Universidad de Buenos Aires
Facultad de Ingeniería
Año 2019 - 1º Cuatrimestre

Taller de Programación I - Trabajo Práctico 4
Portal

Nombre	Padrón
Marino Gonzalo	97794
Mejliker Julian	100866
Saidman Joel	99730

Introducción

Para este trabajo práctico se desarrolló una variante 2D multijugador cooperativo del juego Portal, producido por Valve Corporation en el año 2007. Se trata de un juego de lógica donde los jugadores deben resolver puzzles utilizando un dispositivo de creación de portales inter-espaciales conocido como the Aperture Science Handheld Portal Device.

El objetivo de cada nivel o escenario es poder que todos los jugadores lleguen al pastel. Pero la muerte de uno de los jugadores no implica que el resto no pueda continuar.

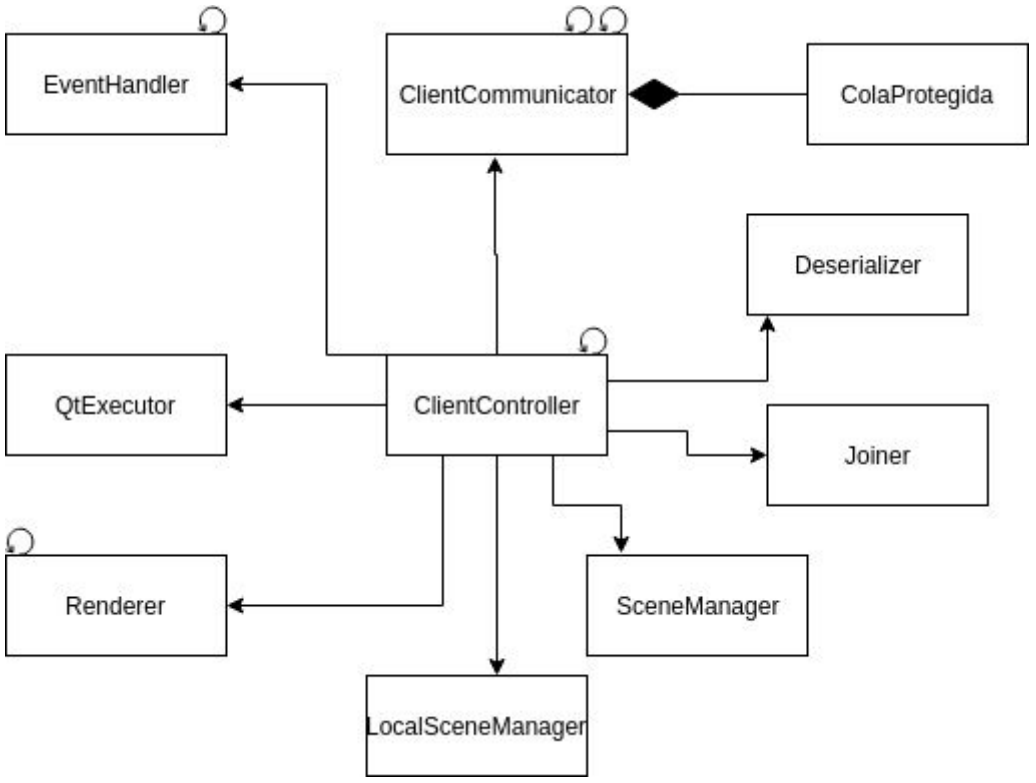
En esta variante del juego se confeccionaron tres aplicaciones: el cliente del juego, el encargado de interactuar con el jugador y mostrarle la interfaz gráfica; el servidor del juego, encargado de administrar las partidas multijugador a través de la red y el editor de mapas que permite crear nuevos mapas de juego.

Distribución

Tarea	Integrante
Cliente - Editor	Julian Mejliker
Cliente - Sonidos	Gonzalo Marino
Cliente - Ffmpeg	Gonzalo Marino
Cliente - Sockets	Joel Saidman / Julian Mejliker
Servidor	Joel Saidman
Servidor - Box2D	Gonzalo Marino

Cliente

El cliente contiene una clase principal encargada de manejar los eventos que van ocurriendo, el siguiente diagrama muestra cómo esta clase, ClientController está conectada con el resto de las clases. Cada flecha circular arriba de las clases representa un hilo ejecutándose.



El ClientController es llamado desde el main del cliente, este lanza primero una venta de qt, manejado por la clase QtExecutor que será la encargada de que el cliente complete sus datos o ingrese al editor de mapa.
Luego, si el cliente completa sus datos tiene la opción de crear o unirse a una sala, cuando alguno de los jugadores de a comenzar el juego comenzará. El Communicator será el responsable de recibir y enviar datos al servidor. El EventHandler es la clase que va leer los comandos y a través de la clase MessegeSender se llamará al Communicator para enviarla.
Las clases LocalSceneManager y SceneManager seran las clases responsables de mostrar la escena correspondiente al momento del juego. Puede haber 3 escenas, la primera es donde el juego ocurre, la segunda es al momento de pasar de nivel, cuando los jugadores ganan el nivel, y la ultima es cuando termina la partida.
El hilo Renderer va ser el encargado de ir renderizando el modelo, mientras que se actualizara en ModelController. Hubo que proteger esta clase ya que había dos hilos accediendo al modelo, uno para actualizar y otro para renderizar.

Vista

Para el desarrollo del cliente se utilizaron las librerías SDL 2.0 Y Qt5.
Para el desarrollo de las texturas, se realizó una base textura, la cual almacena y genera todas las texturas para que luego se soliciten las necesarias. Esto genera que haya un gran ahorro de memoria ya que de esta manera la textura queda cargada para cada clase que deba usarla. Para la clase se utilizó el patrón de diseño Singleton, se decidió utilizar este patrón debido a que cada uno de los componentes de la estructura solicitan estas texturas, ya que la utilización de parámetros hacen que el funcionamiento sea lento.

Por otro lado, con lo que respecta al desarrollo del modelo, se generó un a clase modelo que almacena todos los tipos de elementos que están presente en el videojuego. Estos objetos, dentro poseen los sprites, es decir los distintos fotogramas que representan la animación de los mismo. Los sprites que tienen un tamaño predeterminado utilizan la textura necesaria. La textura es toda la imagen y es por esta razón que posee un atributo src, la cual proviene de una clase llamada Rect que delimita un área en un a posición específica, y de está forma se mostrará la porción de la imagen deseada. Todos los sprites, heredan de la clase abstracta Sprite, la cual renderiza la textura del objeto en la pantalla.

Retomando al tema de control, se desarrolló la clase modelController, la cual se encarga de decodificar los mensajes para luego enviarle al modelo la información necesaria para realizar las distintas acciones recibidas del servidor.

Ventana principal

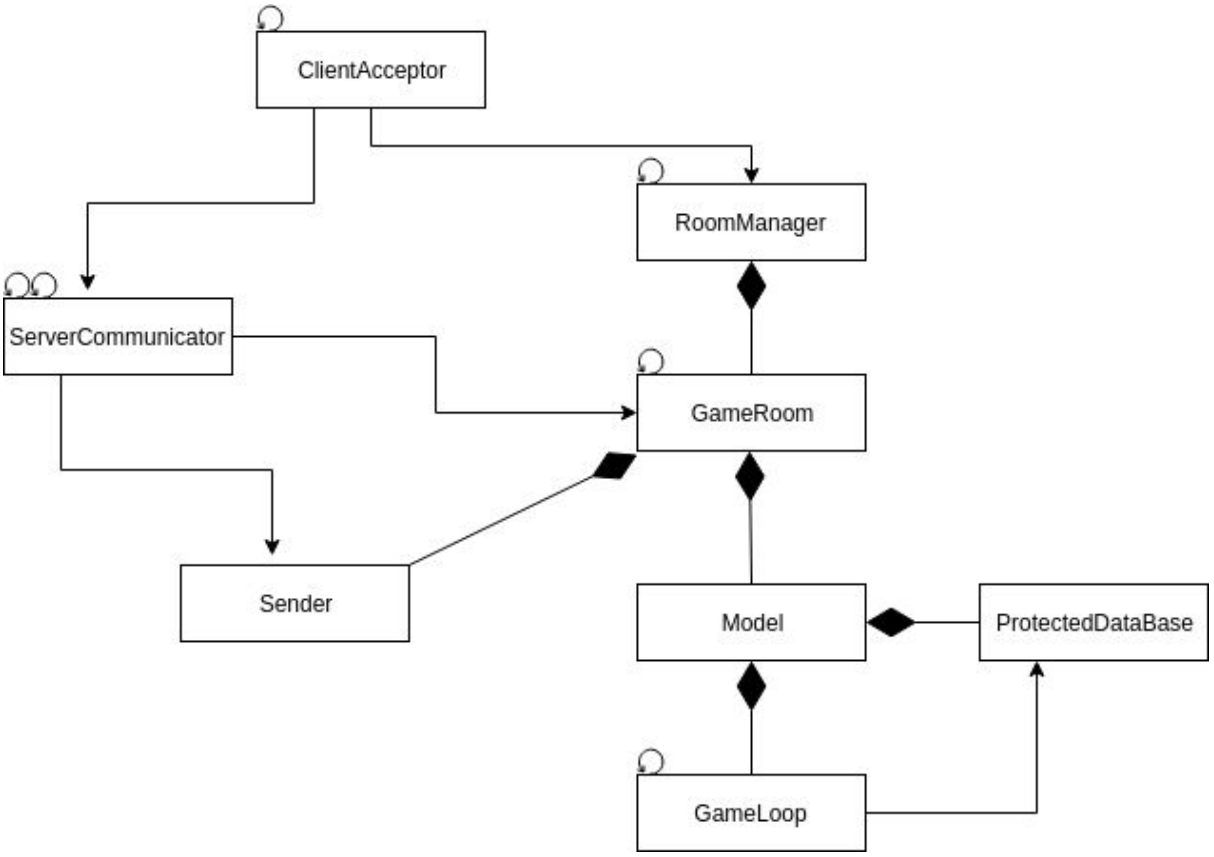
La ventana principal fue desarrollada utilizando Qt5 y Qt Creator.
La ventana principal consiste en toda la primera parte del juego, hasta que se empieza a jugar, es decir la conexión, la elección de sala, la sala de espera y el editor. La aplicación inicia con la opción de jugar o de editar un mapa. Como respecto al editor, es una widget, que dentro tiene una QTableWidget y una QListWidget, la tabla simula un mapa, con casilleros de 1x1 metros, y en la lista están todos los objetos que pueden ser insertados en la tabla.

Ffmpeg

Uno de los requerimientos del trabajo es que el usuario debe poder grabar su partida cuando él lo desee, para esto se utilizó la herramiento ffmpeg. Para la ejecución del mismos se ejecuto el comando "ffmpeg -threads auto -f x11grab -r 30 -s 1920x1080 -i :0.0+0,0 -vcodec libx264 output.mp4". De esta manera se graba la pantalla del usuario.
Para que se pueda grabar la pantalla el juego debe estar en pantalla completa, debido que el comando ejecutado graba toda la pantalla y no una ventana determinada.

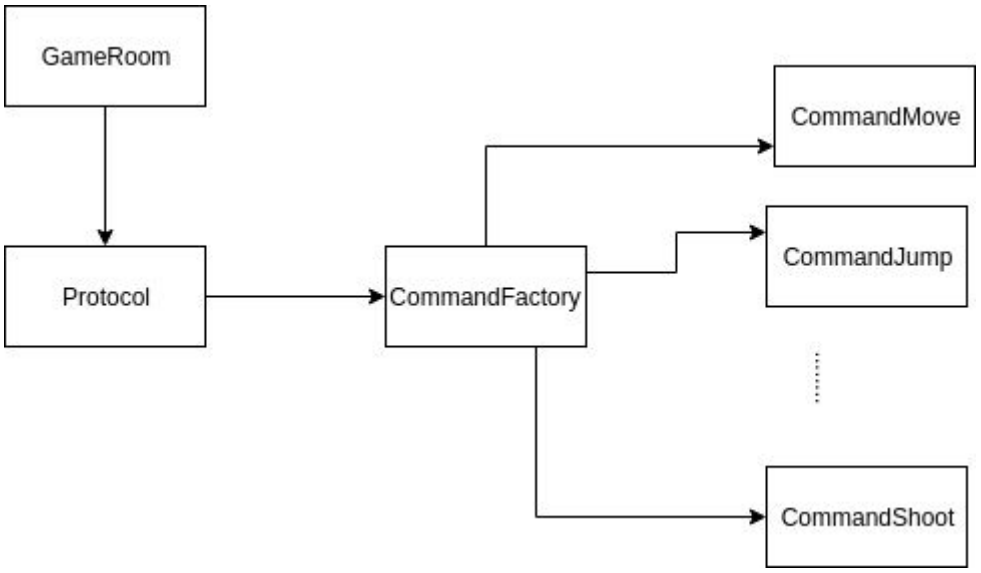
Servidor

El servidor tiene un ClientAcceptor, el cual acepta nuevos clientes. Estos tienen la opción de crear o unirse a una sala. Cuando estos ingresan se verificarán los datos enviados, los cuales son crear o unirse, nombre de jugador y nombre de la sala. En caso de que alguno de estos datos sea incorrecto, se rechazara al jugador. El siguiente diagrama muestra a grandes rasgos la estructura del servidor. Cada flecha arriba de las clases representa un hilo corriendo.



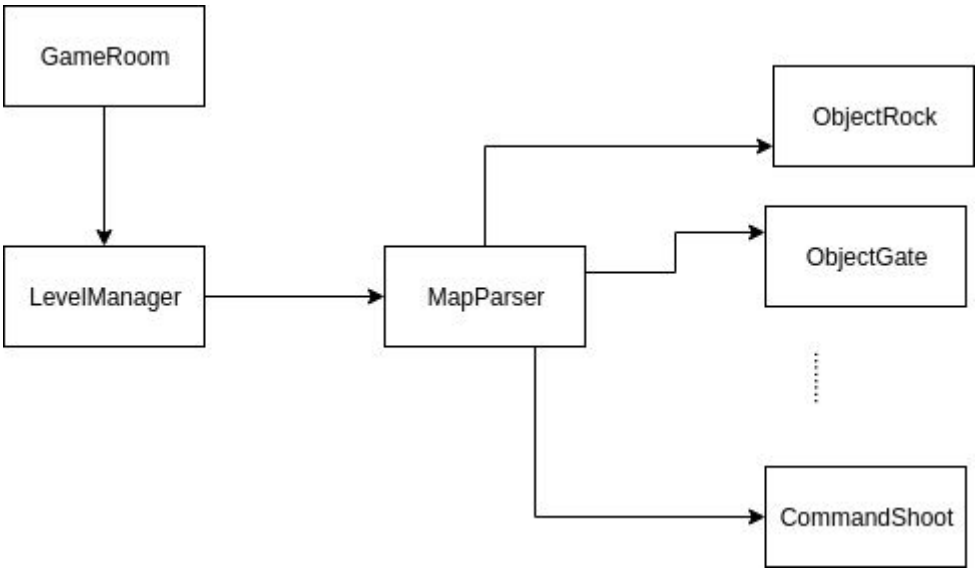
El ServerCommunicator es el responsable de recibir y enviar datos al cliente. Un Client Acceptor contendrá muchos ServerCommunicator, en caso que un jugador se desconecte, este será removido. El RoomManager estará encargado del manejo de las salas, será el que indique si una sala existe o no y si un jugador puede unirse o no a la sala. Contendrá varios GameRoom. Para que un ServerCommunicator envía información a la GameRoom, se encola un mensaje en la cola, que hay una por sala. Cada sala tiene una clase Sender la cual contiene una cola por cada ServerCommunicator. En la misma se encolan los mensajes que se le debe enviar a cada jugador. La clase ProtectedDataBase funciona como monitor de la clase modelo, la que hay dos hilos accediendo a la misma. El hilo del GameLoop y el de RoomGame.

El siguiente diagrama muestra cómo cada sala recibe un mensaje y se convierte en un comando para que se pueda ejecutar en el mundo físico.



El GameRoom llama a una clase protocola la cual le da un mensaje y la misma lo deserializar y le devuelve un Command, creado por la clase CommandFactory. Luego la clase GameRoom llama a su modelo, que será único por cada sala para ejecutar el comando.

El patrón utilizado para crear los mapas fue similar al de ejecutar comandos, hay una clase level manager, responsable de manejar si el nivel termina o si quedan niveles por jugar. En cada nivel LevelManager llama a MapParser, el cual le devuelve un arreglo de objetos que recorre para ir agregando objetos al modelo. El siguiente diagrama muestra la relación entre las clases.



Mundo físico

El juego debe tener una simulación de física para la trayectoria de los objetos y otras dinámicas para esto se utilizó el framework Box2D. Para este segmento del trabajo se implementaron las siguientes clases:

- Acid
- Button
- Cake
- Chell Player:
  Para implementar al personaje se utilizar varias figuras para el cuerpo. se crea un rectángulo que representa el torso, brazos y cabeza, dos esferas que se utilizan para que el personaje se mueva fluido y no se trabe y un cuadrado en los pies para usarlo como sensor para saber cuando este puede saltar.
  Toma de rocas: Utilizando las colisiones, para poder tomar una roca el personaje debe chocar con esta, y si esta tiene activado que debe tomarla, le va asignando a la roca la posición donde debe estar, hasta que la suelta.
  Teletransportación: Para teletransportarse cuando colisiona con un portal le pide a este la ubicación de su pareja, en caso de ser uno azul le pide el portal naranja y viceversa. Además le pide el valor de su normal. Entonces se teletransporta cambiando la ubicación y se asigna la velocidad a la que debe ir según la normal obtenida. (La teletransportación funciona igual en las rocas y en las bolas de energía).
- Energy Ball
- Energy Barrier
- Energy Emitters
- Entity
  Está clase la utlizan como padre, ya que es una clase abstracta para gestionar las colisiones en el mundo.

- Filter Data
  La creación de esta clase se realiza para filtrar colisiones entre los distintos cuerpos, por ejemplo los jugadores no deben chocar con las barreras de energía pero las rocas si pueden colisionar, entonces asignándole los valores con los cual puede colisionar cada clase, estas colisionan o se traspasan.
- Foot Sensor
- Gate
- Ground
- Metal Block
- My Contact Listener
- Player Portals
  Como cada jugador solo puede tener 2 portales en uso, uno de entrada y otro de salida, esta clase almacena los portales de cada jugador.
- Portal:
  Para el desarrollo de los portales, como estos deben crearse según el click del usuario, primero se crea un cuerpo de forma esférica que debe moverse por el mundo hasta que colisione con algún cuerpo. En caso de colisionar con una bloque de piedra, una barrera de energía, el portal desaparece, pero si colisiona con un bloque de metal, o un piso, desaparece la esfera y se crea un cuerpo rectangular que representa al portal al cual puede ingresar un personaje, un bola de energía o una roca. Cuando colisiona con un cuerpo donde puede crearse esta pared bidimensional, la clase utiliza la posición y el punto de contacto del otro cuerpo para saber donde debe crearse. En el caso que el punto de colisión ocurre en el x, el portal debe crearse vertical, en caso de colisión en el eje y se crea horizontal, y en caso de chocar contra una superficie triangular, le pide su ángulo para poder crearse diagonalmente. Al crearse el portal rectangular, se le asigna el valor de la normal para que el cuerpo que se teletransporta pueda saber con velocidad debe salir del portal.
- Rock
- Stone Rock
- World

Editor

Como se mencionó con anterioridad, el editor de mapas, está hecho con QT, para el diseño del mismo se utilizó la aplicación QtCreator, la cual ofrece una forma simple de generar los widgets y todo lo demás. Está planteado con una tabla que simula el mapa, y una tabla con los posibles objetos a agregar. La tabla, posee un contorno generado con bloque de metal, esto delimita el mundo real, este no se puede modificar, lo mismo sucede con la imagen de chell, está colocada en el centro del mismo, mostrando donde inicia la misma, para evitar cualquier error o situación que impida el juego del nivel debido a cosas mal colocadas.
Con lo que respecta a las puertas y a los botones, estos son los elementos, los cuales tienen una lógica atrás a la hora de agregarlas al mapa. Estos se guardan en un mapa, cada objeto tiene dos mapas, uno donde se guarda el objeto, que es de tipo QTableWidgetItem y por otro lado la posición donde está el mismo. Las puerta se guardan con su id, y los botones se guardan en un mapa, donde la key es el id de la puerta a la que están asociados, y de valor una lista de botones y/o posiciones en su defecto. Al momento de eliminar una puerta o un botón, estos son borrados de los mapas donde están los objetos pero no del mapa donde están las posiciones. A la hora de guardar el mapa, se recorre el mapa y se envían los datos con respondientes, a excepción de las puertas y botones, que se las recorre a parte en su mapa correspondiente, se recorre el mapa de objetos y se obtiene la info de las posiciones en el otro mapa.


PORTAL™
MAKER
TEEN
T
ESRB
VALVE