# MEMORIA DE RISK

Evolución y Adaptación del Software

Universidad Rey Juan Carlos

2022

GRUPO 22E05

Jesús González Gironda

Guillermo Grande Santi

Alejandro López Adrados

Gonzalo Ortega Carpintero

# CONTENIDO

# 1. INTRODUCCIÓN

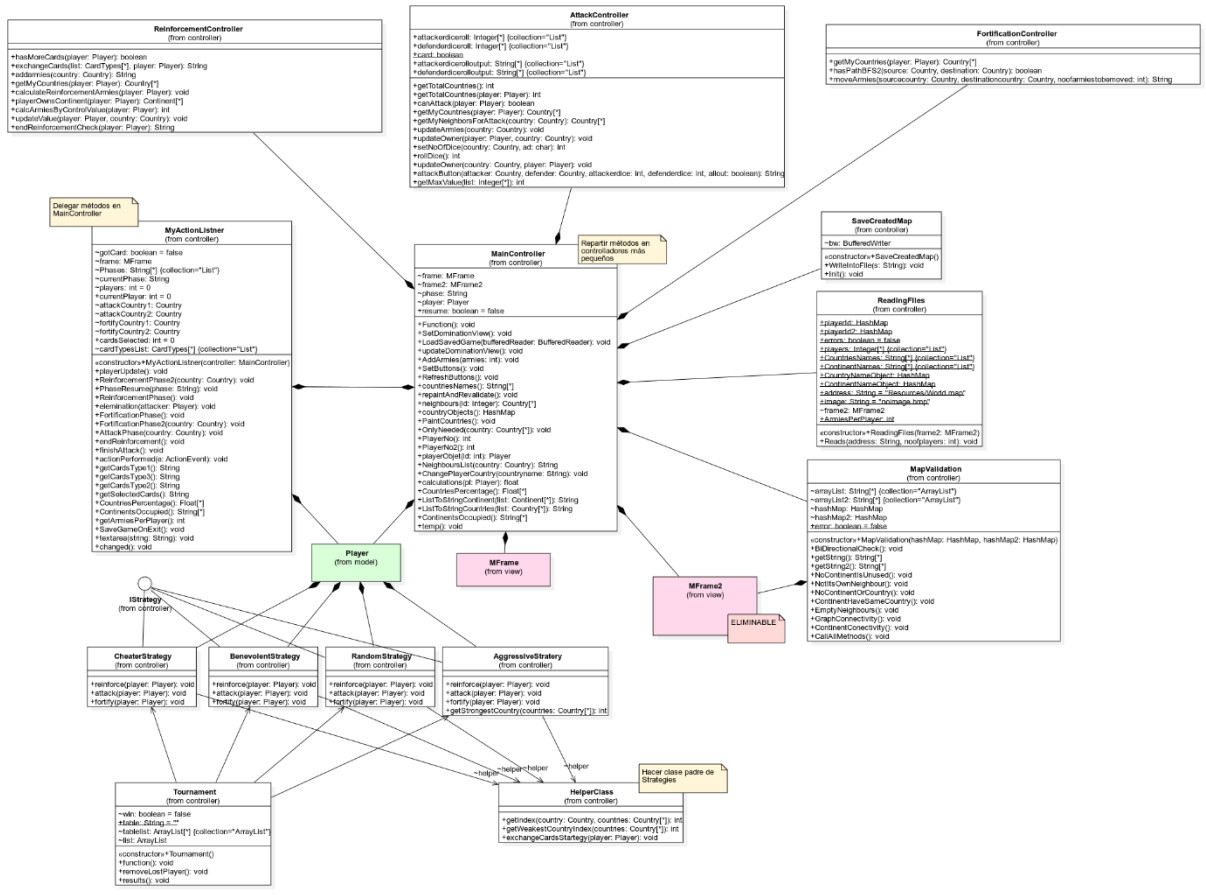
El objetivo de esta práctica era mantener y hacer accesible un viejo repositorio con un proyecto de Java que implementaba una versión digital del juego de mesa Risk. El desarrollo de la práctica se ha dividido en tres fases, una relativa al mantenimiento, otra de diseño y planificación de mejoras y otra de implementación de dichas mejoras.

El mantenimiento del código es una fase de suma importancia debido a que, además de implementar pequeñas mejoras de implementación y arreglar diversos fallos, mejora en gran cantidad el entendimiento del código y facilita su posterior modificación.

Una vez realizado el mantenimiento, se pasa a la fase de evolución, siendo necesario un diseño previo de los cambios a realizar, entre los que destaca un diagrama UML de clases para planificar los cambios a realizar y la nueva estructura del proyecto. Finalmente, en la última fase de la práctica, se implementan todos los cambios y mejoras propuestos anteriormente.

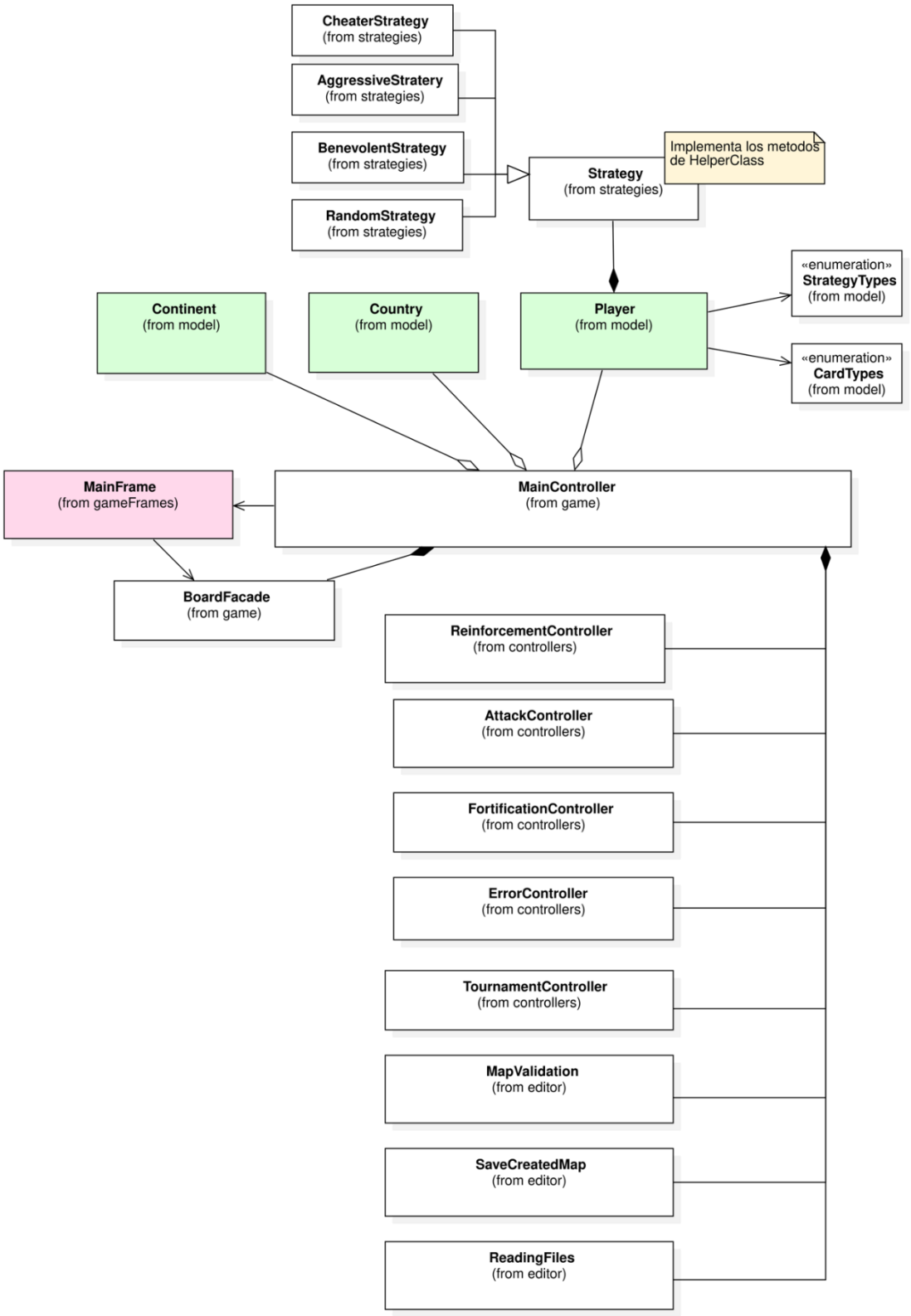
# 2. MANTENIMIENTO

Durante la fase de mantenimiento, una gran parte del tiempo empleado ha sido destinado a la **comprensión del código**. Para ello, hemos elaborado un diagrama UML de clases representando la estructura del proyecto original entes de modificar nada.




En este diagrama destaca principalmente que la mayoría de las clases están interrelacionadas, lo que rompe uno de los principios de **encapsulación** y de **abierto cerrado**. Para arreglar dichos problemas, decidimos pensar en otro diseño conservando las clases y métodos del programa antiguo, pero

cambiando las llamadas entre clases. Para realizar esto, es necesario tener cuidado ya que se pueden dejar atrás ciertas funcionalidades. Es por ello por lo que decidimos diseñar otro diagrama UML para así conseguir una mejor organización.



Como se puede observar, obtenemos un diseño más limpio del programa en el que todas las clases se conectan con la clase *MainController* y ésta es la encargada de realizar el resto de las conexiones.

Tras ello, y una vez conseguido tener un programa más ordenado y comprensible decidimos **reducir la extensión de las clases** para hacerlas más manejables y a su vez más fácilmente modificables. Para esta labor, destacan principalmente las clases *MyActionListener y MainController* las cuales contaban con un gran número de líneas, por lo que hemos creado pequeños controladores para llevar a cabo la lógica de las fases de juego por separado (*ReinforcementController, AttackController, FortificationController*). Esto ayuda en la legibilidad del código y reduce considerablemente el número de líneas de la clase principal de nuestro programa (*MainController*).

Respecto a la parte de vista, no hemos invertido demasiado tiempo en su mantenimiento ya que desde un primer momento decidimos rehacerla de nuevo con una interfaz más moderna e intuitiva, ejemplo de esto es poder mostrar un mapa cuando se juega y poder clicar sobre países en lugar que en una cuadrícula.

Tras terminar toda la fase de organización de código descrita anteriormente, el siguiente objetivo consiste en la **adición de documentación** comentando los métodos incluyendo su objetivo, los parámetros necesarios y el valor que devuelve si éste es necesario. Esto aporta una mayor comprensión de todas las clases para cualquier programador en un vistazo.

Antes de dar por concluida la fase de mantenimiento, hemos incluido **Gradle** en nuestro proyecto para poder compilar el código independientemente del IDE en el que se esté trabajando, además de añadir compatibilidad para futuras modificaciones a largo plazo sin tener que cambiar nada. Además, proporciona una mayor facilidad para importar diferentes librerías como se comentará en la fase de evolución.

En líneas generales, en esta fase hemos realizado cambios en el código para hacerlo más comprensible y fácil de modificar. Es por ello, que la apariencia y funcionamiento del juego tras completar esta fase es idéntica a su funcionamiento general, dando por sentado que su funcionamiento teórico es correcto.

# 3. PROPUESTAS DE EVOLUCIÓN

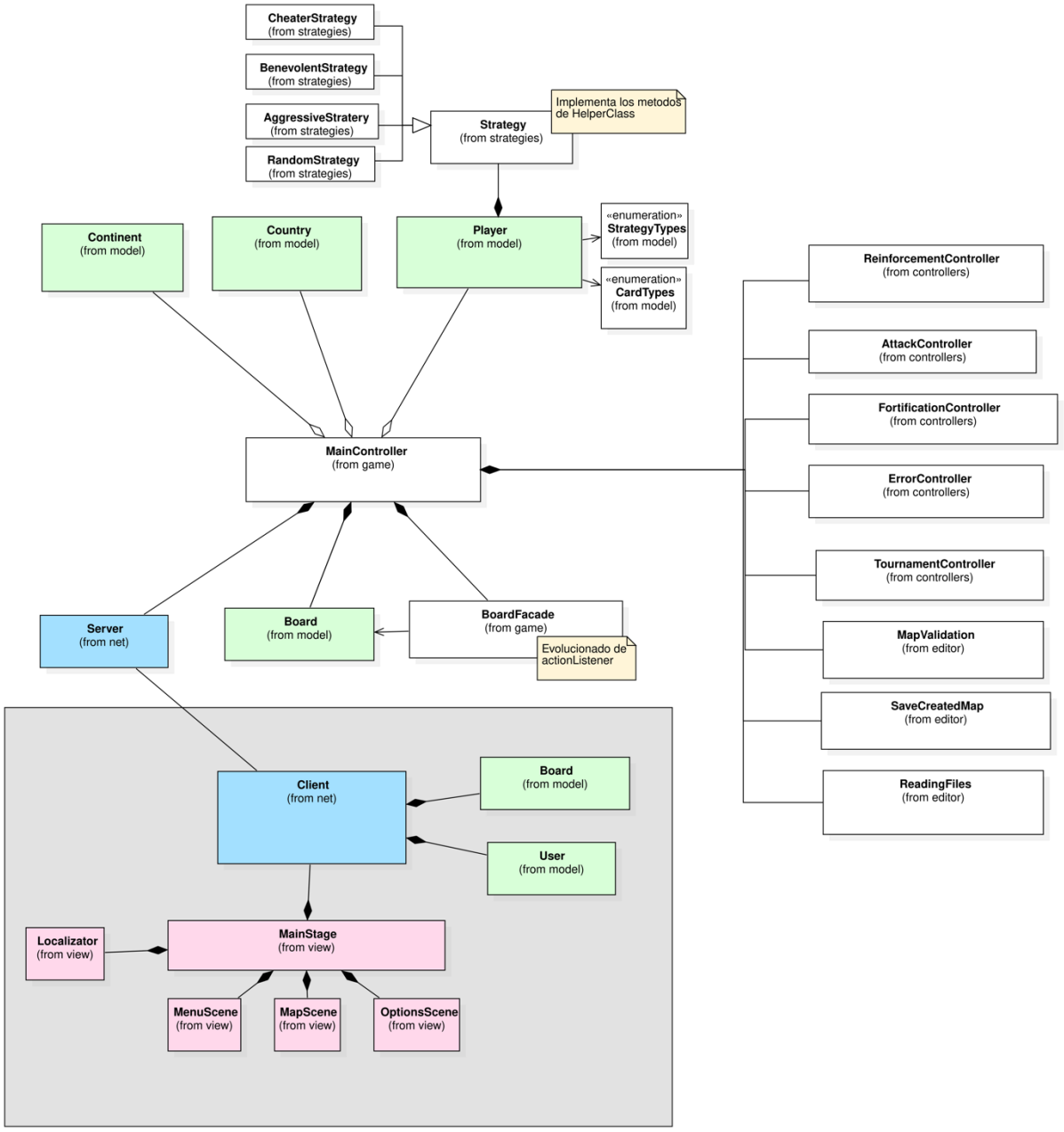
Una vez finalizada la fase de mantenimiento del juego propusimos tres ideas para mejorarlo, las cuales ya detallamos más en detalle en el documento de propuestas de evolución entregado anteriormente. Principalmente, estas ideas se dividían en tres.

Por un lado, y de cara a conseguir una experiencia de usuario más agradable y que permitiera hacer más disfrutable el juego, decidimos cambiar completamente la interfaz de usuario implementando menús más intuitivos y una pantalla de juego en la que se pudiera ver el mapa de juego de forma gráfica, y no únicamente en forma de cuadrícula como en el proyecto original.

Para hacer el juego accesible a más gente, decidimos también implementar un sistema de localización que permitiese jugar al juego en distintos idiomas permitiendo así entender el juego a una mayor base de jugadores.

Por último, y como propuesta estrella, decidimos dar la opción de realizar partidas en línea para poder jugar partidas de forma remota con otros jugadores.

Para planificar todos estos añadidos diseñamos también, junto con el documento de propuestas, el siguiente diagrama de clases mostrando la nueva estructura que tomaría nuestro proyecto.



# 4. EVOLUCIÓN

Para el proceso de evolución hemos dividido el trabajo a realizar en tres bloques, separando las principales facetas de evolución que nos propusimos realizar: la implementación del juego online, el desarrollo de la nueva interfaz, y la implementación del sistema de traducción.

## 4.1. Juego en línea

El juego consta de una implementación básica de cliente/servidor rudimentarios. Para simplificar la complejidad del proyecto se ha optado por la librería de serialización más sencilla de Java, *Serializable*. Dicha decisión simplifica el protocolo un par de paquetes. Primero se envía desde el cliente una petición de saludo encerrada en el paquete *ClientInfo* que valida la información del usuario.

Una vez registrado un usuario en un servidor este envía paquetes de la clase *ClientUpdate* con contenido sobre la actualización de usuario. Las peticiones se organizan siguiendo un patrón comando donde la clase principal *MyActionListener* despeja la cola de peticiones y ejecuta las actualizaciones en el servidor.

Por otro lado, el servidor envía una *ServerUpdate* a todos sus clientes cada vez que se produce un cambio en el estado del tablero virtual. Dicho tablero contiene información sobre las tropas, países, jugadores, mensajes de error, así como valores misceláneos.

La ejecución de ambos programas se realizará en paralelo con la interfaz principal de modo que una petición de escucha no bloque el hilo principal. Así, en caso de jugar una partida local se necesitarán dos hilos de cliente y servidor, así como todos los necesarios para la interfaz. La sincronización se lleva a cabo a través de funciones atómicas *synchroniced* así como de llamadas a las funcionalidades *doLater* de Java FX.

## 4.2. Mejora de la interfaz

Para implementar la nueva interfaz hemos optado por el uso de *JavaFX*, con el cual es sencillo crear diferentes vistas de la aplicación y enlazar métodos a las diferentes acciones que puede realizar el usuario mediante el uso de ficheros FXML. Estos son ficheros de texto que se pueden editar gráficamente mediante el propio editor de *InteliJ* o el editor externo *Scene Builider*.

Hemos creado seis FXML para implementar las vistas necesarias para el funcionamiento del juego, y los hemos enlazado con diversas clases controladoras de la interfaz. La nueva clase principal del programa es *MainUI*, que es la encargada de crear un contenedor *GameController* que será el encargado ejecutar el programa y almacenar toda la información necesaria. Luego, crea una nueva escena en la que cargar los FXML y se muestra *start-view.fxml*. Dicho formulario está asociado al controlador *StartViewController*, quien también se encarga de gestionar los métodos de las vistas de ajustes y créditos.

La pantalla principal del juego contiene botones para salir del juego, ir a la ventana de ajustes o comenzar una partida. En caso de que el usuario quiera comenzar a jugar se le preguntara si quiere jugar de forma local u online, preguntando de ser así, si se quiere crear una partida o unirse a una ya creada. Estas preguntas se muestran al usuario mediante paneles que se muestran y ocultan, recopilando toda la información necesaria para el arranque del juego.

Al hacer *click* en el botón de ajustes se carga el formulario *settings-view.fxml* el cual contiene botones para cambiar el idioma del juego, un enlace a las instrucciones del mismo y un interruptor para la música. Desde la ventana de ajustes también se puede acceder a la vista de créditos, *credits-view.fxml* y tanto la ventana de ajustes como la de créditos cuentan con un botón para volver al menú principal.

Una vez el usuario decide crear una partida, se carga el formulario *customice-view.fxml*, que contiene diferentes cajones de selección para introducir los jugadores que participaran en la partida. En caso de ser una partida local, solo se deberán seleccionar jugadores no humanos, y en cualquiera de los casos, se podrán seleccionar hasta seis jugadores controlados por la consola, que jugarán siguiendo las estrategias implementadas en el proyecto original. También, se permite seleccionar uno de los dos mapas que hemos implementado. Una vez se decide empezar, se muestran los posibles errores que puedan surgir, como no haber seleccionado suficientes jugadores o no haber seleccionado un mapa, y si todo está correcto, se carga el formulario *map-view.fxml* que contiene la pantalla de juego principal, gestionada por *MapViewController*.

Para implementar la selección de países hemos recurrido a mostrar por debajo del mapa que ve jugador, un mapa de colores en el cual, al clicar sobre él, se puede determinar qué país se está seleccionando. Por encima de esto, se muestran las imágenes de los países del mapa correspondiente, cargadas dinámicamente junto con su nombre, y se colorean del color correspondiente al jugador que posea cada país. Además del visor de mapa, esta vista tiene un panel lateral en el que se muestra información relevante de la partida, como el jugador actual, la fase que se está jugando, o el número de tropas que faltan por desplegar. Debajo de estos *labels* con información, hemos colocado tres botones. El primero de ellos manda la señal al servidor para cambiar de fase, la segunda muestra un panel con las cartas del jugador actual, y el último permite salir de la partida.

Todas las ventanas auxiliares usadas para mostrar o pedir información de cualquier tipo están implementadas del mismo modo, mostrando un panel translucido con una franja con los campos necesarios, el cual se oculta al clicar fuera de la franja.

La información que se muestra por pantalla se actualiza continuamente mediante el método *update*, que se comunica con el servidor y realiza las acciones necesarias en función de la fase y el *input* que reciba del usuario. En la fase de refuerzo se puede hacer clic en los diferentes países para añadir tropas sobre ellos, en la fase de ataque se pueden seleccionar dos países para hacer aparecer un panel con el que seleccionar el número de tropas con las que se desea atacar y en la fase de refuerzo se pueden seleccionar dos países para hacer aparecer una ventana con la que elegir las tropas que se quieren desplazar de uno a otro.

Para cualquier tipo de entrada errónea, como intentar desplegar tropas en un país del cual no se es propietario, o intentar mover tropas entre dos países no conectados, se muestran paneles mostrando los errores que en su día lanzaba el proyecto original.

Cuando todos los jugadores son eliminados, se muestra la pantalla de final de juego, asociada al formulario *results-view.fxml*, la cual sería la encargada de mostrar la información relevante de clasificaciones u estadísticas, algo que de momento no está implementado.

### 4.3. Localización

Con el propósito de que nuestro juego llegue a más gente hemos implementado la interfaz en dos idiomas, tanto español como inglés. El proceso de internacionalización del proyecto lo hemos realizado mediante el uso de archivos de *bundles* donde se encuentran la correspondencia entre etiquetas y *strings* en ambos idiomas, las etiquetas en su mayoría se encuentran usadas en los *fxml* y en algún panel de aviso emergente. El constructor de *fxml* permite el paso de un *bundle* lo que hace que automáticamente se traduzca sin complicar el código más allá de un parámetro en dicho constructor. Por otra parte, el uso de la internacionalización ha dado lugar a algún que otro inesperado fallo difícil de seguir, por ejemplo, en español el juego falla en ciertas cosas, no muestra el primer panel emergente al iniciar el juego, en la primera ronda no permite reforzarse pues carga 0 refuerzos,

permite atacar y conquistar, sin embargo, al terminar el turno completo no cambia de jugador. Un fallo bastante inesperado que el cambio de *bundle* desencadene fallos que aparentemente no sean prácticamente dependientes.

### 4.4. Música

Finalmente, para una experiencia más inmersiva y emocionante hemos añadido música al juego. La implementación de la música ha sido mediante el uso de un *stream* de audio y un archivo de música libre almacenado en la carpeta de *resources*. Para fomentar el libre albedrío de los jugadores, la música se puede quitar para que no moleste a un jugador si lo desee.

## 5. GUÍA DE USO

La primera pantalla que aparece al ejecutar el juego contiene tres botones (Ajustes, Jugar y Salir). Una vez se selecciona el botón Jugar, el programa preguntará si se desea jugar en modo local o en modo online.


### 5.1. Instrucciones para el modo individual

En este modo se procederá a la selección de jugadores participantes y al tipo de jugador si va a ser humano (Uno solo o ninguno) o CPU (La cual cuenta con diferentes estrategias de juego). Además, cuenta con la opción de ponerse nombre personalizados (Si se deja en blanco serán nombres estándar). Se puede utilizar de dos modos diferentes:

Para ejecutar el antiguo modo torneo basta con asignar a todos los jugadores diferentes estrategias de CPU sin que ningún humano intervenga.

Por otra parte, si se quiere iniciar una partida contra diferentes CPU por parte del jugador basta con asignarse un color y los contrincantes elegidos.

### 5.2. Instrucciones para el modo multijugador

Si se selecciona esta opción se dará la opción de ser el anfitrión de una partida o unirse a una partida. **IMPORTANTE**: Para poder jugar **varios humanos en un mismo ordenador** el programa ha de abrirse tantas veces como jugadores locales haya.

Siendo el **anfitrión** de la partida, deberás configurar el número de jugadores con su respectivo nombre cada uno y posteriormente comenzar el juego. **IMPORTANTE:** El nombre de jugador que pone el anfitrión debe ser el mismo que el que escribe el jugador que se une. También es muy importante que el anfitrión comience el juego y espere hasta que todos se hayan unido (en la pantalla del mapa), pues es en ese instante cuando podrá comenzar la partida. Con esto se quiere decir que los jugadores no podrán unirse cuando todavía se está configurando el juego (en la pantalla de configurar partida). Además, el anfitrión debe tener el puerto de red 57565 abierto.


Para **unirse** a una partida de forma **online** se debe introducir la dirección IP del anfitrión (recordamos que el anfitrión debe tener el puerto de red 57565 abierto). Tras esto introducirá el mismo nombre que el anfitrión haya asignado ese jugador y comenzará la partida.

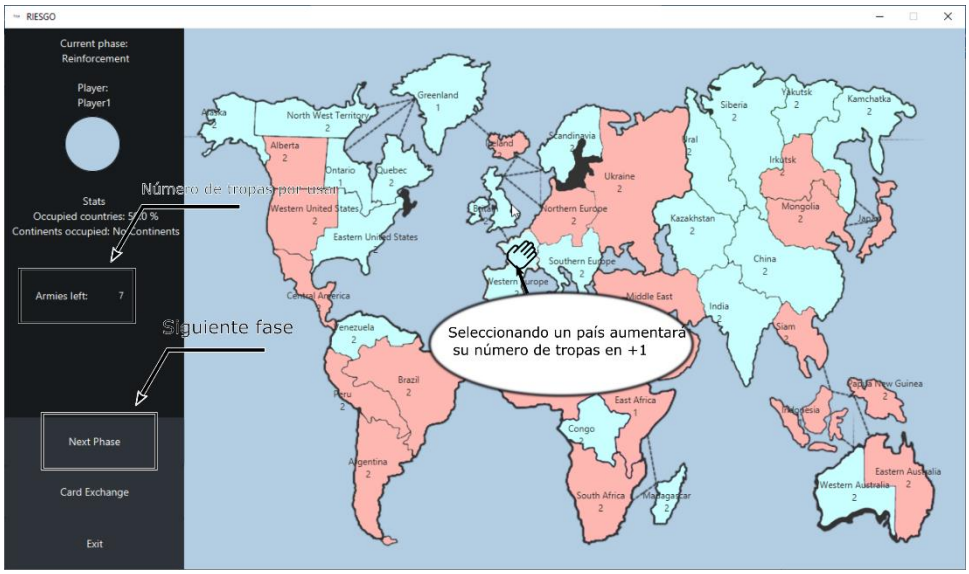
Para **unirse** de forma **local** basta con rellenar en la dirección IP "localhost" y rellenar el nombre de igual manera que en el modo online.

## 5.3. Instrucciones para jugar una partida

Cada turno se divide en tres fases siguiendo las normas del Risk tradicionales.

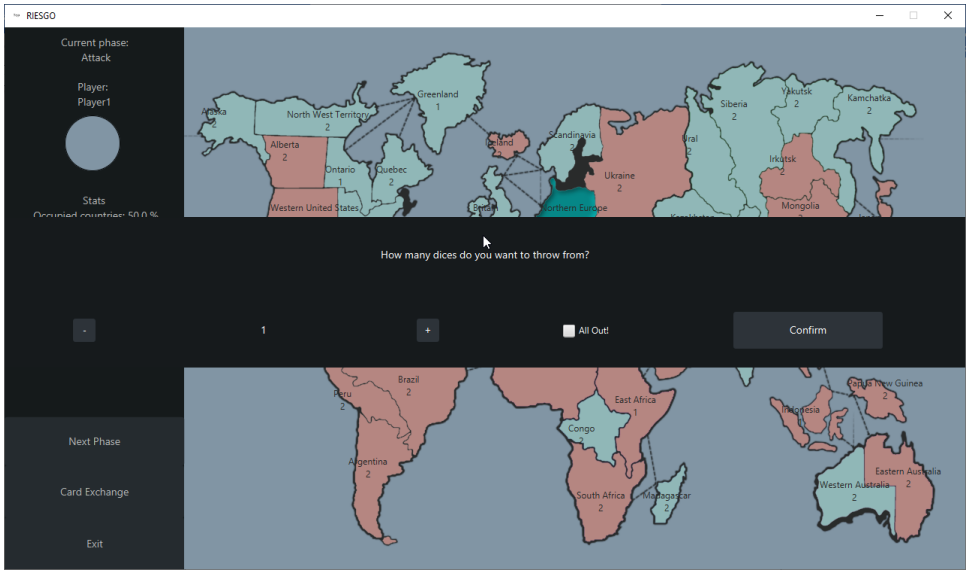
### Fase de refuerzo

En esta fase se permite seleccionar países que pertenezcan al usuario para aumentar en 1 el número de tropas del país. La fase acaba cuando no queden tropas por repartir.



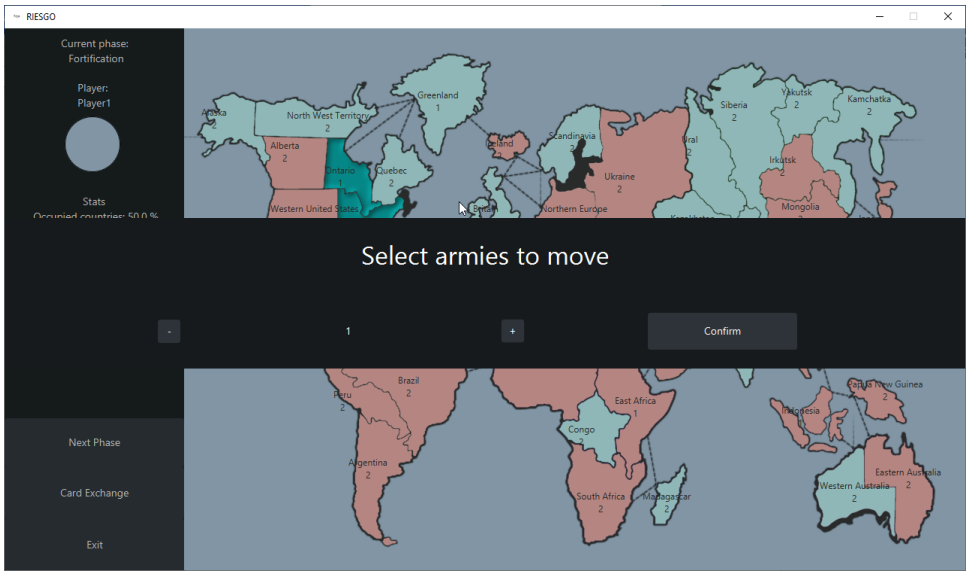
## Fase de ataque

En esta fase se permitirá atacar a países vecinos. Para atacar a un país vecino basta con seleccionar con que país se va a atacar y el país al que se va a atacar. Una vez seleccionados se elige el número de dados con los que atacar.

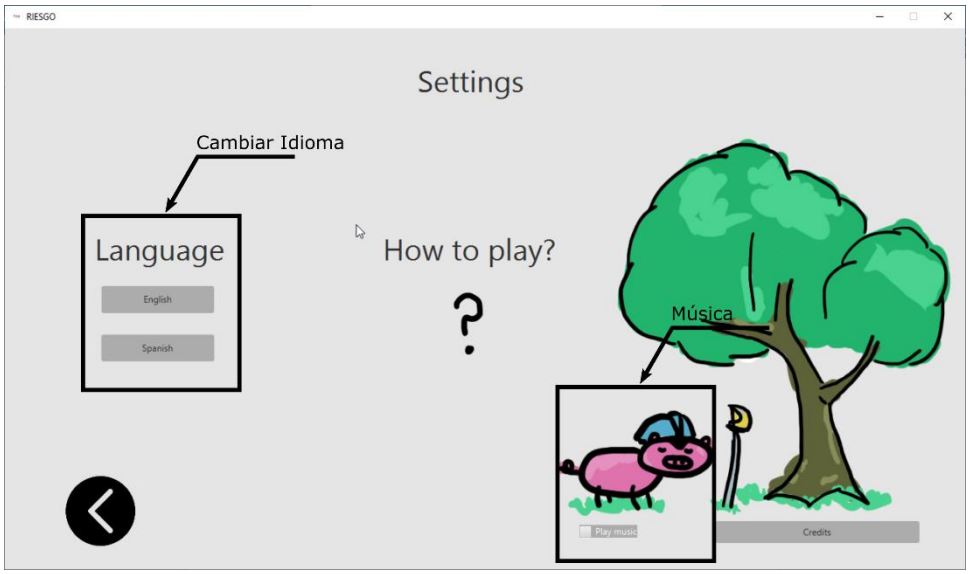
### Fase de fortificación

Por último, en la fase de fortificación se permite el movimiento de tropas entre territorios controlados con un camino entre ellos.




## 5.4. Ajustes

En la pantalla de ajustes se permite la configuración del lenguaje, así como el acceso a la pantalla de créditos, la activación del sonido y enlace a más información sobre las reglas del juego.



# 5 FALLOS Y POSIBLES MEJORAS

Tras terminar nuestra evolución del programa, con demasiado poco tiempo restante para la entrega, nos hemos dado cuenta de varios fallos y mejoras que teníamos planteado resolver pero que no ha sido posible. En general, nos hemos dado cuenta de hicimos más hincapié en la evolución que en el mantenimiento inicial, suponiendo que el primer programa estaba mal hecho en el sentido de dificultad de comprensión, estructuras mal utilizadas, variables públicas, etc., pero que contaba con un correcto funcionamiento del juego en sí. Sin embargo, nos hemos dado cuenta de forma tardía de que también hay varias funciones que están mal implementadas.

Esto ha causado que, durante el desarrollo de la evolución, encontráramos algunas funciones que pensábamos que eran correctas cuando en realidad no era así. Muchas de ellas se han quedado sin arreglar debido al que no habíamos planificado el tiempo que nos hubiera costado comprender completamente el funcionamiento de dichas funciones para poder así arreglarlas o, en el peor de los casos, implementarlas de nuevo.

A veces, en la fase de fortificación, la función *hasPathBFS2* de encontrar un camino entre países que se pueden fortificar a veces falla, y las estrategias implementadas originalmente no hacen nada relevante en una partida, excepto el *Cheater*, cuyo nivel es exagerado. El fallo que hemos encontrado es que fallan al calcular refuerzos y nos habría gustado implementar unas estrategias que de verdad se sintiesen jugadores distintos con sus respectivos niveles, algo que hubiera significado una inversión considerable de horas de trabajo.

En determinadas circunstancias el número de dados que se muestra al ser lanzados es incorrecto ese a que el funcionamiento interno sea correcto, es decir, aunque se tiren tres dados y se muestre únicamente uno, la función de atacar se realizó con los tres dados. El problema reside en una función del programa inicial que devuelve el número de dados erróneamente después de utilizarlos.

Por falta de tiempo, al acabar una partida se muestra una última pantalla en la que pone "*Winner*". Como mejora, nos gustaría que pusiese más información sobre la partida como el jugador que ha ganado (pero habría que añadir funciones en el servidor para mostrar un mensaje distinto a cada jugador).

Algo que sí que hemos implementado en un último momento son las cartas de tropas, y un panel que permite al usuario seleccionar cartas para intercambiarlas por más refuerzos. Sin embargo, no ha dado tiempo a implementar un control de errores extendido con esta función, es decir, si el usuario usa las cartas como se deben de usar, el programa funciona. En caso contrario, no se han hecho pruebas.

# 6 CONCLUSIONES

La realización de esta práctica nos ha servido para comprender la importancia de un buen mantenimiento del software, que hace que la posterior comprensión o modificación del mismo sea mucho más sencilla y requiera de menos tiempo y esfuerzo. También nos hemos dado cuenta de lo importante que es contar con una buena documentación del proyecto para así ser capaces de visualizar rápidamente los objetivos de cada módulo del proyecto, y la intención que tenía su programador original. Hemos aprendido, además, a planificar mejor un proyecto de una envergadura considerable, aunque haya tenido que ser viéndonos obligados a renunciar finalizar ciertas tareas por falta de tiempo y el acercamiento a la fecha de entrega.