

FACULTAD DE INGENIERÍA DE LA UBA

75.42/95.08 TALLER DE PROGRAMACIÓN I
CÁTEDRA VEIGA

Trabajo Práctico Final Micromachines

Segundo cuatrimestre de 2019

Integrante	Padrón	Correo electrónico
Frutos, Constanza	96728	constanza.frutos.ramos@gmail.com
Iribarren, Álvaro	101049	airibarren@fi.uba.ar
Beroch, Santiago	101135	sberoch@gmail.com

Link al repositorio:

<https://github.com/sberoch/MicromachinesTaller>

Índice

1. Manual de proyecto	2
1.1. División de tareas	2
1.2. Inconvenientes encontrados	2
1.2.1. Servidor	2
1.2.2. Cliente	2
1.2.3. Integración cliente-servidor	2
1.3. Análisis de puntos pendientes	3
1.4. Herramientas	3
2. Documentación Técnica	4
2.1. Requerimientos de Software	4
2.2. Descripción general	4
2.3. Modelado Servidor	4
2.3.1. Modelado del juego	4
2.3.2. Threads utilizados por el servidor	6
2.3.3. Salas	7
2.4. Modelado Cliente	9
2.4.1. Modelado de las diferentes escenas del programa	10
2.4.2. Modelado de la vista del juego	10
2.5. Modelado Cliente-Servidor	10
2.5.1. Manejo de las salas	10
2.5.2. LobbyListener	11
2.5.3. Comunicación	11
3. Manual de usuario	13
3.1. Instalación	13
3.2. Ejecución	13

1. Manual de proyecto

1.1. División de tareas

La división de tareas en el proyecto se realizó de la siguiente manera:

- **Frutos Constanza:** Desarrollo del modelo del servidor, creación del mundo físico, de los autos y la interacción entre los mismos.
- **Iribarren Álvaro:** Desarrollo del modelo cliente-servidor, permitiendo la ejecución de partidas en simultáneo en distintas salas.
- **Beroch Santiago:** Desarrollo del cliente, diseñador de la interfaz gráfica y del script de Lua.

1.2. Inconvenientes encontrados

A continuación se explican algunos de los problemas que fueron surgiendo en el desarrollo del juego, explicando como fueron resueltos.

1.2.1. Servidor

- Modelado de las salas para integrar a los clientes.
- Cierre ordenado del servidor.

1.2.2. Cliente

- Al borrar un objeto del mapa, se borraban tambien aquellos que compartian la textura ya que se la estaba borrando por un mal manejo de punteros.
- Fue dificil encontrar los parametros (velocidad maxima, angulo a partir del cual habria que doblar, etc) adecuados para el correcto funcionamiento del bot de lua.

1.2.3. Integración cliente-servidor

El problema que se tuvo que enfrentar en esta sección fue un delay grande entre que se apretaba una de las flechas de movimiento en el teclado y la

visualización del cambio en pantalla. Este era de entre 4 y 5 segundos. Esto no tenía una única causa.

- Demasiadas notificaciones, por cada evento que llegaba se notificaba a todos los jugadores, cuando en realidad se debe enviar la notificación del cambio una vez vacía la cola de eventos.
- ¿Envío del mapa?

1.3. Análisis de puntos pendientes

- Correcto fin del juego. Escena de fin del juego mostrando en orden los resultados de la carrera. Cierre ordenado del programa
- Correcto funcionamiento del bot de lua.
- Refactorizaciones varias.

1.4. Herramientas

- Clion como IDE para C++.
- Github como plataforma para mantener el proyecto.
- Valgrind para los errores de memoria

2. Documentación Técnica

2.1. Requerimientos de Software

- **Linux Ubuntu:** Versión 18.04.
- **Cmake:** Versión 3.10.2.
- **gcc:** Versión: 7.4.0.
- **SDL2:** Comando: sudo apt-get install libsdl2-dev
- **SDL2 image:** Comando: sudo apt-get install libsdl2-image-dev
- **SDL2 mixer:** Comando: sudo apt-get install libsdl2-mixer-dev
- **Json:** Integrado en el proyecto.
Link: <https://github.com/nlohmann/json>

2.2. Descripción general

El desarrollo del proyecto consta de 3 partes. El primero de ellos es el servidor, que a su vez consta de otras 2 partes. La primera de estas es el modelo del juego, donde se contiene la lógica física del mismo. La segunda es el manejo de los clientes y su introducción en las salas.

La segunda parte es la del Cliente. Este debe enviar la interacción del cliente al servidor y luego actualizar la pantalla según la respuesta.

La tercer parte es la de Bibliotecas y Scripts, donde se llevará a cabo la creación del script de Lua para enseñarle a jugar a un Bot, la grabación de la pantalla con "fembed", etc.

2.3. Modelado Servidor

A continuación se muestran las clases implementadas del lado del servidor para la realización del trabajo.

2.3.1. Modelado del juego

Mas allá de las conexiones de los jugadores, fue necesario el diseño del mundo físico del juego. Esto fue logrado a través del uso de la librería Box2D. En el siguiente esquema se muestran algunas de las clases usadas.

Modelado del juego

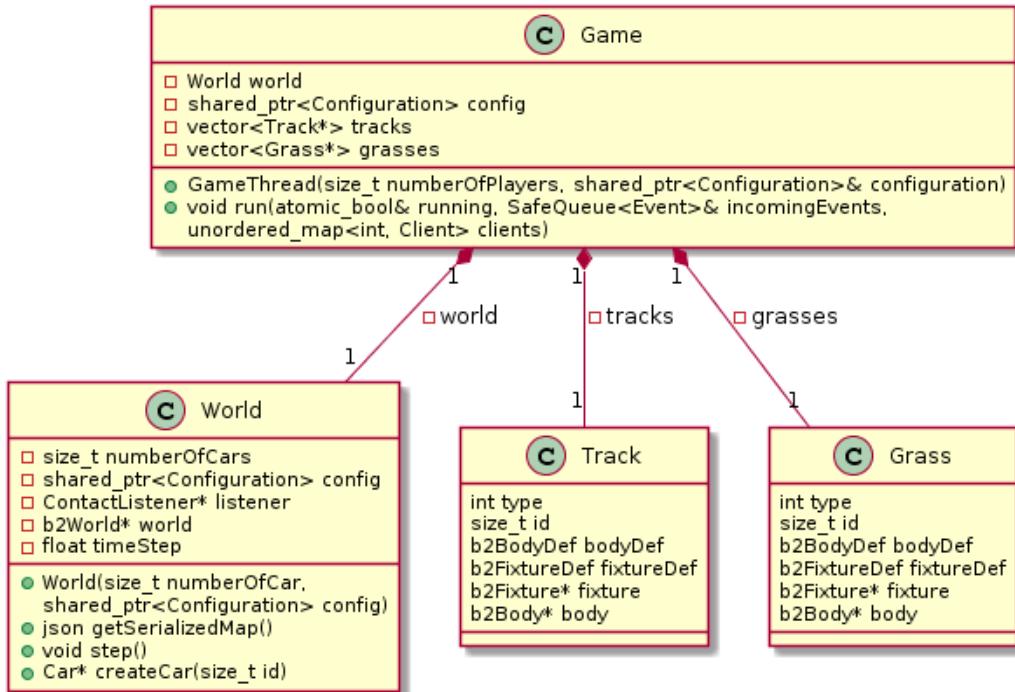


Figura 1: Modelado del juego.

Vemos como el game tiene una única instancia de la clase World. Esta de por sí es muy compleja como para mostrar todos los usos de la librería en un solo diagrama, por lo tanto se deja una demostración extra a continuación.

model.png
Modelado del juego

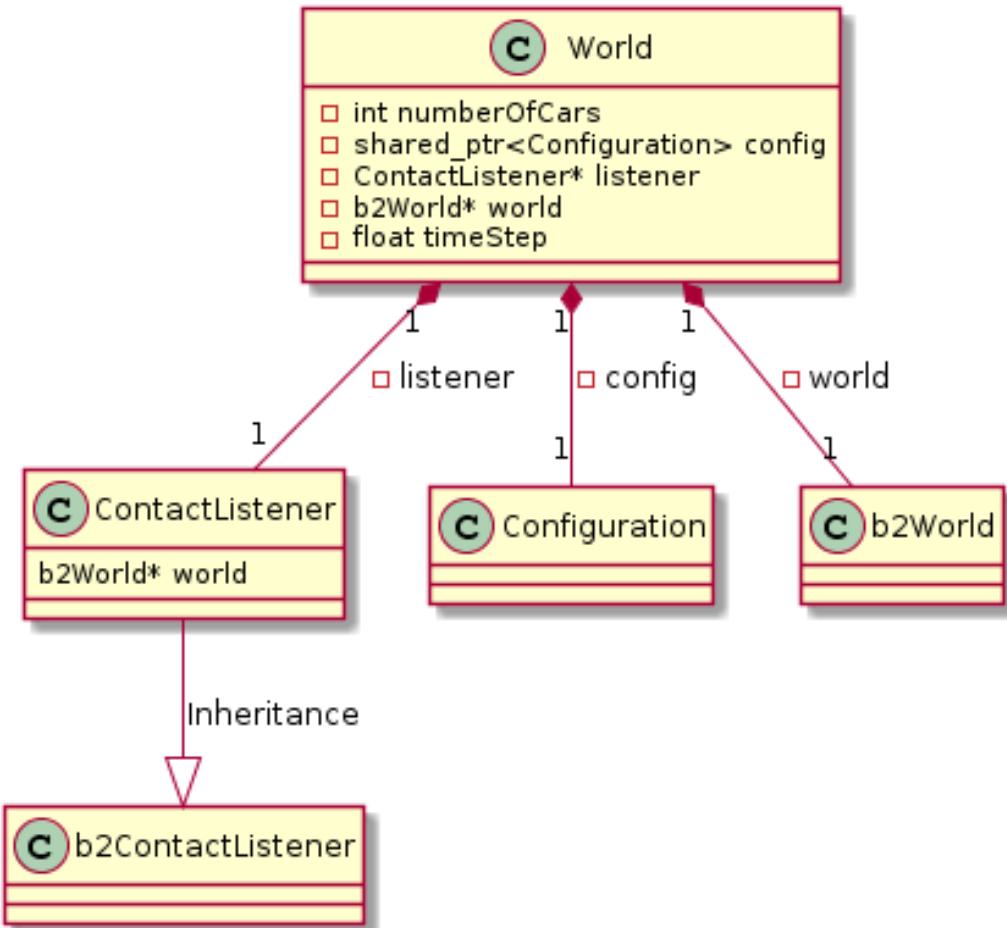


Figura 2: Modelado del mundo.

Esta clase es un wrapper de la librería del world y además posee una instancia de configuración, clase donde se encuentran todos los atributos del mundo físico, y un “ContactListener”, que es también un wrapper. Para no complicar el diagrama y centrar el foco en los hilos, se evitaron las relaciones entre las propias clases.

2.3.2. Threads utilizados por el servidor

Para entender mejor el modelo, se mostrará un diagrama en donde se muestran todos los hilos de ejecución del lado del servidor.

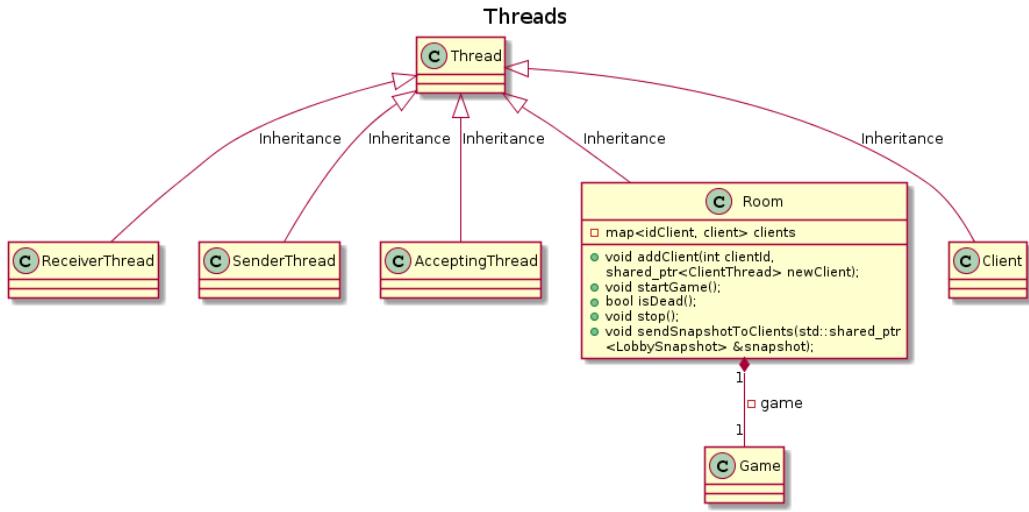


Figura 3: Hilos usados por el servidor.

Primero tenemos el hilo aceptador. Básicamente cada vez que llega un cliente le delega el trabajo al RoomController” del socket recibido. Esto se explica con mas detalle mas adelante. Segundo tenemos el hilo cliente, el cual se ejecuta uno por cada cliente que es aceptado por el servidor. Este a su vez posee dos hilos mas, uno para la recepción constante de eventos y otro para el envío. Tercero tenemos cada sala, el cual es un hilo aparte para poder correr la instancia del juego con los jugadores.

2.3.3. Salas

Se muestra en la siguiente imagen como se diseñaron las salas. Cada una tiene un mapa con los clientes que van a jugar en la misma y una instancia al Game o mundo físico. Es este el encargado de realizar el ciclo del juego.

de una sala.png
Modelo de una sala

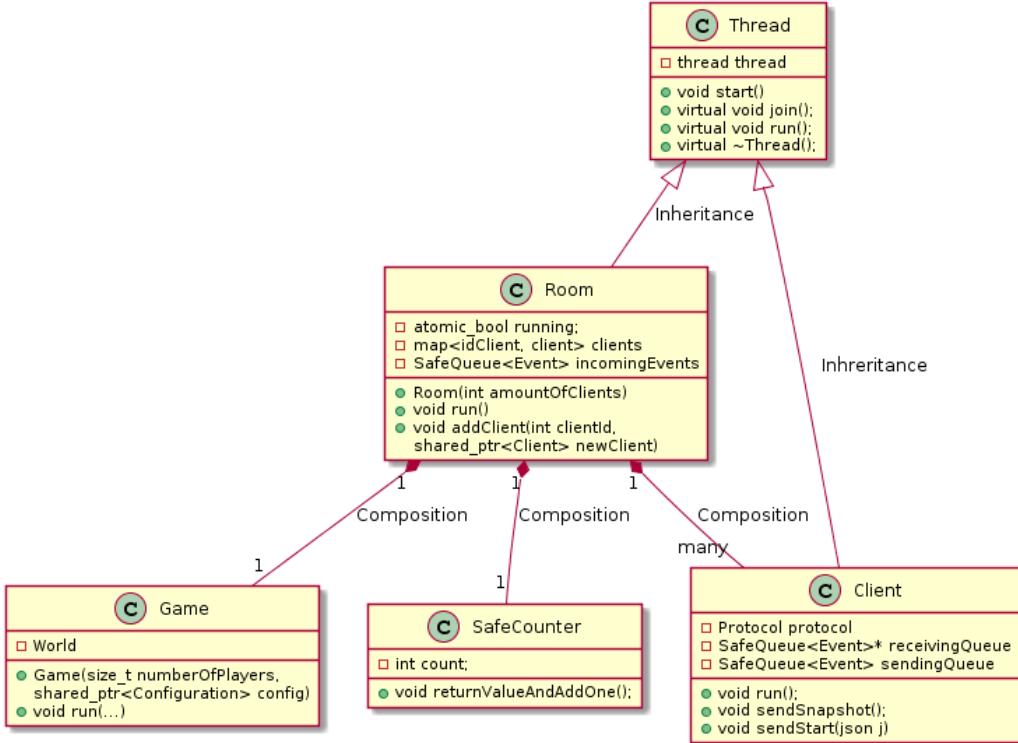


Figura 4: Modelo de una sala.

A continuación se muestra como es realizado el ciclo de juego en código.

```

1 while (acceptSocketRunning && roomRunning) {
2     try {
3         //Creacion de los modificadores.
4
5         std::shared_ptr<SnapshotEvent> snapshot(new
6             SnapshotEvent);
7
8         if (!clients.empty()) {
9             while (incomingEvents.get(event)) {
10                 //Cliente[id].handleInput();
11
12             step();
13
14             for (auto &actualClient : clients) {

```

```

15         actualClient.second->
16     modifySnapshotFromClient(snapshot);
17 }
18     for (auto &actualClient : clients) {
19         actualClient.second->sendSnapshot(snapshot);
20     }
21
22     //wait
23 }
```

2.4. Modelado Cliente

A grandes rasgos, el cliente se ocupa de tres cosas: Actualizarse segun los cambios recibidos del servidor, dibujar la escena actual teniendo en cuenta dichas actualizaciones, y manejar los eventos de mouse y teclado, que se enviaran al servidor. Para la actualizacion, un hilo recibidor de Snapshots del servidor encola los mismos, recibidos por socket desde el servidor, para ser enviados al hilo principal de dibujado. Esta cola es no bloqueante, ya que se deben poder seguir dibujando animaciones aunque no suceda nada. El dibujado es, para cada objeto del juego, renderizarlo en su posicion inicial mas un delta correspondiente a la camara. Los eventos se manejan mediante la funcion PollEvents de SDL, y segun su tipo, se envia dicho evento al servidor. Para esto se crea el evento y se lo inserta en una cola bloqueante, de donde lo levanta un hilo enviador y lo envia por socket.

2.4.1. Modelado de las diferentes escenas del programa

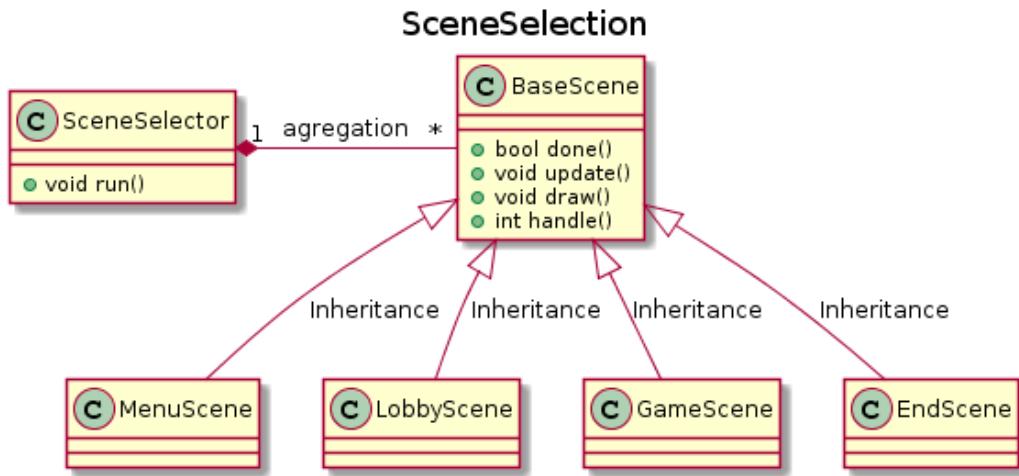


Figura 5: Escenas.

2.4.2. Modelado de la vista del juego

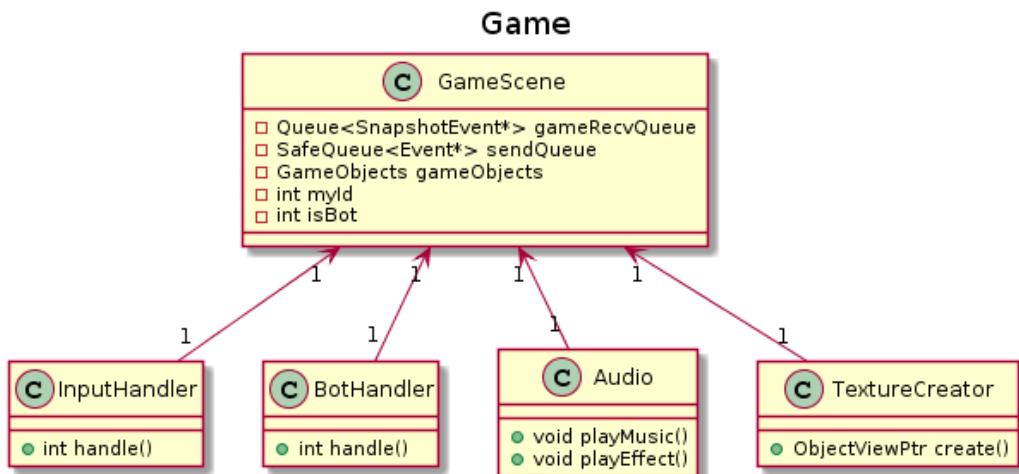


Figura 6: Vista del juego.

2.5. Modelado Cliente-Servidor

2.5.1. Manejo de las salas

Para lograr correr salas en simultaneo lo que hizo fue instanciar un - `roomController`", el cual posee la informacin de los clientes y las salas creadas.

Este se encarga de crear sala y de añadir clientes a una de las mismas, de enviar los eventos ocurridos, etc. Este mismo se instancia en el thread aceptador, el cual le delega el tratado del cliente. Puede comunicarse con un SafeCounter, que es el encargado de asignar las ids cuando se las necesiten. Este mismo se encuentra protegido para evitar condiciones de carreras. En su creación, el RoomController”le da comienzo a otro hilo de ejecución, el ”LobbyListener”.

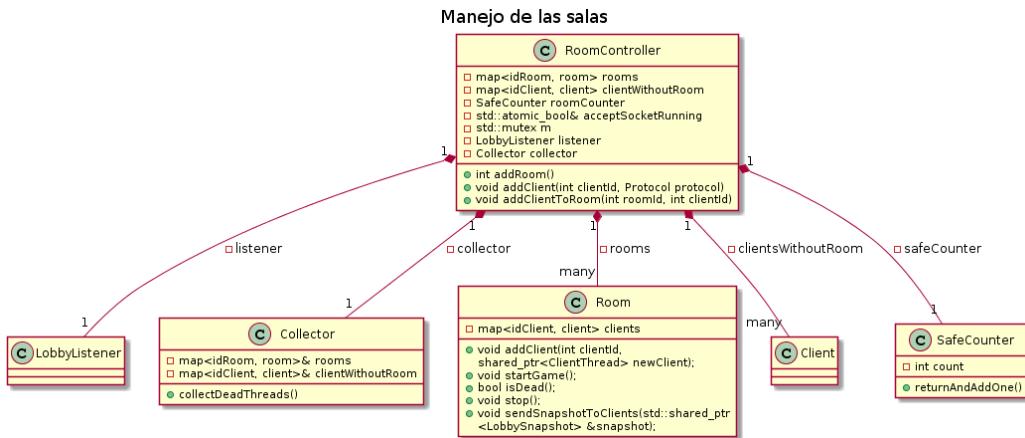


Figura 7: Manejo de las salas.

2.5.2. LobbyListener

Una vez comenzada una partida, el modelo debe ser capaz de seguir aceptando usuarios, seguir creando salas y de comenzar juegos. El encargado de esto es el ”LobbyListener”, que utiliza un recurso de tipo evento llamado ”LobbySnapshot” para que cada cliente actualice su pantalla, mostrando las salas disponibles y los jugadores que hay en cada una de ellas. Al final de cada iteración, el listener se encarga de chequear si alguno de los clientes sufrió una desconexión o si una de las salas sufrió algún inconveniente. De ser este el caso, son retirados ordenadamente.

2.5.3. Comunicación

Para realizar la comunicación cliente-servidor se utilizan los eventos. Cada cola de envío y recepción se manejan con instancias de evento. De los tipos de eventos existentes hay dos muy importantes, el ”LobbySnapshot” el ”SnapshotEvent”. El primero es utilizado para la actualización de la pantalla en el período pre-partida. Se pueden enviar las órdenes de:

- Entrar al lobby general.

- Crear Sala.
- Joinear una sala.
- Seleccionar si jugar o dejar al bot.
- Iniciar una partida.

Una vez que el servidor recibe alguna de estas se crea un snapshot y se envia a cada cliente para que actualice su vista.

El segundo, SnapshotEvent, es utilizado para la comunicación dentro del juego. Se envían los comandos de:

- Acelerar/Frenar/Doblar.
- Explosiones.
- GameOver.
- Barro en pantalla.

El proceso es el mismo, cada jugador del servidor modifica el snapshot con sus valores y luego este mismo es enviado a cada uno de los clientes.

3. Manual de usuario

3.1. Instalación

Para la correcta instalacion del juego se deben seguir los siguientes pasos.

1. Instalación de las dependencias nombradas en la documentación técnica.
2. Abrir terminal en la carpeta base del proyecto.
3. Crear una carpeta build y compilar. Para esto se deben ingresar la siguiente serie de comandos en el orden indicado.

```
1 //Creo carpeta build.  
2 mkdir build  
3 //Ingreso carpeta build  
4 cd build  
5 //Ejecuto el cmake, creando el makefile.  
6 cmake ..  
7 //Compilo el código de source. Puede agregarse  
    -j4 como parametro para acelerar el proceso  
8 make  
9 //Instalo los recursos encontrados en assets.  
10 make install
```

3.2. Ejecución

Una vez completada la sección anterior se generan los ejecutables correspondientes. Para poder correr el ejecutable se debe estar en la carpeta build, de lo contrario se puede ingresar `cd build` desde la carpeta base para ingresar. Siempre debe ser corrido primero el servidor para poder recibir o aceptar clientes.

- Abrir una terminal desde build y ejecutar.

Ejecutar servidor

```
./mm_server
```

- Abrir otra terminal y ejecutar cliente.

Ejecutar servidor

```
./mm_client
```

- First view del cliente.



Figura 8: Pantalla Inicio.

- Una vez abierto el cliente se le debe dar play.

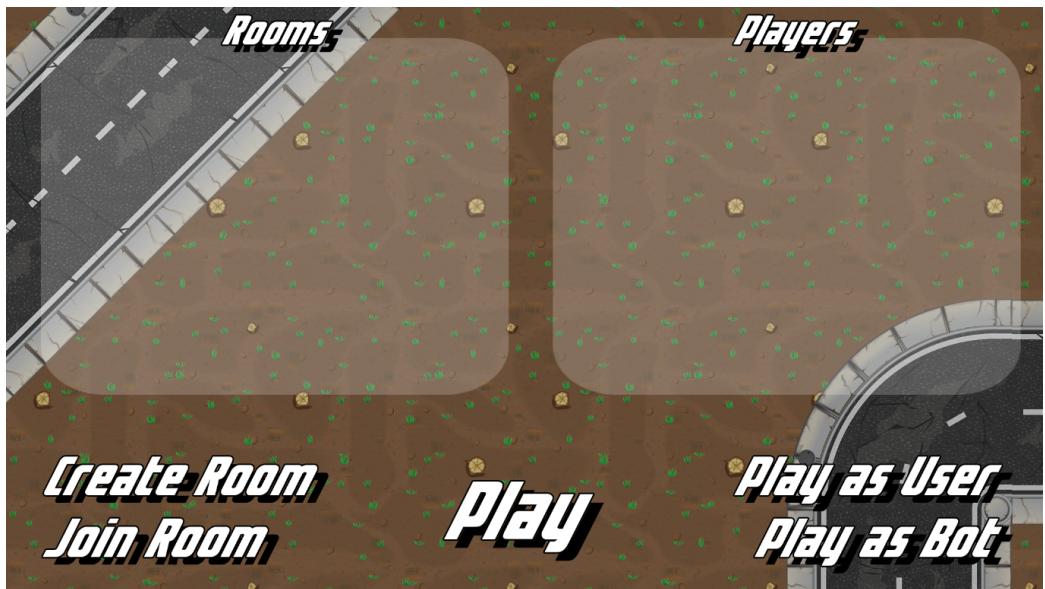


Figura 9: Pantalla principal.

- Crear sala, presionando el botón "Create room"

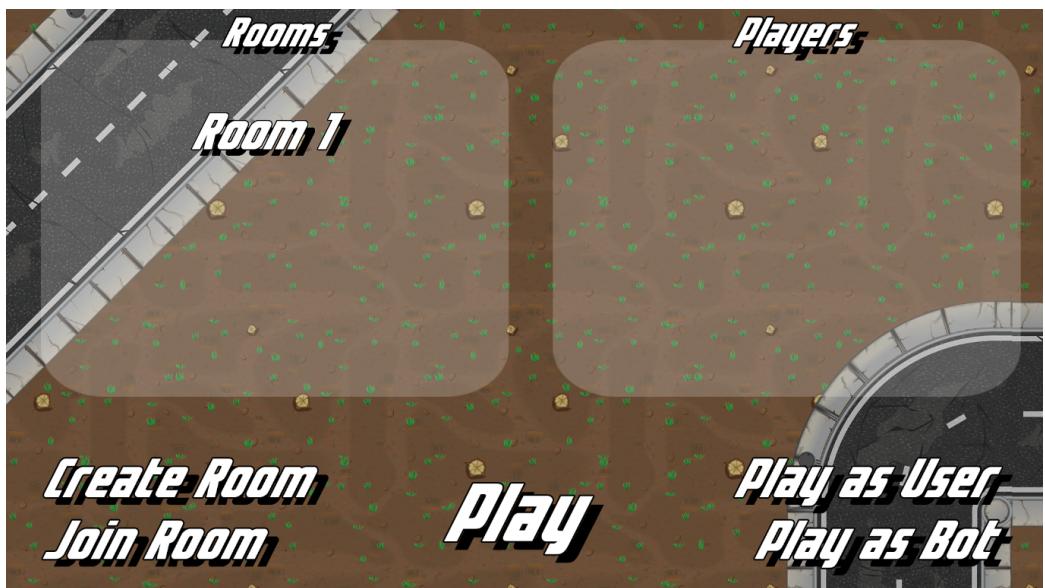


Figura 10: Creación de sala.

- Joinear sala, presiono primero la sala que quiero ingresar y luego "Join room"

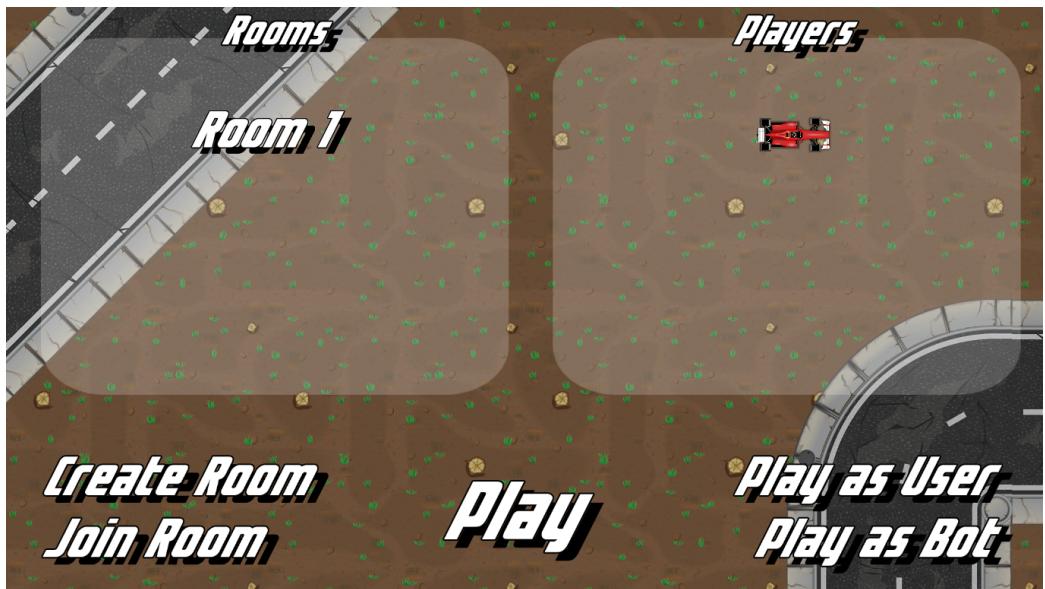


Figura 11: Cliente ingresa sala.

- Una vez dentro de la sala podemos esperar a otros jugadores o presionar play para comenzar la partida.



Figura 12: Cliente ingresa sala.

- Fin del tutorial, a jugar!