_angle> (uint32)

4.2. Asignación de ID al cliente

Formato:

- EVENT_SEND_ID = 0x15
- <player_id> (uint32)

4.3. Error al unirse a lobby

Formato:

- EVENT_LOBBY_JOIN_ERROR = 0x20

4.4. Snapshot del lobby

Formato:

- EVENT_LOBBY_SNAPSHOT = 0x21
- <lobby_id> (uint32)
- <player_count> (uint16)

Payload por jugador (N jugadores):

Por cada jugador:

- <name_len> (uint16)
- <name> (bytes)
- <car_model> (uint8)

4.5. Inicio de partida del lobby

Formato:

- EVENT_START_LOBBY = 0x22

4.6. Snapshot pre-carrera (pre-game)

Formato:

- EVENT_PRE_GAME_SNAPSHOT = 0x23
- <pole_up_left_x>(uint32)
- <pole_up_left_y>(uint32)
- <pole_down_right_x>(uint32)
- <pole_down_right_y>(uint32)
- <direction>(uint8)
 - 0x01 = RIGHT
 - 0x02 = LEFT
 - 0x03 = UP

- 0x04 = DOWN
- <remaining_races>(uint16)
- <map_id>(uint8)
- <race_total_time>(uint32)
- <race_move_enabled_time>(uint32)

4.7. Resultados de carrera

Formato: Race Results Event:

- EVENT_RACE_RESULTS = 0x24
- <is_last_race> (uint8)
 - 0x00 = carrera intermedia
 - 0x01 = última carrera (incluye podio)
- <player_count> (uint16)

Si is_last_race == 0x00:

Por cada jugador:

- <name_len> (uint16)
- <name> (bytes)
- <last_race_time> (uint32)
- <total_race_time> (uint32)
- <finish_flag> (uint8)

Si is_last_race == 0x01 (última carrera):

- **Primero se envían jugadores NO podio** (mismo formato que carrera intermedia)
- <phase_to_show> (uint8)
 - 0 = no mostrar podio
 - 1 = sólo 3°

- $2 = 2^\circ$ y 3°
- $3 = 1^\circ, 2^\circ$ y 3°

Luego se envían 3 jugadores del podio en orden:

Por cada puesto del podio:

- <player_exists> (uint8)
- <name_len> (uint16)
- <name> (bytes)
- <podium_last_race_time> (uint32)
- <podium_total_race_time> (uint32)
- <podium_finish_flag> (uint8)

4.8. Confirmación de unión OK

Formato:

- EVENT_JOIN_SUCESS = 0x30

4.9. Cambio de fase

Formato:

- EVENT_PHASE_CHANGE = 0x32

5. Comunicación Cliente–Servidor

El siguiente diagrama muestra el flujo completo de mensajes entre los distintos clientes y el servidor durante una partida. Cada cliente ejecuta dos hilos dedicados a la comunicación:

ThreadSender, encargado de enviar al servidor los eventos generados por el jugador (entradas de teclado, acciones, upgrades, etc.).

ThreadReceiver, encargado de recibir desde el servidor los snapshots y eventos del juego.

Del lado del servidor, se tiene una logica similar para los ThreadSender y ThreadReceiver pero además mantiene un gameloop por cada lobby, como se puede observar en la imagen, los clientes 1 y 2 al estar conectados a la misma lobby comparten el thread gameloop, mientras que el cliente 3 y 4 pertenecientes a otra lobby tienen su thread correspondiente.

Al igual que el cliente, el servidor también utiliza dos hilos por usuario, uno encargado de recibir las acciones de cada cliente y otro responsable de comunicar el estado de la partida.

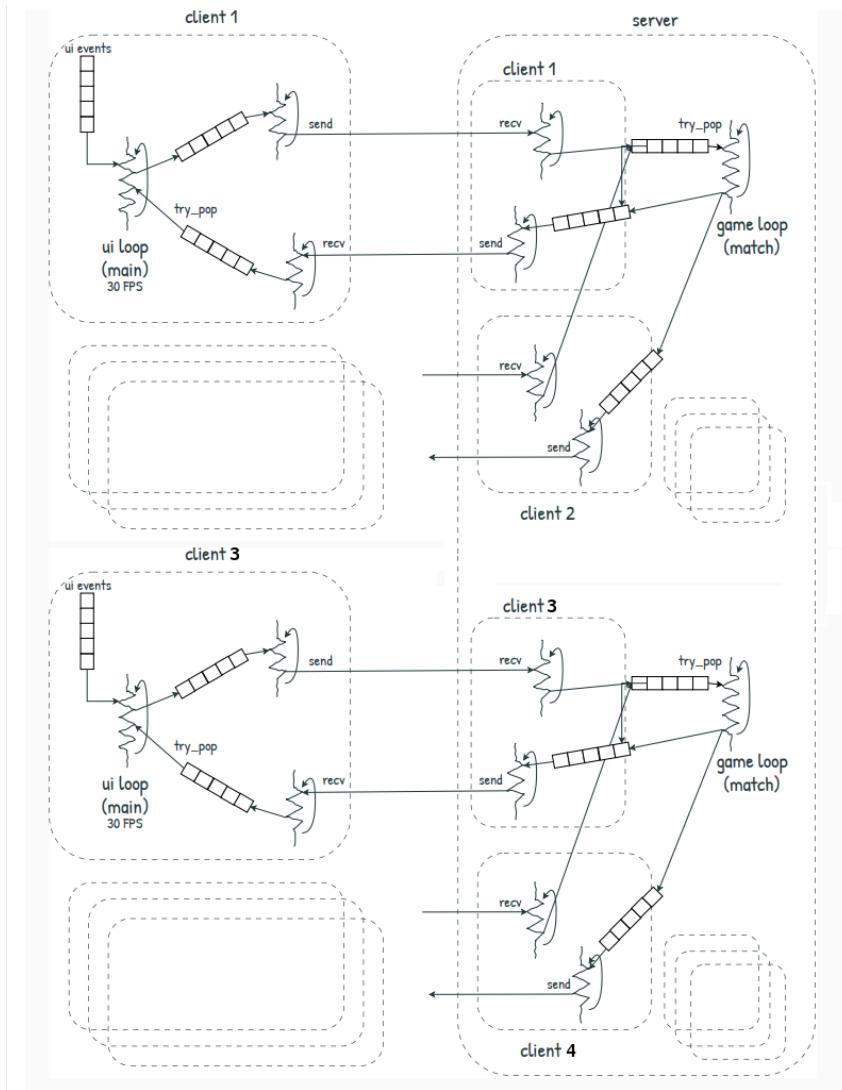


Figura 1: Diagrama de secuencias entre cliente y servidor.

Cabe aclarar, que esta imagen ha sido generada a partir del material correspondiente a la clase de Arquitectura Servidor Cliente (utilizandola como base y agregandole multiples gameloops)

6. Servidor

Luego de ya haber visto un poco cómo funciona la conexión en el diagrama de más arriba, pudimos ver que por cada partida, hay un hilo gameloop que tiene varios mapas. Veamos ahora cómo es que puede ser esto posible:

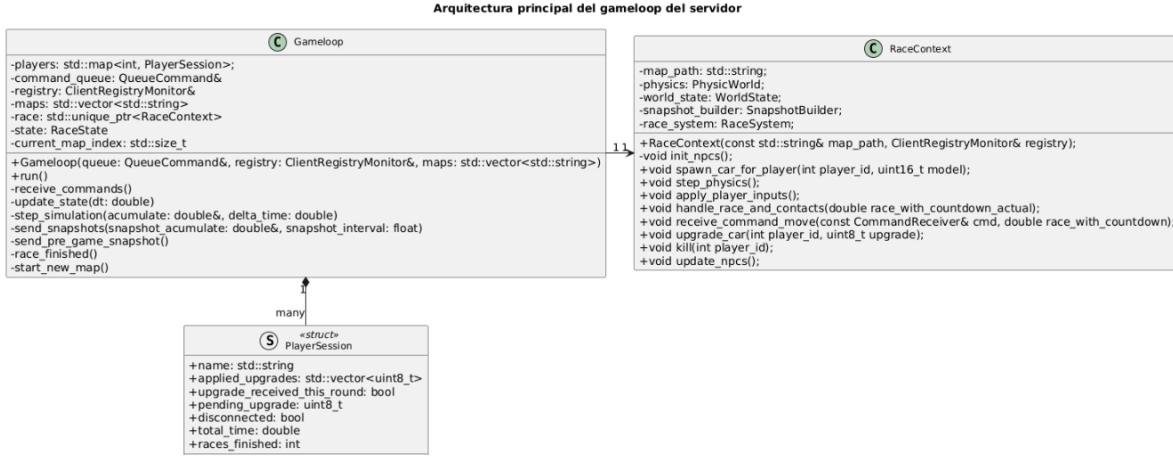


Figura 2: Arquitectura principal del servidor.

Este diagrama muestra la estructura central del servidor durante la ejecución de una partida.

El hilo Gameloop es el núcleo del sistema: recibe comandos de los jugadores, ejecuta la simulación física, controla el avance de fases (carrera, resultados, mejoras, próxima carrera) y coordina el envío de snapshots al cliente.

El Gameloop contiene un mapa de PlayerSession, donde se almacena el estado persistente de cada jugador (tiempos, mejoras, desconexiones, etc.) a lo largo de toda la partida.

Además, mantiene una instancia única de RaceContext, que encapsula toda la lógica de la carrera actual: mundo físico, estado del mundo, sistema de checkpoints, NPCs y generación de snapshots.

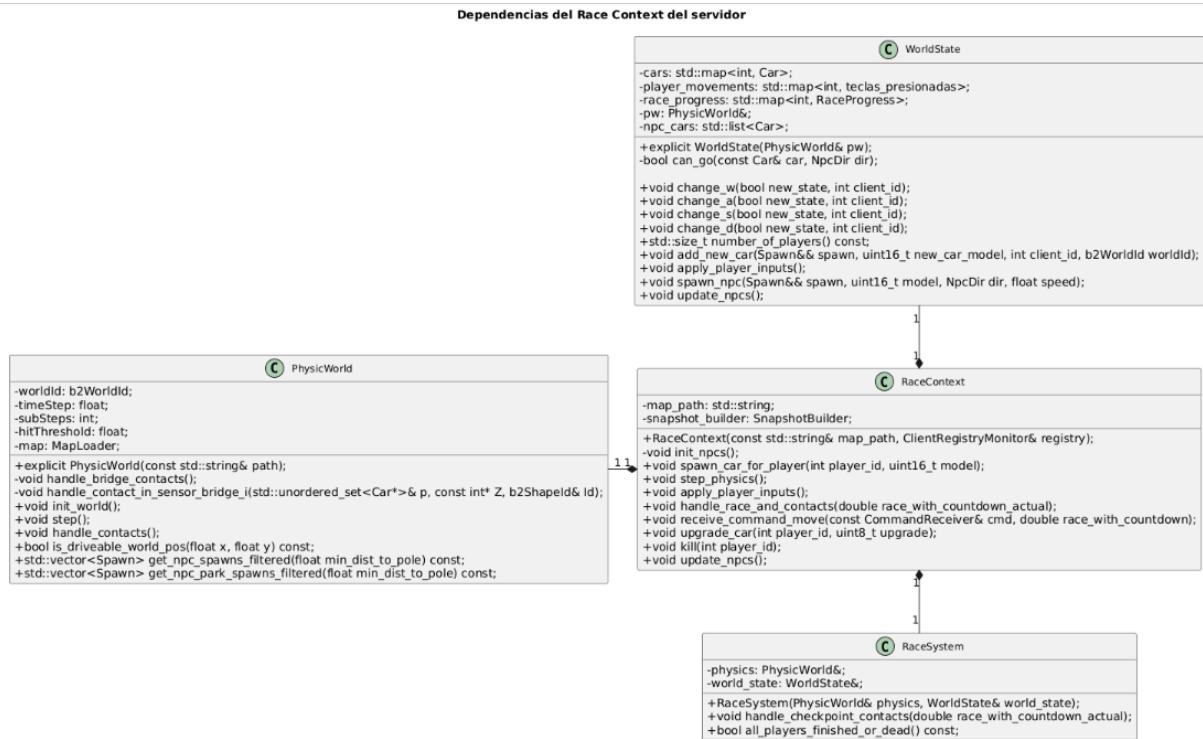


Figura 3: Dependencias del RaceContext del servidor.

Este diagrama detalla las dependencias internas del RaceContext, que representa una carrera individual dentro de una partida. Cuando una partida tenga 20 mapas, se irán creando 20 instancias de RaceContext a lo largo de la partida.

El RaceContext agrupa varios componentes clave:

- PhysicWorld: motor de física basado en Box2D.
- WorldState: estado lógico del mapa y los autos, incluyendo inputs, NPCs y movimiento.
- RaceSystem: sistema de checkpoints, meta y validación de progreso.
- SnapshotBuilder: generador del snapshot enviado al cliente.

La combinación de todos estos módulos permite separar lógicas complejas como:

- Física
- Creación de usuarios
- Inteligencia de NPC
- Detección de colisiones
- Control de carrera
- Serialización del estado

WorldState con Car

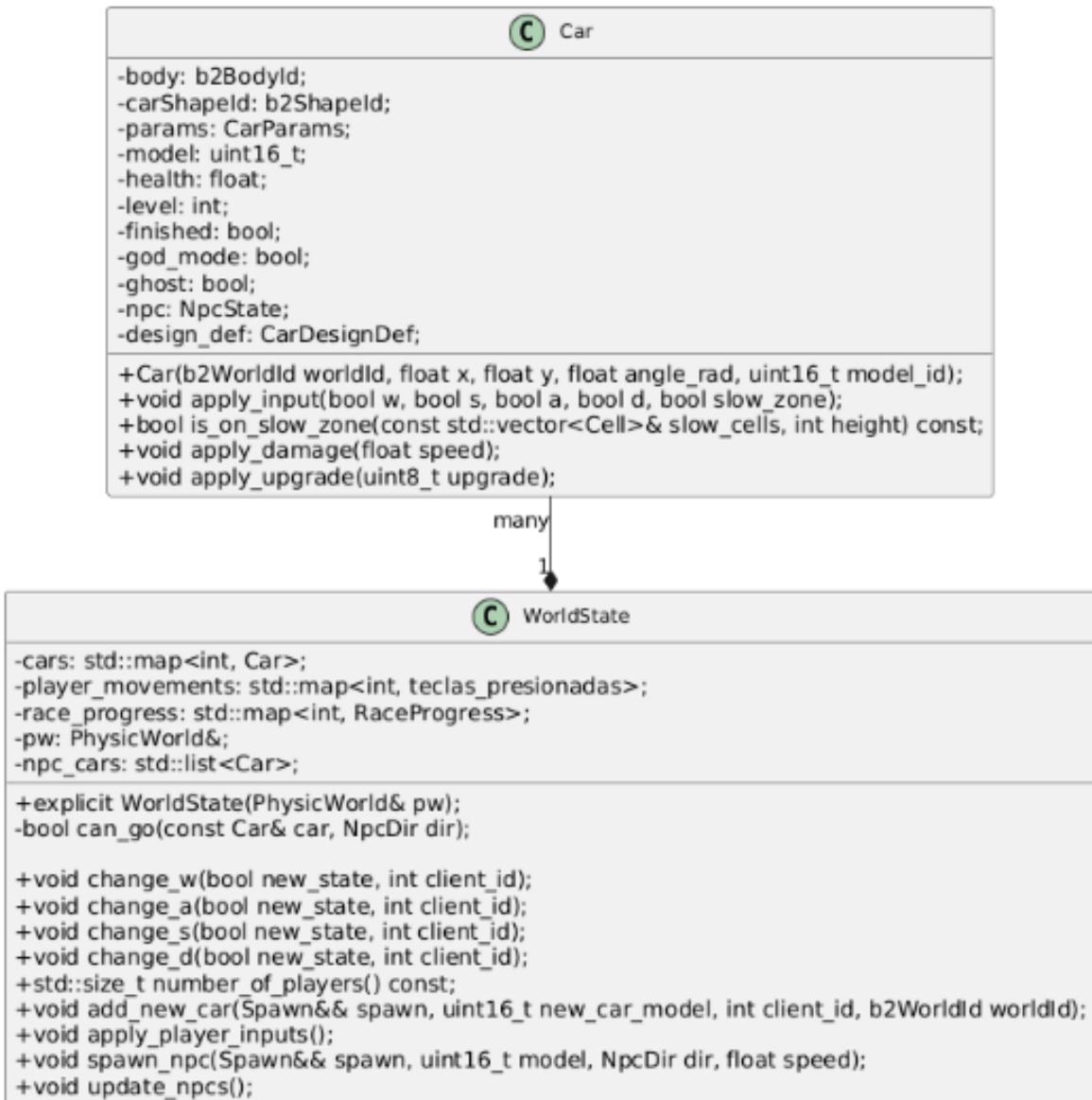


Figura 4: WorldState con Car.

Este diagrama muestra la lógica del estado del mundo durante la carrera.

WorldState mantiene:

- todos los autos del mapa.
- inputs por jugador.
- progreso de checkpoints.
- lista de NPCs.
- También contiene la lógica para:

aplicar inputs del jugador.
 spawnear autos y NPCs.
 ir actualizando el movimiento de NPCs con decisiones aleatorias

- Serialización del estado

Por otro lado, la clase **Car** encapsula:

- Física propia del vehículo.
- Modelo, tamaño, propiedades del mismo.
- Vida.
- Ghost / god mode.
- Velocidad, fricción, aceleración, densidad, etc.



Figura 5: Diagrama de PhysicWorld con MapLoader.

Por último, en **PhysicsWorld** encapsulamos el mundo físico de Box2D, los contactos con los sensores y los choques de los autos, delegando en **MapLoader** la carga del mapa, encargándose de leer el archivo y generar estructuras tales como:

- Crear los *bodies* y *shapes* reales de Box2D.

- Generar sensores de colisión.
- Definir zonas de velocidad reducida o puentes.

7. Cliente

A continuación se presentan los diagramas de clases principales que describen la estructura del cliente.

Diagrama general del cliente

Este diagrama muestra la arquitectura global del cliente y cómo se organiza su comunicación con el servidor mediante los hilos **ThreadSender** y **ThreadReceiver**. También se observa la relación del cliente con las pantallas de Lobby y con el módulo principal del juego, **GameloopSDL**, que gestiona la carrera y el renderizado.

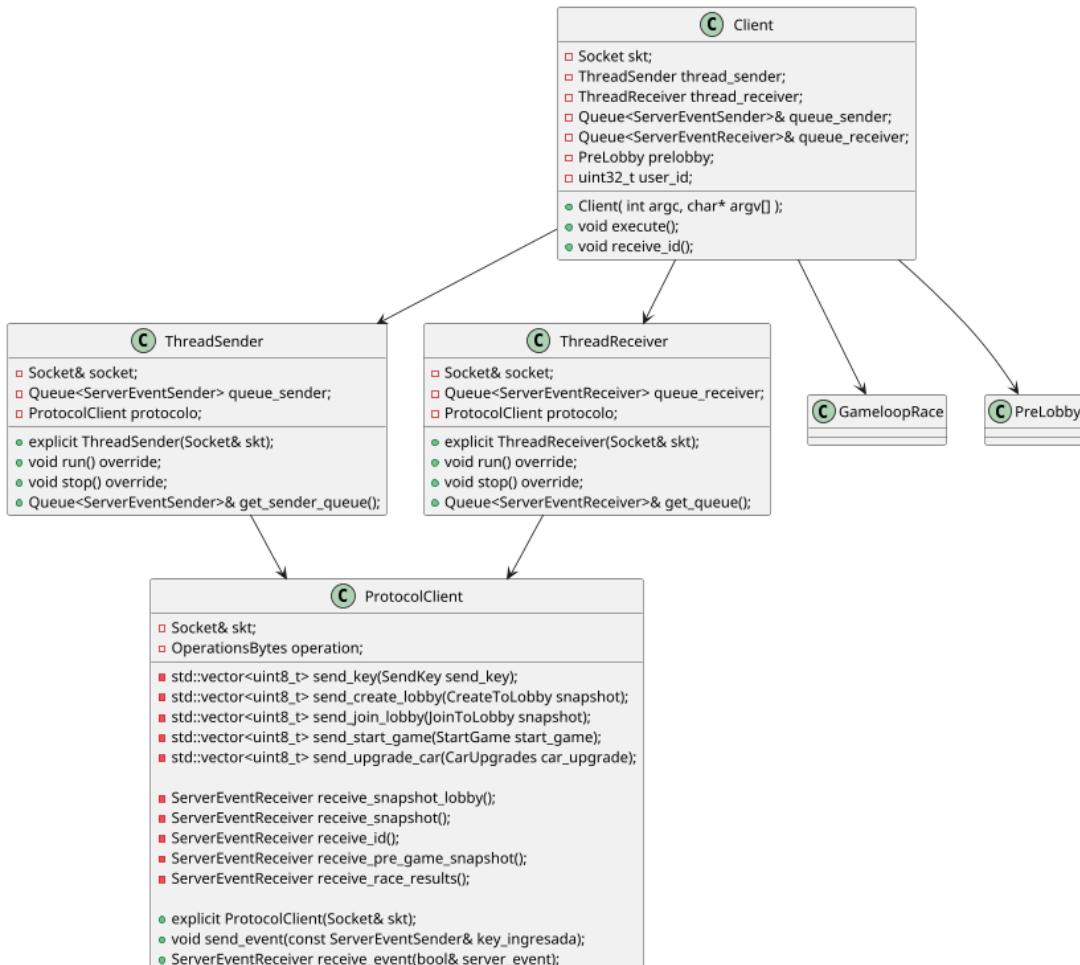


Figura 6: Diagrama general del cliente.

Lobby

Este diagrama describe el flujo de pantallas implementado con Qt, responsable de toda la interacción del usuario antes de comenzar la carrera.

- **PreLobby:** primera pantalla que aparece en el cliente; el usuario elige crear un lobby o unirse a uno existente.
- **CreateLobby:** permite crear una sala nueva seleccionando auto, mapa y nombre del jugador. Envía esta información al servidor mediante colas y espera la respuesta indicando si la lobby se pudo crear.
- **JoinLobby:** ventana utilizada para unirse a un lobby ingresando el código de sala, el nombre del jugador y el auto.
- **Lobby:** sala ya creada o unida. El cliente recibe *snapshots* de lobby desde el servidor (jugadores conectados, código de sala, etc.). Desde aquí se puede comenzar la partida, lo que deriva al **GameLoop**.

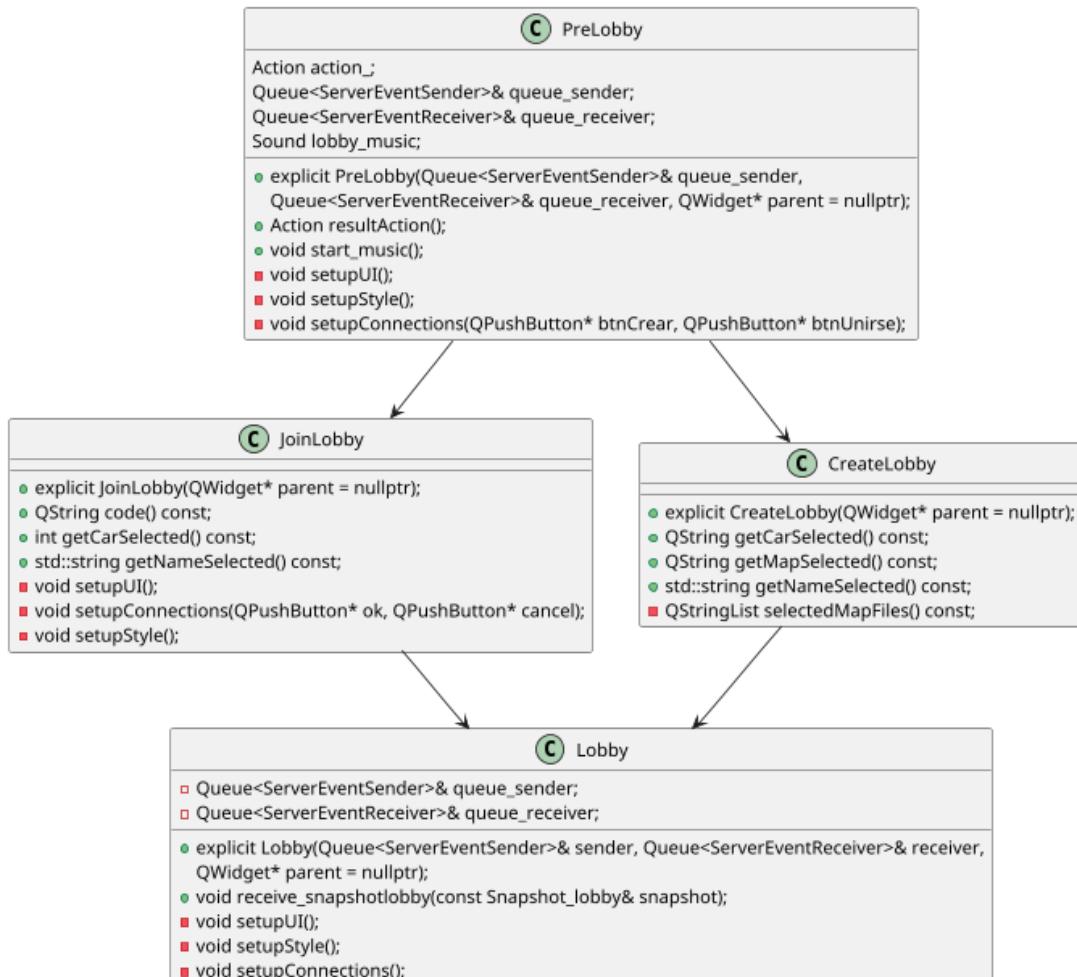


Figura 7: Diagrama de clases del Lobby del cliente.

Juego

Este diagrama muestra los componentes principales que intervienen durante la carrera, todos ellos coordinados por `GameloopRace`. `GameloopRace` procesa eventos recibidos desde el servidor, maneja las entradas del usuario, actualiza el estado local del juego y solicita a la interfaz gráfica (SDL) que renderice cada cuadro.

Para cumplir estas funciones, se utilizan los siguientes módulos:

- **GameSoundManager**: controla la música del juego y su volumen.
 - **GuiSDL**: renderiza todo el contenido visual usando SDL: mapa, autos, *hints*, NPCs, upgrades y pantallas finales; además captura entradas de teclado y ventana.
 - **InputHandler**: traduce eventos de SDL en acciones que el jugador envía al servidor (acelerar, frenar, doblar, cheats, etc.).
 - **RacePhaseManager**: gestiona la lógica de las fases de la carrera, procesando snapshots, pre-game, resultados y definiendo cuándo avanzar de una etapa a otra.

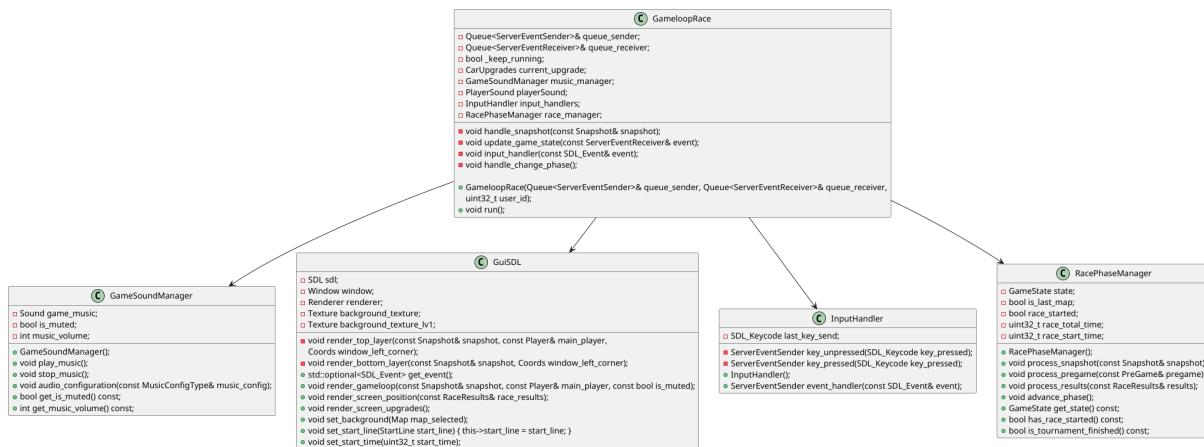


Figura 8: Diagrama de clases del juego (SDL).