

# Manual del proyecto

## 75:42 - Taller de Programación I

Ejercicio N° \_\_\_\_\_ Padrón \_\_\_\_\_

Alumno \_\_\_\_\_ Firma \_\_\_\_\_

Nota:		Corrige:		Entrega #1
				Fecha de entrega
				Fecha de devolución

Nota:		Corrige:		Entrega #2
				Fecha de entrega
				Fecha de devolución


El presente trabajo, así como la entrega electrónica correspondiente al mismo, constituyen una obra de creación completamente personal, no habiendo sido inspirada ni siendo copia completa o parcial de ninguna fuente pública, privada, de otra persona o naturaleza.

# Índice

1. Integrantes	2
2. Enunciado	3
3. División de tareas	14
4. Evolución del proyecto	14
5. Inconvenientes encontrados	16
6. Análisis de puntos pendientes	17
7. Herramientas	18
8. Conclusiones	18

## 1. Integrantes

Apellido y Nombre	Padrón	Correo electrónico
Funes Federico	98372	fedefunes96@gmail.com
Alejandro Peña	98529	alee.pena.94@gmail.com
Marinaro Santiago	97969	santiagomarinaro1@gmail.com

Github: 

<https://github.com/fedefunes96/worms>

## 2. Enunciado

1º Cuatrimestre 2018  
Ejercicio Final - Worms

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

# Worms

## Ejercicio Final

<b>Objetivos</b>	<ul style="list-style-type: none"><li>• Afianzar los conocimientos adquiridos durante la cursada.</li><li>• Poner en práctica la coordinación de tareas dentro de un grupo de trabajo.</li><li>• Realizar un aplicativo de complejidad media con niveles aceptables de calidad y usabilidad.</li></ul>
<b>Instancias de Entrega</b>	<b>Pre-Entrega:</b> clase 14 (12/06/2018). <b>Entrega:</b> clase 16 (26/06/2018).
<b>Temas de Repaso</b>	<ul style="list-style-type: none"><li>• Aplicaciones Cliente-Servidor multi-threading.</li><li>• Interfaces gráficas con <i>gtkmm/cairo/SDL/qt</i></li><li>• Manejo de errores en C++</li></ul>
<b>Criterios de Evaluación</b>	<ul style="list-style-type: none"><li>• Criterios de ejercicios anteriores.</li><li>• Construcción de un sistema Cliente-Servidor de complejidad media.</li><li>• Empleo de buenas prácticas de programación en C++.</li><li>• Coordinación de trabajo grupal.</li><li>• Planificación y distribución de tareas para cumplir con los plazos de entrega pautados.</li><li>• Cumplimiento de todos los requerimientos técnicos y funcionales.</li><li>• Facilidad de instalación y ejecución del sistema final.</li><li>• Calidad de la documentación técnica y manuales entregados.</li><li>• Buena presentación del trabajo práctico y cumplimiento de las normas de entrega establecidas por la cátedra (revisar criterios en sitio de la materia).</li></ul>

# Índice

[Introducción](#)

[Descripción](#)

[Escenario](#)

[Armas con mira](#)

[Arma de combate cuerpo a cuerpo](#)

[Potencia de disparo variable](#)

[Cuenta regresiva](#)

[Teledirigido](#)

[Trayectoria afectada por el viento](#)

[Municiones](#)

[Daño](#)

[Armas y herramientas](#)

[Bazooka](#)

[Mortero](#)

[Granada Verde](#)

[Granada Roja](#)

[Banana](#)

[Granada Santa](#)

[Dinamita](#)

[Bate de Baseball](#)

[Ataque Aereo](#)

[Teletransportación](#)

[Movimientos de los gusanos](#)

[Gusanos y explosiones](#)

[Turnos y rondas](#)

[Cámara](#)

[Animaciones](#)

[Sonidos](#)

[Musica ambiente](#)

[Interfaz del jugador](#)

[Aplicaciones Requeridas](#)

[Cliente](#)

[Servidor](#)

[Editor](#)

[Distribución de Tareas Propuesta](#)

[Restricciones](#)

[Referencias](#)

# Introducción

El trabajo final consiste en la remake del icónico Worms [1]: un juego multijugador en el que los jugadores controlan un pequeño ejército de gusanos altamente armados en la que se enfrentarán a muerte.

El juego tendrá una simulación de física para la trayectoria de los misiles, las explosiones y otras dinámicas usando el framework Box2D [2].

## Descripción

### Escenario

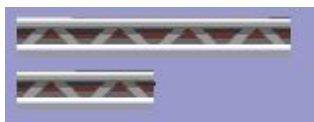
El escenario estará compuesto por vigas indestructibles.

Los gusanos pueden caminar y desplazarse por las vigas horizontales o que tienen una ligera pendiente (menores o iguales a 45 grados).

Las vigas con mayor pendiente y verticales no pueden ser recorridas por los gusanos. Si algún gusano se encontrase sobre una viga tal, el gusano se debe resbalar y caer por acción de la gravedad.

Hay 2 longitudes posibles de vigas: largas (6 mts) y cortas (3 mts) y ambas tienen una altura de 0.80 mts.

Salvo su longitud no tienen ninguna otra diferencia.



La parte inferior del escenario está cubierta de agua. Cualquier gusano que caiga en ella muere ahogado.

El fondo de pantalla debe ser una imagen sin otro objetivo más que la estética.

### Armas con mira

Algunas armas le permiten al jugador apuntar hacia donde quiere disparar.

El jugador puede usar las flechas de arriba (aumenta el ángulo) y abajo (reduce el ángulo) para direccionar la mira.

Los ángulos posibles están en el rango de -90 grados (el gusano apunta verticalmente hacia abajo) y 90 grados (el gusano apunta verticalmente hacia arriba).

Un ángulo de 0 grados apunta horizontalmente.

El jugador puede usar las flechas izquierda y derecha para apuntar hacia la izquierda y derecha respectivamente.

## Arma de combate cuerpo a cuerpo

Simplemente algunas armas producen el daño en el lugar donde está el gusano.

La mayoría de las armas suelen ser a distancia.

## Potencia de disparo variable

Algunas armas permiten un ajuste de la potencia del disparo.

Presionando y manteniendo presionada la tecla Espacio, la potencia se acumula.

Cuando el jugador suelta la tecla o bien se llega a un máximo de potencia, el disparo se produce con una velocidad proporcional a la potencia alcanzada.

## Cuenta regresiva

Los proyectiles de algunas armas no explotan en el instante del impacto sino que lo hacen luego de cierta cantidad de segundos.

El jugador puede prefijar el tiempo de espera presionando algunas de las teclas '1', '2', '3', '4', '5' seleccionando así una cuenta regresiva de 1, 2, 3, 4 o 5 segundos respectivamente.

Por defecto, la cuenta regresiva es de 5 segundos.

## Teledirigido

Algunos proyectiles, e incluso algunas herramientas, no necesitan apuntar; el jugador hace click con el mouse en alguna parte del escenario para determinar el punto exacto donde el proyectil debería caer o la herramienta debería usarse.

## Trayectoria afectada por el viento

Algunos proyectiles son afectados por la dirección y velocidad del viento. El viento puede ir de izquierda a derecha o al revés con una velocidad que varía desde los 0.2 mts/seg a hasta los 10 mts/seg.

## Municiones

Es la cantidad de veces que un gusano puede usar el arma (o herramienta).

## Daño

Cada arma genera un daño **a todos** los gusanos que están en la cercanía del ataque (típicamente una explosión).

El daño es inversamente proporcional al rango de la explosión: a menor distancia del epicentro de la explosión, mayor el daño.

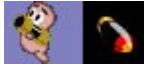
Cada explosión es característica del arma usada y tiene un daño máximo (en el epicentro) y rango particular.

Un gusano que se encuentra en el rango de la explosión adquiere cierta velocidad inversamente



proporcional al rango de la explosión (los gusanos salen volando debido a la explosión) y adquiere cierta dirección.

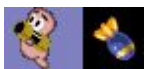
## Armas y herramientas



### Bazooka

Arma predilecta de los gusanos que dispara un misil que estalla al impactar con un tiro parabólico.

Arma con mira: sí  
Combate cuerpo a cuerpo: no  
Disparo con potencia variable: sí  
Cuenta regresiva: no  
Teledirigido: no  
Trayectoria afectada por el viento: si  
Municiones: infinitas  
Daño: 50 pts (epicentro); 2 mts (radio)



### Mortero

Es igual a la bazooka pero al estallar lanza fragmentos al aire con trayectoria parabólica que estallan al impactar.

Arma con mira: sí  
Combate cuerpo a cuerpo: no  
Disparo con potencia variable: sí  
Cuenta regresiva: no  
Teledirigido: no  
Trayectoria afectada por el viento: sí  
Municiones: 10  
Daño (del estallido principal): 50 pts (epicentro); 2 mts (radio)  
Daño (de cada fragmento): 10 pts (epicentro); 2 mts (radio)  
Cantidad de fragmentos: 6

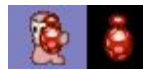


### Granada Verde

La segunda arma predilecta de los gusanos. Como otros tipos de granada no se ve afectada por el viento.

Arma con mira: sí  
Combate cuerpo a cuerpo: no

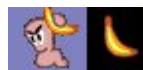
Disparo con potencia variable: sí  
Cuenta regresiva: sí  
Teledirigido: no  
Trayectoria afectada por el viento: no  
Municiones: infinitas  
Daño: 30 pts (epicentro); 2 mts (radio)



### Granada Roja

Al igual que el Mortero, al explotar lanza fragmentos al aire con tiro parabólico que estallan al impacto.

Arma con mira: sí  
Combate cuerpo a cuerpo: no  
Disparo con potencia variable: sí  
Cuenta regresiva: sí  
Teledirigido: no  
Trayectoria afectada por el viento: no  
Municiones: 10  
Daño (del estallido principal): 30 pts (epicentro); 2 mts (radio)  
Daño (de cada fragmento): 10 pts (epicentro); 2 mts (radio)  
Cantidad de fragmentos: 6



### Banana

Es un tipo de granada que tiene la particularidad de rebotar varias veces de forma muy elástica hasta explotar.

Arma con mira: sí  
Combate cuerpo a cuerpo: no  
Disparo con potencia variable: sí  
Cuenta regresiva: sí  
Teledirigido: no  
Trayectoria afectada por el viento: no  
Municiones: 5  
Daño: 70 pts (epicentro); 4 mts (radio)



### Granada Santa

Una de las armas más poderosas en el arsenal. Justo antes de estallar, produce un sonido característico.

Arma con mira: sí  
 Combate cuerpo a cuerpo: no  
 Disparo con potencia variable: sí  
 Cuenta regresiva: sí  
 Teledirigido: no  
 Trayectoria afectada por el viento: no  
 Municiones: 2  
 Daño: 110 pts (epicentro); 8 mts (radio)



### Dinamita

Al activarse, el gusano deja en el lugar una dinámica activa que estalla luego de cierta cantidad de segundos.

Arma con mira: no  
 Combate cuerpo a cuerpo: sí  
 Disparo con potencia variable: no  
 Cuenta regresiva: sí  
 Teledirigido: no  
 Trayectoria afectada por el viento: no  
 Municiones: 5  
 Daño: 50 pts (epicentro); 4 mts (radio)



### Bate de Baseball

Al activarse el arma, el gusano batea lanzando a todo aquel cercano. La dirección de los gusanos lanzados dependerá de la dirección del bateo.

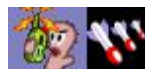
Arma con mira: sí

## Movimientos de los gusanos

Los gusanos pueden desplazarse con las flechas izquierda y derecha a una velocidad pequeña de 0.2 mts/seg. Presionando Enter, el gusano puede dar un salto hacia adelante de 1 mts y de 0.5 mts de alto. Presionando Retroceso, el gusano de una vuelta hacia atrás de 0.2 mts y de 1.2 mts de alto.

Un gusano puede caer de una altura de 2 mts sin sufrir daño alguno y podrá continuar moviéndose. Distancias mayores producen un daño proporcional a la altura (1 punto por metro) con un máximo de 25

Combate cuerpo a cuerpo: sí  
 Disparo con potencia variable: no  
 Cuenta regresiva: no  
 Teledirigido: no  
 Trayectoria afectada por el viento: no  
 Municiones: infinitas  
 Daño: 10 pts



### Ataque Aereo

Caen del cielo 6 misiles hacia el objetivo marcado por el jugador cada uno de ellos explotando independientemente. Los misiles pueden impactar antes si se encuentran en su trayectoria con algún otro objeto.

Arma con mira: no  
 Combate cuerpo a cuerpo: no  
 Disparo con potencia variable: no  
 Cuenta regresiva: no  
 Teledirigido: sí  
 Trayectoria afectada por el viento: sí  
 Municiones: 2  
 Daño (por cada misil): 40 pts (epicentro); 2 mts (radio)



### Teletransportación

Al activarse la herramienta, el gusano puede teletransportarse a cualquier parte del escenario (salvo el interior de una viga).

Arma con mira: no  
 Combate cuerpo a cuerpo: no  
 Disparo con potencia variable: no  
 Cuenta regresiva: no  
 Teledirigido: sí  
 Trayectoria afectada por el viento: no  
 Municiones: infinitas  
 Daño: ninguno

puntos.

Un gusano no puede empujar a otro al desplazarse o saltar. De hecho el otro gusano permanece quieto en el lugar.

## Gusanos y explosiones

Un gusano saldrá disparado por efecto de una explosión si este se encuentra en su rango.

Al salir disparado, el gusano sufre una fuerza y adquiere velocidad: la dirección y magnitud dependen de la distancia y dirección del epicentro de la explosión. Vease la sección "Daño".

En su vuelo, el gusano puede rebotar contra las paredes.

## Turnos y rondas

El juego es por turnos: un jugador puede tener múltiples gusanos pero el turno del jugador sólo le permitirá interactuar y controlar a uno de ellos, siendo el gusano elegido automáticamente de manera cíclica.

El turno del jugador termina si alguna de las siguientes condiciones se cumple:

- si el gusano activo sufre algún daño.
- si el gusano dispara o hace uso de una herramienta.
- si pasan más de 60 segundos.

Si el gusano activo dispara o hace uso de una herramienta, el jugador dispone de 3 segundos adicionales para continuar moviendo al gusano.

Transcurridos estos segundos el turno del jugador termina, pero el turno del siguiente jugador no arranca inmediatamente sino que el juego debe esperar a que todos los proyectiles impacten, las explosiones sucedan y los gusanos queden quietos (posiblemente se moverán debido a alguna explosión).

Recién en ese instante, el juego le da control al siguiente jugador y se da por comenzado su turno.

Una vez que todos los jugadores jugaron un turno cada uno, se termina una ronda.

Inmediatamente comienza la siguiente ronda volviendo cada jugador a jugar un nuevo turno en el mismo orden de la ronda anterior.

Si un jugador perdió, este es saltado en la ronda. El jugador que perdió puede seguir viendo la partida pero no puede hacer nada más en ella.

## Cámara

La cámara muestra una porción del escenario (los escenarios pueden ser muy largos y no entrar en la vista de la cámara) y debe enfocarse en el gusano activo y seguirlo a medida que se desplaza.

El jugador puede mover la cámara con el mouse para poder ver otras partes del escenario pero la cámara volverá a enfocarse en el gusano en cuanto este se mueva o intente disparar.

Cuando el gusano activo dispara un proyectil, la cámara debe seguir al proyectil hasta que este impacte.

Tras el impacto, la cámara debe seguir la posición de algún gusano que se esté moviendo (posiblemente debido al impacto).

Cuando este gusano termine de moverse (o se muera), la cámara debe elegir a otro gusano en movimiento y continuar así hasta que ninguno se mueva.

## **Animaciones**

El juego no debe mostrar imágenes estáticas sino pequeñas animaciones para darle mayor realismo [3]:

- El movimiento de los gusanos: cuando se desplazan, saltan, vuelan entre otros.
- El movimiento de los proyectiles
- Las explosiones.

## **Sonidos**

Como todo juego se debe reproducir sonidos para darle realismo a los eventos y acciones que suceden[4]:

- Cuando hay disparos.
- Cuando hay una explosión.
- Cuando un gusano da un salto
- Cuando un gusano muere

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

## **Musica ambiente**

El juego debe reproducir una música ambiente, con un volumen relativamente bajo[5].

## **Interfaz del jugador**

Se debe mostrar la parte del mapa que el jugador está viendo permitiéndole moverse al desplazar el mouse como lo explicado en la sección Cámara.

Cada jugador tendrá un color asociado.

La vida de cada gusano debe mostrarse encima de cada uno de ellos y debe mostrarse con el mismo color del jugador para poder identificar qué gusano es de quien.

También debe poderse ver la suma total de vidas de los gusanos por cada jugador.

El jugador deberá poder seleccionar qué arma o herramienta usar en su turno.

Al finalizar el escenario se deberá mostrar una pantalla de victoria o derrota dependiendo de cada caso.

# Aplicaciones Requeridas

## Cliente

Se deberá implementar un cliente gráfico para que el usuario pueda conectarse al servidor, crear o unirse a una partida eligiendo el escenario a jugar.

## Servidor

Se deberá implementar un servidor con soporte de múltiples partidas en simultáneo. Deberá poder indicarle a los clientes que se conecta qué escenarios hay disponibles así como también que partidas ya están creadas y están disponibles para que el usuario pueda unirse a alguna de ellas.

Al momento de iniciar la partida el servidor deberá asignar de forma aleatoria a cada jugador los gusanos disponibles en el escenario.

En caso de que la cantidad de gusanos no sea divisible por la cantidad de jugadores, los gusanos de los jugadores con menos cantidad de gusanos tendrán un +25 puntos de vida para compensar.

Todos los atributos de los gusanos (velocidad, altura de salto, etc), de las armas (daño, rango, municiones, etc) y cualquier otro parámetro deben ser configurables por archivo.

*Es importante que todos los parámetros sean configurables: permite que se ajusten para tener un juego más balanceado y divertido a la vez que le permite a los docentes realizar pruebas.*

## Editor

Se deberá implementar un editor de escenarios que permita:

- Elegir el fondo de pantalla.
- Colocar las vigas y gusanos.
- Definir qué armas y herramientas pueden usarse y la cantidad de municiones de ellas (sobre escribiendo los valores por defecto).
- Vida inicial de los gusanos.

Además deberá realizar chequeos de consistencia para evitar errores por parte del diseñador como un escenario sin gusanos.

# Distribución de Tareas Propuesta

Con el objetivo de organizar el desarrollo de las tareas y distribuir la carga de trabajo, es necesario planificar las actividades y sus responsables durante la ejecución del proyecto. La siguiente tabla plantea una posible división de tareas de alto nivel que puede ser tomada como punto de partida para la planificación final del

trabajo:

	<b>Alumno 1 Servidor - Modelo</b>	<b>Alumno 2 Cliente - Modelo</b>	<b>Alumno 3 Cliente - Editor</b>
<b>Semana 1</b> (01/05/2018)	- Draft del modelo (incluyendo lógica del juego y partidas multijugador) - Prueba de concepto con Box2D.	- Mostrar una imagen. - Mostrar una animación. - Mostrar ambas en un lugar fijo o desplazándose por la pantalla (movimiento).	- Draft del cliente y del editor ( <i>wireframe</i> ). - Determinar a partir del draft qué mensajes se necesitarán en el protocolo de comunicación.
<b>Semana 2</b> (08/05/2018)	- Modelado del escenario, los gusanos, los disparos y explosiones.	- Dibujado del escenario. - Dibujado de animaciones (gusanos, disparos, explosiones) - Controles por teclado.	- Editor, creación de escenarios.
<b>Semana 3</b> (15/05/2018)	- Finalización del modelado del juego.	- Finalización de la gráfica del escenario. - Controles por mouse. - Scrolling.	- Finalización del editor. - Carga y guardado de los escenarios (archivos).
<b>Semana 4</b> (22/05/2018)	- Lógica para la creación de partidas multijugador y múltiples partidas.	- Interfaz gráfica al seleccionar un arma o herramienta.	- Implementación del sistema de sonidos y música.
<b>Semana 5</b> (29/05/2018)	- Finalización de la implementación multijugador.	- Interfaz para la conexión con el servidor, elegir un nivel, crear/unirse a una partida.	- Creación de escenarios de prueba y preparación de la demo. - Pantallas de victoria y derrota.
<b>Semana 6</b> (05/06/2018)	- Testing - Correcciones y <i>tuning</i> del Servidor - Documentación	- Testing - Correcciones y <i>tuning</i> del Cliente - Documentación	- Testing - Correcciones y <i>tuning</i> del Editor - Documentación
<b>Preentrega el 12/06/2018</b>			
<b>Semana 7</b> (12/06/2018)	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación
<b>Semana 8</b> (19/06/2018)	- Testing - Correcciones sobre Preentrega - Armado del entregable	- Testing - Correcciones sobre Preentrega - Armado del entregable	- Testing - Correcciones sobre Preentrega - Armado del entregable
<b>Entrega el 26/06/2018</b>			

# Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema se debe realizar en C++11 utilizando librerías *gtkmm*, *SDL* y/o *qt*.
2. Los archivos de configuración deben ser almacenados en formato YAML. A tal fin, y con el objetivo de minimizar tiempos y posibles errores, se permiten distintas librerías externas (consultar sitio de la cátedra). No está permitido utilizar una implementación propia de lectura y escritura de YAML.
3. Para la simulación de la física del juego se debe usar el framework Box2D [2].
4. Es condición necesaria para la aprobación del trabajo práctico la entrega de la documentación mínima exigida (consultar sitio de la cátedra). Es importante recordar que cualquier elemento faltante o de dudosa calidad pone en riesgo la aprobación del ejercicio.
5. Entrega de uno o varios escenarios con la suficiente diversidad de elementos a tal fin que sea fácil mostrar las funcionalidades implementadas.
6. De forma opcional, se sugiere la utilización de alguna librería del estilo xUnit [7]. Si bien existen varias librerías disponibles en lenguaje C++ [8], se recomienda optar por CxxTest [9] o CppUnit [10].

# Referencias

- [1] Worms: [https://es.wikipedia.org/wiki/Worms\\_\(serie\)](https://es.wikipedia.org/wiki/Worms_(serie))
- [2] Box2D: <http://box2d.org/manual.pdf>
- [3] Sprites: [https://www.sprisers-resource.com/pc\\_computer/wormsgeddon/sheet/13597/?source=genre](https://www.sprisers-resource.com/pc_computer/wormsgeddon/sheet/13597/?source=genre)
- [4] Sonidos: <https://www.youtube.com/watch?v=F9YMb0M89DI>
- [5] Musica ambiente:  
<https://www.youtube.com/watch?v=Yc-M2OVlkUs&index=2&list=PL2CAB64D6B77AD3ED>
- [6] YAML: <https://es.wikipedia.org/wiki/YAML>
- [7] Frameworks XUnit: <http://en.wikipedia.org/wiki/XUnit>
- [8] Variantes XUnit para C/C++: [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks#C.2B.2B](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C.2B.2B)
- [9] CxxTest: <http://cxxtest.com/>
- [10] CppUnit: [http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main\\_Page](http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main_Page)

### 3. División de tareas

Se siguió la recomendación de división de tareas propuesta por la cátedra, siendo la distribución de la siguiente manera:

- **Funes Federico:** Se encargó de realizar el modelo del servidor, más específicamente de la lógica del juego multijugador por turnos (con varias salas en diferentes hilos de ejecución), el sistema de rondas, los proyectiles, los gusanos y el dibujado del escenario usando el framework Box2D.
- **Alenadro Peña:** Se encargó de realizar el modelo del cliente, más específicamente la interfaz gráfica de un usuario que se conecta al servidor para jugar al proyecto, incluyendo animaciones, protocolos de comunicación del cliente hacia el servidor, y, pantallas de carga, derrota y victoria.
- **Santiago Marinaro:** Se encargó de realizar el modelo del editor de mapas, más específicamente, un editor que permite crear mapas para el juego y la implementación de los sonidos para la interfaz gráfica del cliente.

### 4. Evolución del proyecto

Se intentó seguir el cronograma de tiempo propuesto por la cátedra, sin embargo, por inconvenientes técnicos y atrasos, el cronograma real se distribuyó de la siguiente manera:



	<b>Federico Funes</b>	<b>Alejandro Peña</b>	<b>Santiago Marinaro</b>
	<b>Server - Modelo</b>	<b>Cliente - Modelo</b>	<b>Editor</b>
<b>Semana 1</b> <b>(01/05/2018)</b>	- Draft del modelo (incluyendo lógica del juego y partidas multijugador) -Prueba de concepto de Box2D.	-Mostrar una imagen -Mostrar una animación -Mostrar ambas en un lugar fijo o desplazándose por la pantalla (movimiento).	- Draft del cliente y del editor ( wireframe ). - Determinar a partir del draft qué mensajes se necesitarán en el protocolo de comunicación.
<b>Semana 2</b> <b>(08/05/2018)</b>	- Modelado del escenario, los gusanos, los disparos y explosiones.	- Dibujado del escenario. - Dibujado de animaciones (gusanos, disparos, explosiones) - Controles por teclado.	- Editor, creación de escenarios.
<b>Semana 3</b> <b>(15/05/2018)</b>	-Correcciones del escenario, los gusanos, las disparos y explosiones.	- Finalización de la gráfica del escenario. - Controles por mouse. - Scrolling.	- Finalización del editor. - Carga y guardado de los escenarios (archivos).
<b>Semana 4</b> <b>(22/05/2018)</b>	-Correcciones generales del modelo y conexión con el cliente.	-Correcciones generales del cliente y conexión con el servidor.	- Implementación del sistema de sonidos y música.
<b>Semana 5</b> <b>(29/05/2018)</b>	- Lógica para la creación de partidas multijugador y múltiples partidas. - Correcciones del modelo.	Interfaz gráfica al seleccionar un arma o herramienta.	- Creación de escenarios de prueba y preparación de la demo. - Pantallas de victoria y derrota.
<b>Semana 6</b> <b>(05/06/2018)</b>	- Finalización de la implementación multijugador. - Lectura de yaml del editor - Testing - Correcciones y tuning del Servidor - Documentación	- Interfaz para la conexión con el servidor, elegir un nivel, crear/unirse a una partida. - Testing - Correcciones y tuning del Cliente - Documentación  - Inserción de animaciones y efectos	- Testing - Correcciones y tuning del Editor - Documentación
<b>Semana 7</b> <b>Preentrega</b> <b>(12/06/2018)</b>	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación
<b>Semana 8</b> <b>(19/06/2018)</b>	- Testing - Correcciones sobre Preentrega - Armado del entregable	- Testing - Correcciones sobre Preentrega - Armado del entregable	- Testing - Correcciones sobre Preentrega - Armado del entregable
<b>(26/06/2018)</b>		<b>Entrega final</b>	

Cuadro 1: Cronograma real

## 5. Inconvenientes encontrados

Se listaran a continuación los inconvenientes encontrados tanto para el modelo del cliente, como para el servidor y el editor y su funcionamiento en conjunto:

### ■ Server:

- Los valores estándar propuestos para la física del juego no son consistentes con lo que se esperaría de un juego con físicas maso menos reales, hubo que cambiar todos los valores.
- Implementar el salto trajo muchos inconvenientes para verificar cuando el gusano realmente podía saltar evitando bugs como: Saltar muchas veces en el aire, saltar muchas veces por estar chocando una pared, saltar mientras se está deslizando.
- El deslizar del gusano no fue algo fácil de implementar en Box2D, ya que no hay forma eficiente de usar una fricción que permita deslizar al gusano por una viga y no quedarse trabado en el contacto contra, por ejemplo, una viga en vertical. De manera que tuvo que deshabilitarse la gravedad del mismo cuando contacta con un objeto con un cierto ángulo y evitar bugs con otro tipo de eventos.
- La creación de una cola eventos trajo muchos inconvenientes por el tema de que no se pueden mover condition variable y, por lo tanto, no se pueden mover colas de eventos.
- La creación de proyectiles que se utilizan en el mismo lugar que el gusano (Ej: dinamita) trajo consigo el problema de que el gusano tiene que poder traspasar el proyectil hasta que sale del tamaño de su radio, donde ahora si puede colisionar con la misma.
- El esperar a que los objetos del mundo dejen de moverse para terminar el turno.

### ■ Cliente:

- Problema con la visualización gráfica de Qt5, ya que es requisito para el dibujado que este se ejecute siempre en el hilo principal, y se había intentado crear un hilo que maneje el dibujado del juego y otro que esté a la escucha permanente de paquetes del servidor, y transmita dichos mensajes al juego, pero no era posible con referencias de las clases, ya que se terminaba ejecutando bajo el hilo en el cual era llamado. Si-guiente se intento hacer una cola de eventos en la cual el hilo que escucha al servidor iría agregando eventos, y en el otro hilo se irían descolando estos eventos y proce-sando, pero el problema persiste ya que no estaban en el hilo principal. Se solucionó finalmente conectando señales entre la clase que se encargaba de la visualización, creada en el hilo principal, y el hilo que recibe paquetes del servidor. De esta forma, cuando es emitida una señal desde el segundo hilo, en el hilo principal es atrapada, y si está conectada con alguna función/método, se llama a dicha función/método.
- Problema con la captura de los eventos del mouse, ya sean click o movimiento. Hacían que se pierda el foco sobre el worm con el que se estaba jugando. Se solucionó no seteando foco a los items, sino asignarles un estado y preguntar por el mismo.
- Problema con las animaciones. Se inició realizando una mala implementación ya que no se debían hacer cálculos de traspolación. Se solucionó enviando por protocolo úni-camente con cambios de posición o estado y con un mensaje por protocolo agregado del estado del worm.

- La rotación de las imágenes para cualquier proyectil se complicó ya que el simple rotar de qt, rota teniendo en cuenta el vértice izquierdo superior como pivote, por lo que traía problemas con respecto a la manera de rotar del juego, ya que se debería rotar sobre el centro de la imagen. Se solucionó básicamente utilizando una transformación de coordenadas que brinda QT llamada Qtransform.
- El mantener teclas. Los eventos de teclado en Qt tienen la particularidad de que si se mantiene apretada una tecla, los eventos de la misma son llamados permanentemente. Se tuvo que implementar un chequeo de si la tecla es presionada repetitivamente para ignorar todos los eventos en el medio que suceden de la misma tecla, y así, capturar únicamente la primer pulsación de la tecla y cuando ésta se suelta.
- Dificultad con la cámara, ya que hubo problemas relacionados al click del mouse que anteriormente se mencionó, y sumado a esto, los clicks en la cámara tenían coordenadas distintas a las que la escena manejaba y había que hacer una conversión.
- Problemas con el movimiento entre ventanas, no se podía lograr un cierre en ciertos casos. La solución fue utilizar una variable de estado.

#### ■ Editor:

- La implementación de los sonidos, ya que requiere la instalación de muchas librerías externas de multimedia. Las mismas fueron:
  - libqt5multimedia5-plugins
  - gstreamer1.0-plugins-base-apps
  - libqt5gstreamer-dev
  - qtmultimedia5-dev
  - qtmultimedia5-examples
- Otro problema del editor sucedió en la primera semana de desarrollo. Al principio se empezó a hacer la aplicación usando la librería SDL y se optó por cambiar a Qt dado que ya tiene implementado todo el sistema de botones, entrada de texto, checkbox, spinbox, escenarios, y otras cosas, ahorrando tiempo de programación.

#### ■ Conjunto:

- El manejo de números negativos y flotantes por protocolo para enviar posiciones entre el servidor y el cliente, ya que el servidor maneja toda la física en flotantes y con valores negativos, mientras que el cliente maneja todos valores positivos y enteros, por lo que se optó por enviar los flotantes como enteros multiplicados por un múltiplo de 10 grande (Ej: 10000), de manera que al llegar al cliente, éste lo divida por el mismo factor, lo multiplique por la escala de píxeles que utiliza y no pierda tanta precisión al dibujar el escenario, a su vez, los valores negativos se envían como positivos más un byte extra indicando el signo.
- Qt: Esta biblioteca trajo varios inconvenientes, entre ellos, los leaks, que ocurren porque Qt hace optimizaciones de buffer entre otras cosas. También, si las aplicaciones hechas con Qt corren con gestores como gnome, que utiliza GTK, primero tiene que levantar la librería gráfica de GTK, que causan leaks y errores de todo tipo porque el sistema hace uso de caches y optimizaciones al cargar librerías gráficas.

## 6. Análisis de puntos pendientes

#### ■ Servidor:

- El recibir datos de los jugadores usando ifs para verificar comandos y luego ejecutarlos debería de ser cambiado por un creador de comandos (Factory method) que cree el comando indicado y lo ejecute si es debido.
  - Evitar de cualquier manera el "cheating" por parte del cliente descartando los paquetes que envíe.
- **Cliente:**
    - Mejorar y agregar más animaciones del juego.
    - Mejorar visualización ventanas de logeo.
    - Mejorar movimiento camara libre con el mouse.
    - Mejorar tipo de explosiones.
    - Agregar la posibilidad de pasar turno.
  - **Conjunto:**
    - Mejorar la clase protocolo para la comunicación entre el cliente y el servidor.

## 7. Herramientas

Las herramientas que se utilizaron se listaran a continuación:

- **Box2D (v2.1.2):** Libreria para manejar las físicas del juego.
- **Qt (v5):** Libreria para manejar la interfaz gráfica del usuario y el editor de mapas.
- **Yaml-cpp (v0.6):** Libreria para leer archivos con extensión yaml.
- **Cmake (v3.1):** Para compilar el proyecto.
- **Latex:** Para escribir las diferentes secciones del proyecto del proyecto.

## 8. Conclusiones

Se darán a continuación, las conclusiones que se obtuvieron a lo largo de la realización del proyecto:

- **Organización del grupo:** Diríamos que esta es la más crítica de todas las conclusiones, ya que una buena organización de las tareas en conjunto, hubiese provisto menos errores y frustraciones por parte del grupo de trabajo.
- **Protocolo de comunicación:** Es recomendable realizar el protocolo de comunicación entre cliente y servidor (y viceversa) antes de ponerse a programar, ya que el mismo sentará las bases de como se va a realizar el código, y su falta (O estado incompleto), determinará ciertos errores que a lo largo del proyecto requieren cambios grandes al mismo.
- **Conexión entre interfaz gráfica y modelo:** Es recomendable conectar la interfaz gráfica del cliente lo antes posible a un modelo de juego, ya que ahí se podrá visualizar y tener una noción de los detalles y valores que se necesitan utilizar en el servidor. Además, nada puede asegurar el buen funcionamiento del servidor si no hay una interfaz gráfica que respalde al mismo (Particularmente con los bugs que son muy difíciles de encontrar si no se pueden visualizar).

- Manejo de tiempos y prioridades: No es recomendable avanzar con una porción particular del proyecto si el resto de los compañeros están teniendo dificultades con ciertas partes del mismo. Ya que aunque uno avance, si el resto no funciona o tiene muchos errores, a la larga requiere más tiempo para depurar (Todo esto sin contar los posibles errores que pueden haber en el código sobre el que se avanzó sin la prueba de conjunto).
- Reuniones: Es muy recomendable reunirse con el grupo de trabajo varias veces a la semana y programar en conjunto (O en su defecto, programar vía cualquier aplicación que permita comunicación de voz a través de internet, por ejemplo, Skype), ya que esto permite ver como van los demás compañeros de trabajo, así como también de analizar que puntos se deben de modificar con mayor prioridad. Aplicado a nuestro caso, mejoró sustancialmente el avance del proyecto en mucho menos tiempo que si se hubiese hecho en separado.
- Utilizar un repositorio: Utilizar un repositorio (Ej: Github) permite ver como va fluyendo el avance del proyecto, así como también permite que todos puedan tener acceso al código que cada uno va programando. Además, permite ubicar tareas en el mismo con cierto nivel de prioridad para que cuando uno entre al repositorio, vea que cosas son necesarias modificar antes que otras.
- Organización: Mantener un orden de los archivos y carpetas que actualmente utiliza el proyecto, si hay algo que ya no se utiliza, mejor sacarlo lo antes posible del repositorio o donde se encuentre el proyecto.