**Taller de Programación I (75.42)**
**Facultad de Ingeniería**
**Universidad de Buenos Aires**

# *Need for Speed*

# Índice



# Introducción

En este TP se implementara una reencarnación del mítico *Need for Speed* en una versión 2D donde los jugadores correrán carreras por distintas ciudades para ver quien es el más veloz.

# Descripción

## Carrera y partida

Cada partida de hasta 8 jugadores consta de un número fijo de carreras. Cada carrera tendrá un recorrido por una ciudad determinada de las tres disponibles: Liberty City, San Andreas y Vice City.

Al finalizar cada carrera deberá mostrarse una tabla de posiciones con el tiempo de cada jugador, tanto para la carrera que acaba de finalizar como con el total acumulado de las carreras terminadas hasta el momento.

Una misma ciudad puede reutilizarse para múltiples carreras, basta con solo tener recorridos distintos. Estos recorridos están marcados por la ciudad en forma de checkpoints y hints.

Los checkpoints son franjas que cruzan la calle que cada competidor debe cruzar y que los autos deben cruzar para considerar que está realizando el recorrido trazado para la carrera. El punto de partida y de llegada son checkpoints que deben mostrarse gráficamente distintos al resto de los checkpoints.

Dado que los competidores pueden no ver donde está el siguiente checkpoint, sobre las calles deberán aparecer hits o flechas que indiquen la dirección hacia donde ir (adelante, doblar) para alcanzar al siguiente checkpoint.

Tanto los checkpoints como los hints deben aparecer en orden: el cliente gráfico debe mostrar solamente el primer checkpoint y los hints hacia él y cuando el auto del jugador cruza el checkpoint, recien ahi se muestra el segundo checkpoint y hints hacia él y así sucesivamente.
Notar que checkpoint y hints muestran depende del auto del jugador que el cliente gráfico está controlando.

Los checkpoints son la forma que tiene el juego para que los jugadores vayan por un camino en particular por lo que es razonable que haya muchos checkpoints frecuentemente, como trazando el camino a hacer.

## Finalización de la carrera

La carrera finaliza cuando todos los autos llegan a la meta. Si hay autos que han quedado descalificados, estos no cuentan. Si la carrera dura más de 10 minutos, esta finaliza automáticamente.
Cada jugador obtiene como tiempo de carrera el tiempo que le llevó completarla menos las penalizaciones.

## Elección y mod del auto

Al iniciar una partida cada jugador elegirá con que auto correrá: cada auto tiene una velocidad, aceleración, salud, masa, controlabilidad máxima.
Por ejemplo, un camión tendrá una velocidad y aceleración menores que un auto deportivo pero tendrá más salud, masa y controlabilidad.

Un mismo jugador deberá mantener el auto elegido para todas las carreras de la partida.

Al finalizar cada carrera y hasta unos 10 segundos de arrancar la siguiente, cada jugador podrá mejorar algunas de las propiedades de su auto. Por ejemplo, podrá hacerlo un *poco* más rápido.
Cada mejora tiene un costo que se computa como una penalización del tiempo de llegada de la siguiente carrera.

## NPCs

En las calles habrán autos que no son parte de la carrera y que están circulando por ellas (o algunos estarán estacionados). El recorrido que hagan debe estar generado al azar (por ejemplo, un auto puede avanzar y continuar avanzando hasta llegar a una esquina y ahí decidir si continuar avanzando o doblar).

## Choques

Cuando un auto choca contra un edificio o contra otro auto, su velocidad y salud decrecen. La severidad del choque depende del ángulo: si un auto choca lateralmente contra un auto, la velocidad y salud apenas decrecen, en cambio, si un auto choca frontalmente contra otro que venía en contramano, el impacto será mucho mayor y por ende, su velocidad y salud se decrecerá considerablemente.

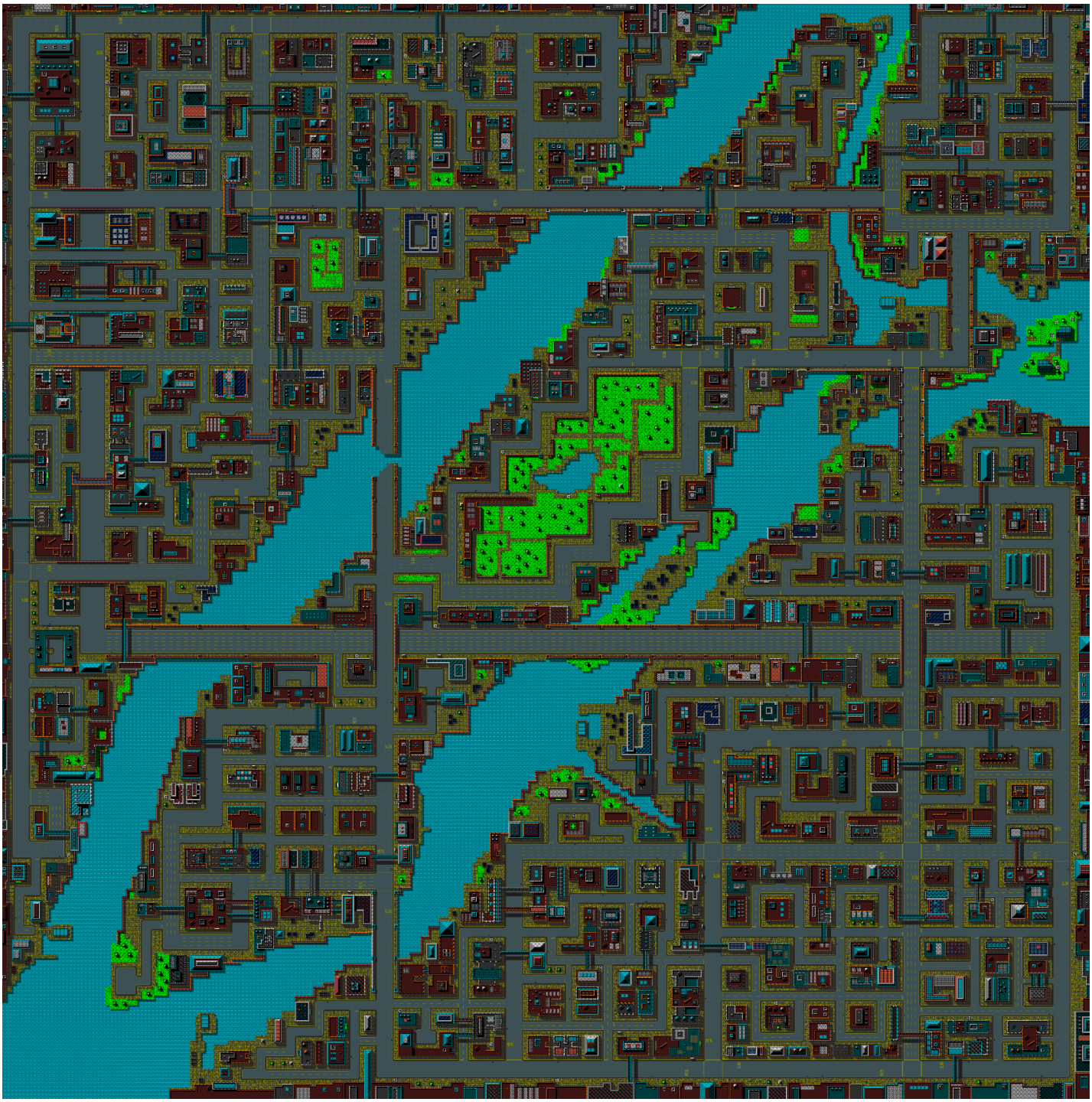
Choques de alto impacto deben mostrar alguna animación o flash para tener un hint visual.
Si la salud de un auto cae a 0, este se frena, explota y el jugador pierde la carrera.

## Ciudades

Habrán 3 ciudades predefinidas en las que los jugadores podrán correr una carrera (Liberty City, San Andreas y Vice City).

Notar que las ciudades tienen puentes que cruzan las calles. Como los autos pasarán por debajo de ellas, es necesario que el cliente gráfico lo renderice como tal.

El cliente gráfico **no** puede renderizar en pantalla toda la ciudad sino solo una parte de ella donde se encuentra el auto del jugador.
El editor en cambio puede renderizar en pantalla toda la ciudad, lo que le facilitara al usuario poder ver el trazado de la carrera globalmente.



## Minimapa

Cada cliente gráfico deberá mostrar en una de las esquinas un minimapa donde se pueda ver el recorrido de la carrera y la posición del auto lo que le facilitar al jugador a orientarse y a ver donde esta el siguiente checkpoint.

## Física del juego

El juego deberá usar **Box2d** como motor físico (no debe ser implementado por ustedes) para los choques. Ver [https://box2d.org/](https://box2d.org/)

### Configuración

Todos los valores numéricos (constantes numéricas) deben venir de un archivo de configuración de texto YAML.

Tener los parámetros en un solo lugar les permitirán hacer modificaciones al juego si este está desbalanceado y hace que el juego no sea divertido.

Un archivo de configuración le permitirá a ustedes hacer *ajustes* más finos sin recompilar y al docente le permitirá cambiar ciertos valores para facilitar la corrección.

**Aclaración:** No se debe implementar un parser YAML, sino que se debe investigar y utilizar uno ya existente.

### Cheats

El juego debe poder aceptar cierta combinación de teclas para activar vida infinita, ganar automáticamente, perder automáticamente, etc. Implementar esto lo más tempranamente posible para que facilite las pruebas (por ejemplo, para probar si la pantalla de victoria funciona correctamente, en vez de “perder el tiempo jugando”, se puede activar el cheat directamente). ***¡Qué trucazo!***

### Sonidos

Como todo juego se debe reproducir sonidos para darle realismo a los eventos y acciones que suceden:

- Cuando hay un choque se debe emitir el sonido característico.
- Cuando se finaliza una carrera.
- Cuando se pega una frenada.

Si la cantidad de eventos que suceden es muy grande, algunos sonidos deben ser evitados para no saturar al jugador con tanta información.

El volumen de los ruidos deberá estar modulado por la distancia: eventos cercanos producirán sonidos o ruidos más fuertes.

*Nota: pueden buscar sonidos en Youtube.*

### Musica

Se debe reproducir una música de fondo.

### Animaciones

Las explosiones, los choques y todo lo que sea dinámico tendrá que ser animado.

## Entregables

- Documentación (véase los detalles en la página web oficial de la materia):

    - Manual del usuario
    - Documentación técnica
    - Manual de proyecto

- Instalador (sea un makefile, cmake o un script the shell) que debe:
    - Descargar las dependencias del proyecto e instalarlas (por ejemplo SDL)
    - Compilar el proyecto
    - Correr los test unitarios
    - Copiar/mover binarios a `/usr/bin`, los assets (como imágenes, audios) a `/var/NAME/` y los archivos de configuración a `/etc/NAME/`, donde `NAME` es el nombre del TP que el grupo decida darle.
      Nota: dentro de `/var/NAME/` o de `/etc/NAME/` pueden haber tantas subcarpetas como lo necesiten/deseen.

- Repositorio en Github con el código, el instalador, documentación y todo el material necesario para la compilación y ejecución.
- Un video promocional de cómo se juega mostrando todos los features implementados (incluyendo al editor). Idealmente sería orientado a la audiencia potencialmente compradora del juego.

### Aplicaciones Requeridas

El proyecto consta de 3 aplicaciones:
- Cliente gráfico
- Servidor
- Editor gráfico

El editor no tiene por qué conectarse al servidor: los niveles editados pueden ser manualmente movidos al servidor. No así el cliente que debe descargarse del servidor el/los niveles a jugar.

**El proyecto debe poderse compilar y ejecutar en un Ubuntu 24.04 (o en su versión más ligera Xubuntu 24.04).**
Este es el mismo entorno que fue usado durante la cursada por el Sercom.

### Unit tests

El protocolo de comunicación debe estar testeado con GoogleTests u otro framework de testing para C++. Los tests pueden mockear los sockets o no.

# Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:
1. El sistema se debe realizar en C++ (C++20) con el estándar POSIX 2008 utilizando librerías SDL y Qt y se permite el uso de QTCreator.

2. El protocolo de comunicación debe ser binario.
3. Los archivos de configuración deben ser en YAML.
4. Todo socket utilizado en este TP debe ser bloqueante (es el comportamiento por defecto).

# Referencias

Imagines (assets): https://9508.cercom.com.ar:62128/course/2025c2/$common/tpfinal_assets/assets.zip