Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Hyperchess

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: José Manuel González Peñalva

Tutor: Ramón Pascual Mollá Vayá

Curso 2020-2021

Resumen

El trabajo trata de la creación de un videojuego llamado “HyperChess”, donde se permite al usuario poder crear y jugar a sus propias variantes del ajedrez, pudiendo diseñar piezas y tableros e introducirlos en el juego mediante el uso de archivos XML. Todo esto se implementa usando el motor gráfico Unity.

Palabras clave: videojuego, ajedrez, configurable, modular, Unity, XML.

Abstract

The assignment consists of the creation of a videogame by the name of “HyperChess”, where the user is able to create and play their own chess variants, being able to design their own pieces and tables, and introduce them in the game with the use of XML files. All this is implemented using the graphic engine Unity.

Keywords : videogame, chess, configurable, modular, Unity, XML.

Tabla de contenidos

1. Introducción

Todos los juegos han sufrido algún tipo de modificación a medida que se han ido extendiendo por el mundo y ha pasado el tiempo, y el ajedrez no es una excepción. Los primeros indicios de juegos del estilo del ajedrez se encuentran en el siglo VI en la parte norte de la península india, basado en las 4 divisiones de tropas que usaban en la época: la caballería a carro, la caballería estándar, los elefantes de guerra y la infantería.[1] Este juego se fue extendiendo por el norte africano hasta llegar a Europa sobre el siglo X, donde durante los próximos cinco siglos fue evolucionando hasta el ajedrez que conocemos hoy en día, con cambios notables como el cambio del elefante por el alfil (1). Aunque la primera edición del reglamento del ajedrez fue escrito sobre el siglo XVIII, eso no detuvo a la gente de seguir modificando el ajedrez de muchas formas, desde modificaciones a la posición inicial del tablero, como en el Ajedrez 960[2] del campeón mundial Bobby Fischer donde la posición inicial se determina de manera aleatoria, pasando por la creación de nuevas fichas y modificación de las dimensiones del tablero, como en el Ajedrez Capablanca[3] el cual incorporaba dos fichas nuevas que resultaban de la combinación de fichas ya existentes en un tablero más ancho para acogerlas, hasta modificar la misma base del tablero, como las diferentes variantes de ajedrez hexagonal que se han desarrollado durante el tiempo. El problema que esto conlleva es que hay que manufacturar tanto las nuevas piezas como los tableros para poder jugar, cosa que no suele convencer a la gente comprar un nuevo set de piezas y tablero para cada variante.

Con la llegada de la informática y, con ella, los videojuegos, se facilitó mucho más la modificación de los juegos y el poder compartirlos con todo el mundo. Juegos como Doom 2 (1994) o The Elder Scrolls V: Skyrim (2012) son algunos de los más conocidos por la gran cantidad de modificaciones que los jugadores han creado para estos, y que cada vez es más común gracias a herramientas oficiales para modificar los juegos creadas por los desarrolladores (2) y herramientas como la “workshop”, una plataforma para la tienda de videojuegos Steam para publicar modificaciones de estos, donde con un simple clic se instalan solos. Estos tipos de modificaciones no solo han alargado la vida de muchos de estos juegos, si no que en algunos casos ha llegado a dar trabajo a algunos de los creadores de estos, o incluso convertirse en algunos de los juegos más importantes de la industria, siendo los ejemplos más remarcados el Counter-Strike: Global Offensive (2012), salido de la modificación del juego Half Life (1998) llamada Counter-Strike y comparada por la misma desarrolladora del juego que modificaron, o Dota 2 (2013), salida de la modificación del juego Warcraft 3 (2002) llamada Defense of the Ancients.

En la actualidad, aunque se ha popularizado mucho el juego del ajedrez en línea y algunas de las webs más usadas incluyan la posibilidad de jugar algunas variantes de este, como el Ajedrez 960 mencionado anteriormente, no permiten crear tus propias piezas o modificar el tablero de ninguna manera, como mucho la posición inicial. Poder ofrecer algún tipo de herramienta para poder crear y experimentar con este tipo de

[1] https://www.chessvariants.com/historic.dir/chaturanga.html
[2] https://www.chessvariants.com/diffsetup.dir/fischer.html
[3] https://www.chessvariants.com/large.dir/capablanca.html

modificaciones puede fomentar la creación de nuevas variantes del ajedrez, y esto es lo que se quiere cubrir con este trabajo.

1.1. Motivación

Los diferentes motivos para la realización de este trabajo son los siguientes:

- Desde siempre ha sido un objetivo personal el poder incorporarse a la industria de los videojuegos profesionalmente, y un trabajo como este puede ser un buen primer paso y toma de contacto con el proceso que conlleva el desarrollo de un videojuego, especialmente uno tan modular como puede llegar a ser este.

- Además, aunque esta industria sea la que más se diferencia respecto a otros sectores de la informática respecto a cómo se desarrolla, sigue asemejándose mucho en metodología, así que obtener este tipo de experiencia puede extrapolarse a otros sectores de manera cómoda.

- La industria del videojuego se ha convertido en los últimos años en una de las industrias que más dinero genera alrededor del mundo, superando ya a algunas de las más asentadas como son las de la música o el cine.

- Los juegos de estrategia siempre han sido uno de mis géneros favoritos, y el ajedrez siempre ha despertado curiosidad en mí. Poder explorar todo tipo de variantes de este y poder experimentar las distintas versiones que se han creado para este de manera mucho más cómoda con una aplicación que lo soporte es algo que mucha gente como yo agradecería.

- El motor de juegos Unity es uno de los más utilizados y asentados dentro de la industria del videojuego. Poder trabajar y familiarizarse con este programa puede ser una gran experiencia para poder incluirse a la industria con mayor facilidad.

1.2. Objetivos

El objetivo de este trabajo es el crear un videojuego basado en el ajedrez utilizando el motor de videojuegos Unity como base para este, utilizando las librerías que este motor ofrece para leer archivos XML para cargar las diferentes variantes del ajedrez desde estos.

Crear un formato para los archivos XML que sea sencillo de entender y modificar y a la vez suficientemente completo como para poder incluir gran variedad de tipos de movimiento distintos.

Los tableros que carguen deben ser funcionales, ofreciendo un sistema de turnos alternos para que varios jugadores puedan jugar una partida completa sin problemas, haciendo que las fichas se muevan y capturen como está escrito en los archivos XML

que los describen e incluyendo un sistema que evite que un jugador pueda realizar un movimiento ilegal.

1.3. Metodología

En primer lugar, se seleccionarán las herramientas para el desarrollo de la aplicación entre todas las opciones disponibles y se instalarán para su uso. Se hablarán de ellas más adelante en el apartado 4.1 con más detalle.

Después, se realizará un diseño general de la aplicación, primero eligiendo el alcance que tendrá esta respecto a los formatos de tableros que se soportarán, seguido del diseño de los archivos XML que se utilizarán para estos.

A partir de aquí, se utilizará el desarrollo del videojuego con el método ágil, desglosándolo en distintos subsistemas, identificando las dependencias entre ellos y ordenándolos por prioridad. Esta lista de subsistemas puede cambiar posteriormente por distintas razones.

Cada uno de estos subsistemas se implementarán siguiendo el mismo proceso:

- Primero se identificarán los requisitos del subsistema.
- Después se hará un diseño de su lógica, seguido, si fuera necesario, de un diseño de la interfaz que lo acompaña.
- Se sigue con la implementación de dicha lógica e interfaz, si tuviese. En caso de que sea necesaria una interfaz, los contenidos de autor utilizados se obtendrán de librerías de uso libre o, en caso de que no exista o que sea un elemento muy sencillo, se crearán unos propios.
- Una vez implementado, se procederá a realizar una serie de pruebas para confirmar que no existe ningún error aparente.

1.4. Estructura de la obra

Esta memoria se divide en un total de diez apartados. Para una fácil localización de todos estos apartados, se procede a explicar ligeramente que se trata en cada en cada apartado.

El primer apartado es la introducción, donde se pretende presentar el trabajo de manera general, explicar la finalidad del trabajo y ofrecer una explicación de que se puede encontrar dentro de esta memoria.

El segundo apartado es el estado del arte. Aquí se pretende poner en contexto al lector de dos conceptos principales. El primer concepto son los diferentes avances que ha habido en la industria de los videojuegos, tanto respecto a la facilidad de crear videojuegos gracias a la creación de motores gráficos que ofrecen al usuario una base desde la que crear sus propios videojuegos y con una serie de herramientas para facilitar tanto el desarrollo como la prueba de estos videojuegos, como respecto a las modificaciones de fans de estos juegos que, entre otras cosas, alargan la vida de un

videojuego. El segundo concepto es el del diseño y desarrollo de variantes del ajedrez, presentando los diferentes intentos no solo de creación de variables de por sí, sino también de, junto a la llegada de la informática, los diferentes intentos de facilitar esta creación de modificaciones del ajedrez.

El tercer apartado es el análisis del problema, donde exponemos los requisitos que tiene el desarrollo del videojuego, tanto del videojuego en sí con las funcionalidades que debe ofrecer, como de los archivos XML que el usuario usa para introducir sus variantes en el videojuego.

El cuarto apartado es el diseño de la solución, utilizando la información obtenida en el apartado anterior para plantear diferentes soluciones y arquitecturas que se pueden implementar para las diferentes necesidades encontradas, comparándolas entre ellas y eligiendo que solución es la más adecuada.

El quinto apartado es la implementación de la solución. Aquí detallamos de manera más concreta como se ha implementado todo el diseño realizado en el apartado anterior y donde se adjuntan algunas imágenes desde dentro del videojuego donde se observan estos sistemas en acción.

El sexto apartado son los resultados obtenidos, donde compararemos el alcance que ha tenido el proyecto finalizado con el que se pretendía obtener en un principio.

El séptimo apartado son las conclusiones, analizando como ha sido el desarrollo del proyecto, el grado de satisfacción con este, lo aprendido con este trabajo y la relación de este con los estudios cursados.

El octavo apartado son los trabajos futuros. Aquí se ofrece una serie de sugerencias para una posible ampliación del proyecto, tanto con objetivos no alcanzados en este desarrollo, como diferentes ideas que surgieron al principio del desarrollo pero que se dejaron fuera por falta de tiempo, como de ideas surgidas una vez se terminó el proyecto.

En el noveno apartado se listan las referencias bibliográficas usadas para algunos apartados del trabajo.

Por último, el décimo apartado se trata de un anexo donde se describe con detalle el funcionamiento de la aplicación desarrollada, específicamente el dónde se deben introducir los archivos para ser leídos por el juego y la sintaxis de los ficheros XML utilizada para indicar al sistema cómo funcionan las diferentes piezas y tableros creados.

1.5. Convenciones

Para listar la bibliografía, se ha utilizado la convención ISO 690 - Referencia numérica.

Para mencionar la bibliografía, se ha marcado entre paréntesis el número del libro referenciado.

En el anexo, para marcar el nombre de elementos del archivo XML se han introducido estos nombres entre los símbolos de mayor qué y menor qué.

En el anexo, para marcar el nombre de atributos del archivo XML se han introducido estos nombres entre comillas.

2. Estado del arte

Para entender el problema, primero debemos analizar la historia y la situación actual tanto del ajedrez como de la industria de los videojuegos.

2.1. Tecnologías

Antiguamente, los videojuegos se tenían que programar desde cero, tanto la lógica de estos como la manera en la que estos se mostraban en pantalla, utilizando las funciones ofrecidas por la consola donde estabas desarrollando. Esto conlleva muchos problemas, el primero siendo que cada consola funciona internamente de manera distinta, haciendo que los juegos tuviesen que ser rehechos casi enteros para poder compilarlo en una nueva plataforma (2). Otro de los problemas es el tiempo de desarrollo, ya que se pierde mucho tiempo creando la parte de presentación del juego, y crear un código único para el renderizado de juegos en una consola no era viable en aquella época porque ese código era muy probable que no se pudiese reutilizar de ninguna manera con las consolas de la siguiente generación. Aun así, se crearon algunas herramientas para la creación de juegos sencillos sobre los años 80, aunque se trataban de kits limitados y que solo servían para la creación de juegos de un solo género.

Todo esto cambió con la llegada del 3D en los años 90, cuando algunas desarrolladoras decidieron separar el motor de renderizado de los recursos gráficos del juego y su lógica, especialmente en el género de los juegos de disparos en primera persona (o FPS, por sus siglas en inglés), lo que ahorraba costos a la hora de crear secuelas para estos, ya que solo tienen que modificar los recursos del juego y su lógica. Los ejemplos más notables fueron los de las empresas id Software y su saga de juegos Quake, y Epic Games y su saga de Unreal, hasta el punto de que otras empresas pagaban licencias para utilizar estos motores para sus juegos y así ahorrarse el trabajo de hacer uno propio. Esto no solo suponía una ayuda económica para estas desarrolladoras, que les ayudaba a crear nuevos juegos más cómodamente, si no que algunas de estas empresas llegaron a invertir más tiempo y dinero en el desarrollo de estos motores para su comercialización. (3)

Actualmente, estos motores son el centro de la industria, ya que se encargan de encargan de la renderización y la optimización de los recursos para que el desarrollador del juego solo tenga que preocuparse de la lógica y de los recursos como imágenes o modelos para reducir los costos tanto económicos como de tiempo a la hora de desarrollar juegos.

A la hora de elegir el motor con el que desarrollar un videojuego, se deben tener en cuenta muchos factores, como los objetivos que se quieren alcanzar y los recursos de los que se disponen, no solo monetarios, sino también en la experiencia de los trabajadores, y evaluar la importancia de las diferentes herramientas que ofrecen cada uno de ellos. A continuación, hablaremos de tres de los motores más utilizados en la

industria de los videojuegos, Unity, Unreal Engine y CryEngine, y nos centraremos en sus ventajas, inconvenientes, precio y disponibilidad. Nos centraremos en estos motores más extendidos en la industria ya que un gran número de desarrolladores conlleva una gran comunidad disponible a solucionar dudas y a crear tutoriales básicos para estas herramientas.

2.1.1. Unity

Unity[4] es uno de los motores más usados en la industria de los videojuegos gracias a ser un motor grafico fácil de instalar y aprender mientras ofrece herramientas de desarrollo comparables a las de grandes videojuegos con alto presupuesto. Además, al tener una versión gratuita para desarrolladores con bajo presupuesto y aprendizaje por más tiempo que los demás, ha permitido crear una gran comunidad de desarrolladores y entusiastas dispuestos a ayudar con los problemas y dudas que te puedan surgir, y con una gran cantidad de guías y tutoriales disponibles, algunos incluso oficiales creados por los mismos desarrolladores del motor. Además, aunque el motor en si no es tan rico en herramientas como algunos de sus competidores, la comunidad puede crear sus propias herramientas y compartirlas con el resto de los desarrolladores. Otra característica es que, aun habiendo sido creado en C++, el lenguaje utilizado para programar en Unity es C#, que hace que no sea necesario preocuparse en algunos aspectos de optimización requerido en C++ como el manejo de la memoria, lo cual agiliza la creación de juegos. [5]

A pesar de esto, aunque Unity permita crear juegos con estilos artísticos muy variados, si el objetivo es crear juegos con gráficos relistas, no es la opción adecuada ya que es el más atrasado de los tres en este aspecto. Además, aunque programando en C# hace que no sea tan necesario preocuparse por la optimización de recursos dado que el propio lenguaje lo controla por el desarrollador, algunos desarrolladores prefieren tener más control sobre que hace el motor para una optimización más avanzada y centrada en el género o mecánicas del juego particular, haciendo otros entornos donde se programa en C++ más tentadores para grandes proyectos.

Como se ha destacado antes, Unity tiene una versión gratuita para estudiantes y proyectos de bajo presupuesto, pero al llegar a una cierta cantidad de ingresos debes pagar por una licencia. Cuando tus ganancias pasan de los 100.000 dólares anuales, debes pagar 40 dólares mensuales por una licencia para poder seguir utilizándolo, o 150 dólares mensuales si tus ingresos superan los 200.000 anuales. Unity está disponible tanto para Windows como para Mac OSX como para algunas distribuciones de Linux, y te permite compilar los videojuegos que creas para una gran cantidad de plataformas, incluido WebGL para videojuegos que se deben reproducir en navegador.

[4] https://unity.com/
[5] https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you

2.1.2. Unreal Engine

Unreal Engine[6] es otro de los grandes motores de la industria por su gran potencia gráfica, considerada una de las mejores y más avanzadas, y que está en constante evolución hacia gráficos cada vez más realistas, como la optimización automática de los distintos polígonos que componen un modelo 3D, permitiendo así el uso de modelos de un detalle solo visto antes en cine.[7] Además, permite un gran control sobre este gracias no solo a que se usa programando en C++, sino que también te da acceso al código fuete no solo para entenderlo mejor a nivel de programación, sino también para modificarlo si fuese necesario para estar aún más optimizado de lo que ya es de base y personalizado para cada proyecto. También incluye una gran variedad de herramientas para crear tus videojuegos, considerado el más variado en este aspecto. Otra gran ventaja que ofrece es que te permite programar usando bloques, que permite a alguien que no sea familiar con la programación crear códigos sencillos para sus juegos o crear pequeños prototipos, aunque se suele recomendar usar C++ si planeas crear un proyecto a mayor escala.[8]

El problema con este motor es que no es tan fácil de controlar como otros, ya que está más diseñado para grandes desarrolladoras que ya están familiarizados con el entorno y el método de trabajo de un videojuego, hasta el punto de no ser un motor recomendable para trabajar una sola persona o en proyectos pequeños. Además, C++ es más complicado de controlar y entender que otros como C#, lo cual añade más dificultad a su aprendizaje.

Unreal Engine te permite su uso gratuito completamente mientras que tu proyecto no supere el 1.000.00 de dólares en ganancias, a partir de los cuales debes pagar un 5% de lo que ganes. Está disponible para desarrollar tanto en Windows como en Mac OSX como en Linux y te permite compilar en las plataformas más importantes de la industria.

2.1.3. CryEngine

CryEngine[9] no está tan extendido como los mencionados en los últimos apartados, pero también se han ganado su sitio en la industria gracias a sus avances en cuanto al ambiente y entorno del juego gracias a su excelente motor de iluminación y de líquidos, y de las herramientas de creación de entornos que ofrece. Al igual que Unreal Engine, también se programa en C++, dando un gran control sobre la optimización del juego, y la creación de fragmentos de código sencillos mediante programación por bloques, aunque una vez mas no sea recomendable para grandes proyectos. Además, este motor incluye una inteligencia artificial básica con funciones simples. [10]

[6] https://www.unrealengine.com/en-US/
[7] https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5
[8] https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unreal-engine-4-the-right-game-engine-for-you
[9] https://www.cryengine.com/
[10] https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-cryengine-the-right-game-engine-for-you

El principal problema con este motor es que está centrado a la creación de videojuegos de disparos en primera persona, e intentar crear algo con este motor fuera de este género requiere un entendimiento más avanzado de este. Además, es un motor muy difícil de entender si no se tiene experiencia anterior con este tipo de entornos.

CryEngine usa un sistema de monetización similar a Unreal Engine, debiendo pagar el 5% de tus ganancias anuales que excedan los 5.000 dólares. Un problema más es que te fuerza a utilizar Windows para poder acceder a esta herramienta, aunque te permite compilar tus proyectos a las plataformas más comunes e importantes.

2.2. Ajedrez

En la actualidad hay muy pocas variantes del ajedrez clásico que todos conocemos, y todas estas variantes tienen una cosa en común, que es que todas estas se pueden jugar utilizando las mismas piezas que el ajedrez estándar. La mayoría de los sitios web donde jugar ajedrez en línea te permiten jugar a todas o casi todas estas variantes de ajedrez tanto contra la máquina como contra otra gente, aunque su popularidad no es tanta como la del ajedrez estándar.

2.2.1. Ajedrez pasapiezas y Crazyhouse

El ajedrez pasapiezas[11] se trata de una variante por parejas en la cual se juegan dos partidas de ajedrez en paralelo, donde en uno de los jugadores del equipo juega con las piezas negras y el otro con las blancas. El reglamento respecto a cómo las fichas se mueven y capturan permanece intacta, pero la pieza capturada pasa al otro tablero bajo el control de su compañero. En este caso, el compañero puede elegir si realizar un movimiento normal o si colocar una de las piezas capturadas por su compañero en cualquier casilla vacía del tablero. En esta variante gana el equipo que haga el primer jaque mate.

La variante de Crazyhouse[12] se juega igual con la diferencia de que solo juegan dos personas una contra la otra. Las piezas capturadas pasan a ser propiedad de quien la capture, el cual puede decidir colocarla en el tablero en una casilla vacía en vez de realizar un movimiento.

2.2.2. Ajedrez 960

El ajedrez 960 fue creado por el campeón del mundo Bobby Fischer buscando una variante del ajedrez que recompense más la creatividad de los jugadores y no tanto la preparación previa y el estudio de aperturas. En esta variante las reglas del juego son las mismas y solo cambia la posición inicial del tablero, que se decide de manera aleatoria con el lanzamiento de un dado. Los peones se mantienen en la misma posición ocupando la segunda fila de cada jugador, pero el resto de las piezas se colocan en la primera fila siguiendo estos pasos:

[11] https://www.chessvariants.com/multiplayer.dir/tandem.html
[12] https://www.chessvariants.com/other.dir/crazyhouse.html

El primer paso es colocar los alfiles, primero uno en casillas negras y luego el otro en casillas blancas, lanzando un dado y colocando el alfil en la casilla indicada por el dado contando desde la izquierda. En caso de sacar un 5 o un 6 se vuelve a tirar.

El segundo paso es colocar la reina en la casilla que indique el dado, contando desde la izquierda y saltando las casillas ya ocupadas por los alfiles.

El tercer paso es colocar los caballos siguiendo el mismo procedimiento que en el segundo paso y volviendo a tirar los dados que saquen un número mayor al de casillas disponibles como en el primer paso.

Una vez realizados estos pasos, falta colocar las torres en las casillas vacías que estén más próximas a cada lateral y el rey en la casilla restante. Las piezas negras obtienen la misma posición inicial, pero en espejo.

2.2.3. Antichess

Esta variante del ajedrez conserva la posición inicial, pero se invierte el objetivo del juego: para ganar tienes que perder todas tus piezas.[13] El movimiento y la captura es igual que en el ajedrez normal, con el cambio de que, si un jugador tiene la posibilidad de capturar una pieza enemiga, este está forzado a capturarla. En caso de que existan varias capturas posibles, el jugador puede elegir cuál de ellas realizar. En esta variante se ignoran las mecánicas relacionadas con el rey y los jaques y se convierte en una ficha más sin mayor importancia.

2.2.4. Software de creación de variables y juego en línea

Con la llegada de la informática, se han realizado varios intentos para facilitar la creación de variantes del ajedrez.

Sin ir más lejos, la web chessvariants.com ofrece poder jugar a algunas de las variantes que tiene registradas en su biblioteca[14], pero esta herramienta está desactualizada y solo permite jugar a través de correspondencia, es decir, a base de intercambio de correos cada vez que se realice un movimiento.

Otras webs como chess.com y lichess.org permiten ofrecen jugar en línea y en tiempo real a muchas variables del ajedrez ya mencionadas como el ajedrez Crazyhouse mencionado en el apartado 2.2.1 o el Antichess mencionado en el apartado 2.2.3, así como permitir iniciar partidas con la disposición inicial de las piezas modificada. Además, en el caso específico de chess.com existe la posibilidad de jugar un ajedrez de 4 jugadores, donde también permite modificar la disposición inicial de las piezas e incluso pudiendo incluir una serie de piezas que no existen en el ajedrez estándar.

[13] https://www.chessvariants.com/diffobjective.dir/giveaway.html
[14] https://www.chessvariants.com/play/pbm/

En ambas webs, pero, el usuario sigue estando limitado a las piezas y tableros base que las propias webs les ofrece, además de que toda pieza, tablero y variante que quieran implementar sería un trabajo nuevo que deben hacer los desarrolladores de estas webs.

2.3. Crítica al estado del arte

Actualmente el ajedrez conserva el mismo problema que tenía antiguamente, y es que, si alguien quiere crear una nueva variante que incluya nuevas piezas o cambiar el tablero de alguna forma tienes que crear un nuevo programa desde cero, lo cual es muy costoso.

Como se ha mencionado en la introducción, muchos videojuegos han visto su vida alargada gracias a las modificaciones que ha creado la comunidad que, sin la ayuda de herramientas ofrecidas por los desarrolladores, podrían haber sido una tarea excesivamente costosa e incluso imposible. Hace falta algún tipo de herramienta que permita la creación de variantes de ajedrez de una manera más cómoda y que no sea necesario el reprogramar todas las mecánicas base de este.

2.4. Propuesta

Con este proyecto se pretende cubrir esta necesidad de un software que te permita la creación de nuevas variantes del ajedrez con las que jugar, permitiendo no solo facilitar el compartir estas variantes, si no también eliminando la necesidad de crear un nuevo sistema de cero cada vez que se desee crear una nueva variante.

Por tanto, se pretende crear un sistema que permita la creación de nuevas piezas de una manera cómoda para el usuario, permitiendo una gran variedad de movimientos distintos, a la vez de un sistema para implementar dichas piezas en nuevos tableros iniciales con la posibilidad de utilizar casillas de distintos tipos y de tamaños y formas configurables.

De esta forma se pretende aumentar aún más la variedad estratégica que ya trae de por si el ajedrez y desatar la creatividad tanto de los usuarios que pretendan probar su habilidad a la hora de diseñar nuevas variantes como de la creatividad estratégica de los jugadores de estas variantes.

3. Análisis del problema

Aun conociendo ya el problema y la solución propuesta, es necesario analizar el problema en más profundidad para conocer mejor los requisitos del videojuego y, con esto, los pasos a seguir.

3.1. Análisis de soluciones

En este apartado se pretende repasar las más importantes para este proyecto en particular, pero la mayoría de estos se comparten con el resto de los videojuegos, incluso de géneros totalmente distintos.

En primer lugar, hay que tener claro en que plataforma se pretende publicar el juego. Cada plataforma acoge de manera distinta los diferentes géneros de juegos existentes dependiendo de los controles que tenga a su disposición y las diferentes situaciones en los que estas estén disponibles. En el caso de los juegos de estrategia por turnos, y en especial el ajedrez, se puede exportar a una gran cantidad de plataformas gracias a que puede ofrecer tanto partidas rápidas y dinámicas como partidas lentas y sosegadas, por lo que tenemos un gran abanico de posibilidades para elegir. Además, no es necesario encerrarse en una única plataforma, ya que como se ha comentado en el apartado 2.1, los motores de hoy en día permiten crear el juego solo una vez y publicarlo en diferentes plataformas.

Otro aspecto que tener en cuenta es que controles se van a poder utilizar para interactuar con el juego. Esta decisión se ve muy influenciada por la plataforma escogida o escogidas, ya que algunas son más limitantes que otras en este aspecto, y suele ser muy determinante en la satisfacción del usuario con el videojuego. En los juegos por turnos, a no ser que estos turnos tengan un límite de tiempo que requiera de jugar de manera rápida, suele ser indiferente el control utilizado mientras sea posible navegar por todas las opciones.

Otra serie de aspectos que hay que tener en cuenta a la hora de desarrollar un videojuego son las distintas mecánicas dentro de este. En el caso de este proyecto, siendo un juego de estrategia por turnos, hay que plantearse cuestiones sobre cómo funcionan estos turnos, si tienen algún límite de tiempo, que pasa si ese tiempo se acaba, que se puede hacer en un turno... En este caso, se plantean dos soluciones posibles:

- Mantenerse en la línea del ajedrez clásico, permitiendo solo un movimiento a cada jugador por turno, pudiendo elegir si estos tienen un límite de tiempo o no. En caso de que sí que haya límite de tiempo, al acabarse, el jugador al que se le haya acabado el tiempo pierde.
- Permitir modificar estas reglas, permitiendo realizar varios movimientos en un solo turno o que, en el caso de que los turnos tengan un tiempo límite, que los efectos en la partida sean más diversos que su brusca finalización.

A parte, se debe tener en cuenta cómo funciona la aplicación de manera interna y, en nuestro caso, como el usuario va a poder modificar el contenido del juego para personalizarlo al gusto. Como se ha comentado en el apartado 3.3, se utilizarán una serie de ficheros XML para crear y modificar tanto piezas como tableros para jugar. A la hora de diseñar un sistema para los archivos XML con las que se crearán y modificarán las piezas y los tableros con los que jugar, se han barajado tres opciones distintas:

- La primera opción es crear las piezas dentro del sistema y que los archivos XML indiquen los tableros sobre los que se juega. Aunque esta aproximación supondría que se pueda optimizar la interacción entre las diferentes piezas, no

permitiría la libre creación de nuevas piezas, uno de los puntos más importantes de este proyecto.

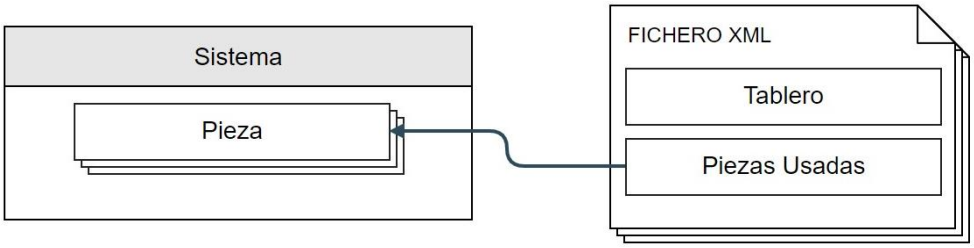

Ilustración 1 - Esquema del sistema en el caso de la primera opción planteada, donde las piezas están ya implementadas en el juego y los tableros hacen referencia a estas.

- La segunda opción sería indicar el formato del tablero y el funcionamiento de sus piezas en el mismo documento XML. Aunque esto soluciona el problema de la opción anterior, viene con el problema de que crearía archivos XML muy complejos, grandes y difíciles de leer y modificar.

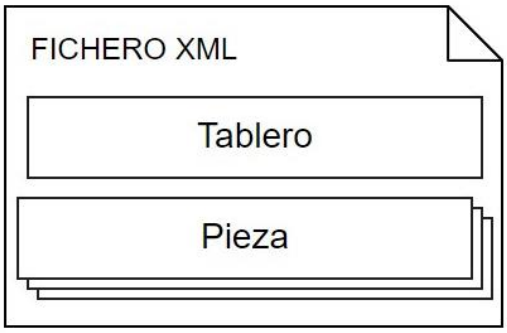

Ilustración 2 - Esquema del sistema en el caso de la segunda opción planteada, donde en un solo fichero tenemos la descripción del tablero y de las piezas.

- La tercera opción se trata de separar las piezas y los tableros en archivos distintos y que los tableros hagan referencia a las piezas que utilicen. De esta manera, obtendremos archivos más sencillos y legibles, pero esto puede dar problemas a la hora de compartir estas piezas y tableros con otros usuarios si varias piezas se crean bajo el mismo identificador.

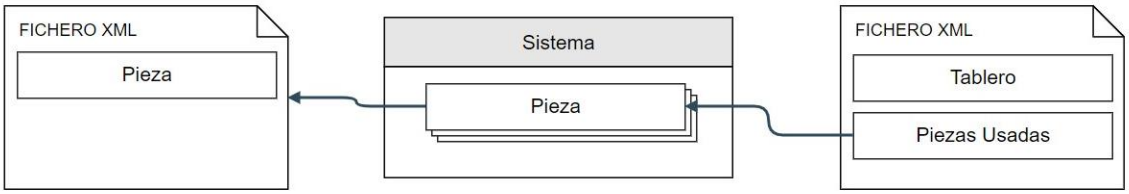

Ilustración 3 - Esquema del sistema en el caso de la tercera opción planteada, donde existen dos tipos de ficheros XML, uno de piezas y otro de tableros, estos últimos haciendo referencia a los primeros.

	Pros	Contras
Primera opción	Pocos archivos XML Archivos XML simples y pequeños Interacción entre piezas optimizada	No es posible crear piezas nuevas Formatos de tableros limitados
Segunda opción	Pocos archivos XML Personalización completa del juego	Archivos XML largos y complejos No hay reutilización de código/piezas
Tercera opción	Personalización completa del juego Reutilización de código/piezas Archivos XML más sencillos	Dos tipos de archivos XML Posible solapamiento de IDs de piezas Posible uso de IDs no existentes

Tabla 1 – Comparación entre las 3 opciones planteadas para el sistema en el que se guardarán los ficheros XML.

3.2. Solución propuesta

La plataforma elegida para el desarrollo del juego se va a centrar en el PC, dado que es más sencillo probar el funcionamiento de este desde dentro del editor y que es más sencillo que el usuario pueda introducir los archivos XML para las diferentes variantes. No obstante, se desarrollará la interfaz del juego de manera que sea compatible con la mayor cantidad de plataformas posibles para habilitar la posibilidad a futuro de extender el producto a más público.

Para los controles a utilizar, se ha optado por admitir las entradas del teclado y el ratón, el primero utilizando las flechas de dirección para mover el cursor por la mesa y la tecla enter para seleccionar la pieza o el movimiento a realizar, y el segundo tanto pudiendo hacer clics individuales para seleccionar la pieza o el movimiento tanto como haciendo clic y arrastrando la pieza que se quiera mover. Esto nos permitirá en un futuro adaptar las teclas a los diferentes controles de diferentes periféricos, agilizando la adaptación del juego a otras plataformas.

Respecto al funcionamiento del juego, para simplificar tanto los documentos XML como del algoritmo interno que controle la partida, se opta por la opción de mantenerse dentro del funcionamiento original del juego de un movimiento por turno. En un futuro se puede permitir realizar pequeñas modificaciones al reglamento original creando modificaciones opcionales de los archivos.

En cuanto al formato de los archivos XML, se va a optar por la opción de tener las piezas y los tableros por separado y hacer que los tableros referencien a las piezas que utilizan. De esta manera, se puede reutilizar un mismo archivo múltiples veces para múltiples tableros distintos, además de hacer más fácil para el usuario encontrar los errores que puedan surgir mientras este crea sus propias variantes.

3.3. Requisitos

En este caso, al tratarse de un sistema donde la interacción con el usuario es tan importante o incluso más que en otros sectores, debemos dividir los requisitos en requisitos funcionales y requisitos de interacción con el usuario.

3.3.1. Funcionalidad

- Leer los archivos XML de las piezas, obteniendo tanto sus datos básicos identificativos, como los movimientos que realiza, como la imagen o modelo que lo representa.
- Leer los archivos XML de los tableros, obteniendo las dimensiones y las casillas disponibles, así como obteniendo las piezas utilizadas.
- Que el sistema pueda detectar cuando un archivo XML contiene errores, cancelando su carga e informando al usuario de los errores que contiene.
- Poder mover las piezas en partida como está indicado en su respectivo archivo XML.
- Poder leer el estado de la partida.
- Declarar que movimientos puede realizar cada pieza y reglas del juego, evitando así movimientos ilegales e identificando cuando la partida termine.
- Implementar un sistema de turnos para alternar entre los jugadores y que pueda incluir un temporizador para limitar el tiempo tanto de los turnos como de la partida.
- Implementar todos estos sistemas dentro del motor de Unity.

3.3.2. Interacción con el usuario

- Ofrecer al usuario una interfaz que le muestre toda la información del estado de la partida, como el jugador que tiene el turno en ese momento, el tiempo restante para cada jugador en caso de estar jugando con límite de tiempo, como el estado de las piezas en el tablero.
- Poder interactuar con las piezas, utilizando el ratón para mover a estas por el tablero en su turno.
- Poder marcar casillas y dibujar flechas sobre el tablero, utilizando el ratón para marcar que casillas marcar o de que casilla a que casilla dibujar la flecha.

4. Diseño de la solución

Por una parte, hablaremos de las herramientas utilizadas para la creación de este videojuego y, por otra, de cómo se estructura este de manera interna.

4.1. Tecnologías utilizadas

A la hora de elegir las herramientas, lo primero sería elegir el motor a utilizar para nuestro proyecto, ya que, dependiendo de que motor utilicemos, vamos a necesitar una serie de herramientas u otras para complementarlo.

4.1.1. Motor elegido

A la hora de decidir qué motor utilizar, se procede a comparar los aspectos más destacados de los diferentes motores que se han descrito en los diferentes subapartados del apartado 2.1, teniendo en cuenta su precio, lenguaje utilizado, la complejidad del

uso de cada una, la fidelidad visual alcanzable, la cantidad de plataformas a las que se puede exportar, la optimización que ofrece y las herramientas que ofrece, además de explicar la importancia de estos factores para el caso de este proyecto.

Nombre del motor	Unity	Unreal Engine	CryEngine
Precio licencia	0$, 40$ o 150$ mensuales por desarrollador, dependiendo de los ingresos anuales.	Los primeros 1.000.000$ que ganes con un juego son gratis, después se paga el 5% de las ganancias.	Los primeros 5.000$ que ganes con un juego al año son gratis, después pagas el 5% de las ganancias.
Lenguaje utilizado	C#	C++	C++
Complejidad de uso	El más sencillo de utilizar.	Algo más complejo que la media.	Muy complejo de usar, especialmente si se pretende crear un juego que no sea de disparos en primera persona.
Fidelidad visual	Se queda algo atrás en comparación con otros.	La mejor en términos generales.	La mejor en cuanto a entornos.
Plataformas para exportar	18	13	8, con soporte para smartphones en desarrollo
Optimización	La más básica y genérica entre los tres.	De los más optimizados entre la competencia.	Centrada en juegos de disparos en primera persona.
Herramientas	El que menos ofrece, pero muy ampliable.	El que más variadas ofrece de los tres.	Las más avanzadas en cuanto a creación de entornos. Incluye IA básica para juegos de disparos en primera persona.

Tabla 1 – Comparación entre los 3 motores planteados para el desarrollo del trabajo.

Respecto al precio de la licencia, al no haber recibido ningún ingreso inicial para los gastos, no es necesario pagar licencia en ninguno de los tres motores. Aun así, para desarrollos más grandes que hayan recibido una financiación previa, esto puede ser un factor muy importante.

Respecto al lenguaje de programación utilizado, aunque los programadores más experimentados prefieran el uso de C++ por la optimización que se puede alcanzar, esto también puede suponer un retraso en el desarrollo de un videojuego. En cambio, esto no es necesario en C#, ya que el compilador se encarga de ello por nosotros. Esto es la mayor ventaja de Unity, ya que permite crear juegos de una manera más rápida y sencilla para desarrolladores novatos, a la vez que su mayor inconveniente, ya que, si se pretende crear juegos complejos, Unity puede llegar a ser más una molestia que una ayuda. En el caso de este proyecto, no se pretende crear algo que requiera una gran complejidad mecánica ni mucho menos visual, por lo que el uso de C# en Unity puede ayudar a recortar en costes temporales.

Respecto a la complejidad de uso, Unity es el que lidera entre nuestras tres opciones, no solo por su sencillez, sino también por la gran cantidad de tutoriales y de preguntas respondidas por la comunidad, como hemos mencionado en el apartado 2.1.1. Mientras tanto, Unreal Engine es algo más complejo de utilizar en parte por la gran variedad de herramientas que ofrece, aunque se ha ido simplificando con el paso del tiempo. CryEngine sería el más complejo de utilizar, especialmente si no se ha trabajado

anteriormente en la industria de los videojuegos, y se acompleja mucho más si intentas utilizar el motor para algo fuera del género de los disparos en primera persona.

Respecto a la fidelidad visual, no es algo muy importante aquí, ya que se van a utilizar o entornos 2D o entornos 3D con modelados muy sencillos sin apenas animaciones, por lo que cualquiera de los motores nos puede ser útil en el desarrollo del proyecto.

Respecto a las plataformas a las que poder exportar nuestro proyecto, aunque Unity sea el que más posibilidades ofrezca, realmente todos ofrecen exportar a las plataformas principales de la industria, con la excepción más notable siendo CryEngine sin la posibilidad de exportar a smartphones.

Respecto a la optimización, como hemos comentado anteriormente en este apartado, los motores que utilizan C++ para programar ofrecen muchas más posibilidades a la hora de optimizar al máximo el uso de recursos de nuestro juego, pero requieren conocimientos más extensos y una mayor inversión en optimizarlos, mientras que C# hace una optimización más sencilla pero automática. Aunque una buena optimización es necesaria en todos los videojuegos, en nuestro caso, al tratarse de un juego por turnos, no es tan crítica como en otros géneros en tiempo real. Por tanto, cualquiera de los tres nos es útil para el proyecto.

Por último, respecto a las herramientas que ofrecen, Unity es el que menos ofrece de base de todos, pero ofrece herramientas especializadas en el desarrollo de juegos en 2D, mientras que el resto ofrece muchas más herramientas que probablemente no utilicemos en este proyecto. Dado que este proyecto se trata de un juego de estrategia por turnos, CryEngine puede convertirse rápidamente en una molestia, dado que se centra mucho en el desarrollo de juegos en primera persona y hace que salir de este género complique mucho el desarrollo de nuestro videojuego. Por otra parte, Unreal Engine, aun siendo viable su uso para este tipo de juegos, se trata de un motor algo más complejo de usar y puede resultar tedioso especialmente si se trata de un desarrollo de una sola persona, como es el caso de este proyecto. En este caso, Unity sería el más aconsejable para este tipo de proyectos, ya que es un motor que facilita el desarrollo individual gracias a su sencillez en cuanto a herramientas y a la gran comunidad que se ha labrado durante los años dispuesta a ayudar en lo que haga falta, además de que la programación en C# permite despreocuparte de algunos detalles que serían importantes a la hora de programar en C++ a costa de un rendimiento algo menor, algo que no es tan importante en un juego de estrategia por turnos. En conclusión, se ha elegido Unity como motor para el desarrollo de este proyecto.

4.1.2. Otras herramientas

Una vez se conoce el motor elegido, podemos elegir el resto de las herramientas que utilizaremos durante el desarrollo de esta aplicación.

Como sistema operativo sobre el que trabajar, se ha seleccionado Windows 10 simplemente porque ya se encuentra instalado en la máquina sobre la que se va a trabajar.

El lenguaje sobre el que se va a programar es C# porque es el único lenguaje disponible para Unity. Además, se utilizará XML para permitir a los usuarios crear y utilizar las distintas piezas y tableros sobre los que jugar.

Como entorno integrado de desarrollo se utilizará Visual Studio 2017 por dos razones. La primera es que este tiene una serie de compatibilidades con Unity que permite el uso de sus librerías de una manera más sencilla y rápida gracias a su autocompletado, además de la ya incluida revisión de código estática para evitar errores de sintaxis y lógicos básicos. La segunda es que Visual Studio 2017 tiene incluido herramientas para el diseño y creación de documentos XML, lo cual facilitará el trabajar con este tipo de documentos para las piezas y tableros personalizados.

Para el control de versiones se ha optado por el uso de GitHub y su aplicación de escritorio GitHub Desktop simplemente por la familiaridad y la experiencia que se ha obtenido durante el curso académico para la realización de trabajos en grupo.

Por último, para llevar un control de que funcionalidades se debe incluir en el videojuego, cuando y que dependencias tienen las unas de las otras se ha optado por el uso de Trello por su sencillez, su fácil visualización y su flexibilidad.

4.2. Arquitectura del sistema

En el caso de este trabajo, hay dos apartados a los que tenemos que prestar atención a la hora de diseñar la arquitectura de nuestro sistema: el diseño de los ficheros XML y el del mismo juego. Aunque no es necesario que ambos tengan exactamente la misma estructura, sí que se influencian entre ellas. En este caso, nos centraremos primero en la estructura de los ficheros XML y como se deben guardar para que el sistema los lea, ya que es lo que el usuario final tendrá que utilizar de continuo y, desde ahí, diseñaremos la estructura interna. Antes de entrar en detalles, pero, hay que especificar que se necesita exactamente de cada tipo de elemento, además de especificar como afectaría cambiar la forma de las casillas del tablero.

Dado que dependiendo de la forma de las casillas (si estas son cuadradas, hexagonales o triangulares) y el número de dimensiones que el tablero utiliza (2 dimensiones, 3 o incluso más) cambia como representar las coordenadas de una casilla o como representar un movimiento de una pieza, se ha optado por hacer que los diferentes tipos de tableros tengan ligeras variantes de como representan y guardan sus piezas o tableros. La implementación de esta aplicación se realizará de manera que sea posible implementar nuevos tipos de tableros reutilizando la mayor cantidad de código posible, pero no será posible introducir una pieza diseñada para un tipo de tablero en otro tipo de tablero.

Con respecto a las piezas, lo más importante para poder usarlas es el listado de movimientos que esta puede realizar. Para los movimientos, se ha decidido dividirlos en dos apartados: tipos y estilos. Los tipos de movimiento hace referencia a si este permite únicamente la captura de piezas enemigas, únicamente el movimiento por el tablero sin captura o cualquiera de las dos. Por otro lado, los estilos hacen referencia a como realiza el movimiento, indicando si estos se pueden repetir de manera indefinida o no y si estos pueden ignorar si las casillas en medio del camino están ocupadas o no.

De un movimiento no solo es necesario saber su tipo y su estilo, si no a que casilla se pueden mover, indicado con una serie de coordenadas relativas a la posición de la pieza.

De un tablero es necesaria mucha más información, ya que estos son los que indican realmente la variante de ajedrez a la que se va a jugar. Lo más primordial es, en este caso, un listado con los jugadores que participan, incluyendo su color, equipo y dirección a la que apuntan sus piezas, una lista de IDs de las piezas que se utilizan y la misma posición inicial del tablero. Además, se puede incluir una lista con los denominados "movimientos especiales", que son los movimientos que necesitan que se cumplan una serie de condiciones para poder realizarse.

4.2.1. Estructura de los ficheros XML

Como hemos comentado en el apartado 2.4, se crearán dos tipos distintos de ficheros, uno para las piezas y otra para los tableros, y los ficheros de tableros harán referencia a los ficheros de piezas para obtener toda la información de estas. Cada tipo tendrá una carpeta para almacenar todas las entradas, cada entrada dentro de una subcarpeta, la cual contendrá su archivo XML y una imagen que la acompañará para identificarla.

En el caso de los ficheros XML de las piezas, solo es necesario dos cosas:

- Una serie de metadatos para obtener información básica de la pieza, entre ellos un identificador para que la pieza pueda ser referenciada desde los ficheros XML de tablero.
- Una lista con los movimientos posibles por la pieza. Cada movimiento debe indicar la casilla objetivo con coordenadas relativas a la posición de la pieza, además de indicar su estilo como nombre del elemento y su tipo como atributo.

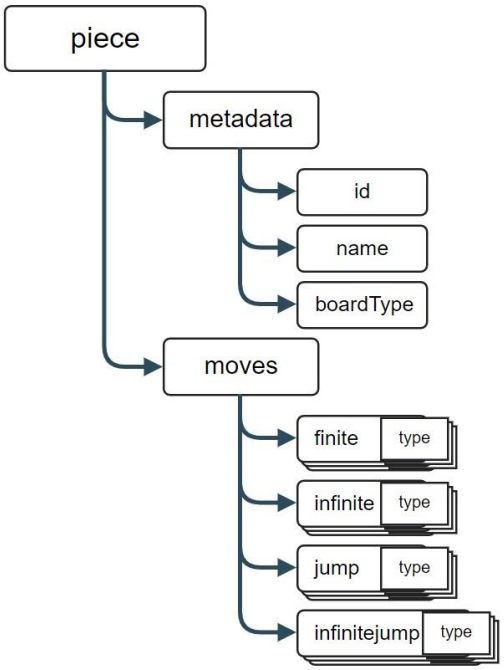

Ilustración 4 - Estructura mínima del fichero XML de una pieza.

En el caso de los ficheros XML de los tableros, es necesaria mucha más información, ya que es la que contendrá la definición de la variante que se va a jugar. Estos ficheros están compuestos de:

- Una serie de metadatos para obtener información básica del tablero, al igual que las piezas.
- Una lista de los jugadores que componen la partida, definiendo si van en equipo o no y la dirección de sus piezas.
- Las piezas que componen la variante, junto a información relativa a el rol que componen en esta.
- La posición inicial de la partida, así como que casillas permiten promocionar las piezas tipo peón a piezas mejores.
- Una lista con los que se han denominado como “movimientos especiales”.

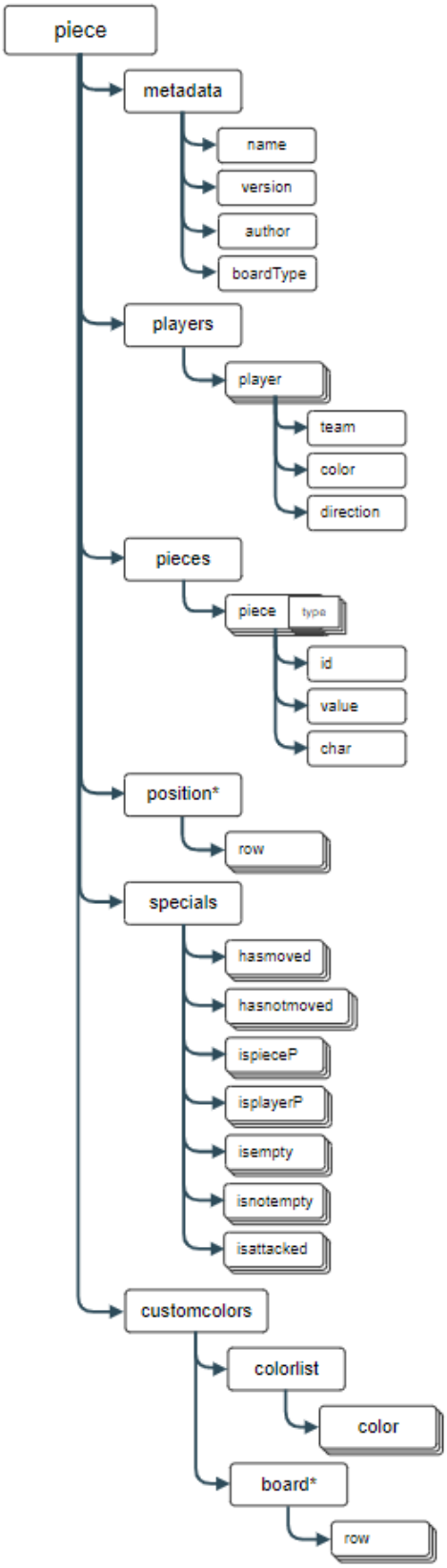

Ilustración 5 - Estructura completa del fichero XML de un tablero. El interior del objeto "position" y del objeto "customcolors/board" dependen del tipo de tablero, aquí se asume el tablero clásico bidimensional de casillas cuadradas.

En el anexo 10.1 se explicará más en profundidad la estructura no solo de los archivos XML, sino también de otros archivos adjuntos como imágenes y la estructura que deben tener para funcionar correctamente.

4.2.2. Librería interna de piezas y tableros

Internamente, se utilizará un objeto que comprobará que los ficheros estén correctamente formateados y que contiene todos los elementos necesarios para poder ser usados, cargará y almacenará todas las piezas y tableros para poder usarse en el resto de la aplicación. Este objeto será el único objeto disponible en todo el juego en todo momento. Dado que los tableros referenciarán a las piezas a través del ID de estas, se ha decidido utilizar la estructura de diccionarios de C#, utilizando las ID de las piezas como clave, relacionándolas con objetos que contengan toda la información necesaria de estas. De la misma manera se almacenarán los tableros, esta vez utilizando su nombre como clave.

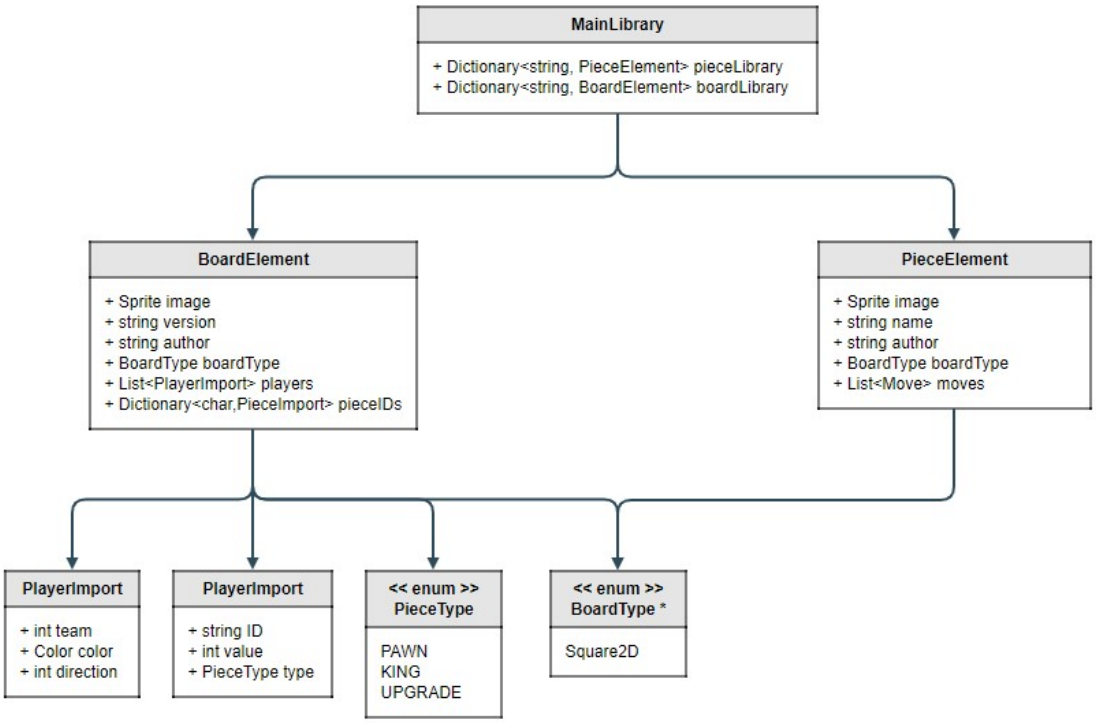

Ilustración 6 - Diagrama UML de la librería de piezas y tableros. El enumerador marcado con un asterisco se puede ampliar en un futuro para permitir nuevos tipos de tableros.

4.2.3. Estructura del sistema en partida

En partida serán necesarios una serie de controladores que hablen entre ellos para comprobar el estado de la partida y actualizando con las instrucciones introducidas por el usuario. Cuatro de ellos son los principales para poder jugar: uno que genera las casillas y las piezas siguiendo las indicaciones introducidas anterior mente por el usuario en los archivos XML, otro que detecte las indicaciones que realice el usuario para que el sistema responda a esta, otro para coordinar los turnos de los diferentes

jugadores y una principal que coordine a todos los anteriores, además de coordinar la posición de las piezas.

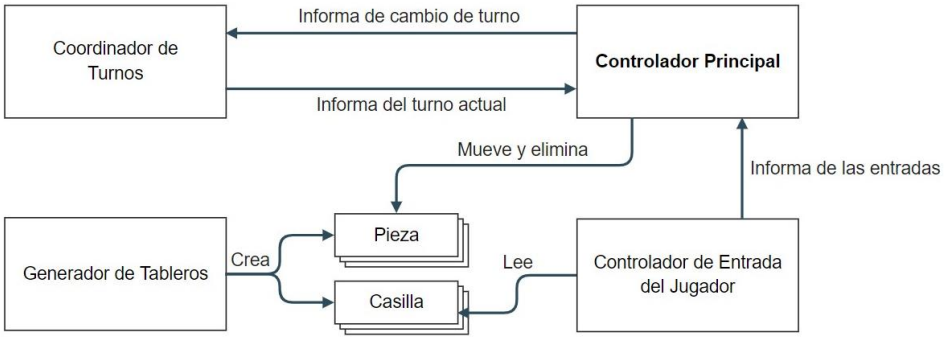

Ilustración 7 - Diagrama básico de la estructura de controladores y como se comunican entre ellos.

5. Implementación de la solución

Ahora comentaremos como se han implementado las distintas partes del proyecto, las diferentes dificultades que han aparecido durante este y como se han solucionado.

5.1. Lector de archivos XML

El juego empieza la lectura de los archivos XML tan solo se ejecuta, empezando por las piezas, ya que los tableros referencian y utilizan a estos ficheros.

Para las piezas, se comprueban tanto sus datos como los movimientos introducidos y la imagen adjunta, y lo guarda todo en un diccionario usando la ID de sus metadatos como clave. Durante la comprobación, se va generando una lista con todos los errores que contienen las piezas para informar al usuario al finalizar, llegando a ignorar las piezas que no contengan la información mínima necesaria o que tengan una ID duplicada.

```
<piece>
  <metadata>
    <ID>ile.001</ID>
    <name>Battering Ram</name>
    <version>v1.0</version>
    <author>José Manuel</author>
    <boardType>Square2D</boardType>
  </metadata>

  <moves>
    <infinitejump type="both">1,0</infinitejump>
    <infinitejump type="both">-1,0</infinitejump>

    <jump type="move">0,1</jump>
    <jump type="move">0,-1</jump>
    <finite type="move">0,2</finite>
    <finite type="move">0,-2</finite>

    <jump type="capture">1,1</jump>
    <jump type="capture">1,-1</jump>
  </moves>
</piece>
```

Ilustración 8 - Ejemplo de una pieza personalizada llamada Battering Ram (Ariete). Esta pieza puede moverse adelante y atrás igual que una torre, capturar diagonalmente como un peón y moverse dos casillas hacia los lados sin poder capturar.

Para los tableros, se comprueban tanto sus datos, las piezas utilizadas, los jugadores que participan, la posición inicial y los movimientos especiales. Al igual que con los ficheros de piezas, se van informando de los errores típicos que se puedan cometer a la hora de crear el fichero, como que falte información mínima necesaria, o que se intenten utilizar una pieza de un tipo de tablero distinto, guardando las que cumplan todas las condiciones necesarias en un diccionario usando el nombre como clave, e ignorando las que no lo hagan.

```
<board>
  <metadata>
    <name>A Funky Board</name>
    <version>v1.0</version>
    <author>José Manuel</author>
    <boardType>Square2D</boardType>
  </metadata>

  <players>
    <player>
      <team>-</team>
      <color>170,0,0</color>
      <direction>1</direction>
    </player>
    <player>
      <team>-</team>
      <color>0,0,170</color>
      <direction>-1</direction>
      <imagevariant>UpsideDown</imagevariant>
    </player>
  </players>
  <pieces>
    <piece type="pawn">
      <ID>classic.pawn</ID>
      <value>1</value>
      <char>P</char>
    </piece>
    <piece type="king">
      <ID>classic.king</ID>
      <value>99</value>
      <char>K</char>
    </piece>
    <piece type="upgrade">
      <ID>classic.knight</ID>
      <value>3</value>
      <char>N</char>
    </piece>
    <piece type="upgrade">
      <ID>classic.bishop</ID>
      <value>3</value>
      <char>B</char>
    </piece>
    <piece type="upgrade">
      <ID>classic.rook</ID>
      <value>5</value>
      <char>R</char>
    </piece>
    <piece type="upgrade">
      <ID>classic.queen</ID>
      <value>9</value>
      <char>Q</char>
    </piece>
    <piece type="upgrade">
      <ID>ile.001</ID>
      <value>7</value>
      <char>A</char>
    </piece>
  </pieces>
```

```
<position>
    <row>_,_,0:1,0:1,0:1,_,_</row>
    <row>_,0:1,0,2K,0,0:1,_</row>
    <row>0:1,2A,2R,2Q,2R,2A,0:1</row>
    <row>2N,0,2N,_,2N,0,2N</row>
    <row>2B,2B,_,_,_,2B,2B</row>
    <row>0,0,_,_,_,0,0</row>
    <row>2P,2P,2P,_,2P,2P,2P</row>
    <row>0,0,0,0,0,0,0</row>
    <row>0,0,0,0,0,0,0</row>
    <row>0,0,0,0,0,0,0</row>
    <row>1P,1P,1P,_,1P,1P,1P</row>
    <row>0,0,_,_,_,0,0</row>
    <row>1B,1B,_,_,_,1B,1B</row>
    <row>1N,0,1N,_,1N,0,1N</row>
    <row>0:2,1A,1R,1Q,1R,1A,0:2</row>
    <row>_,0:2,0,1K,0,0:2,_</row>
    <row>_,_,0:2,0:2,0:2,_,_</row>
</position>
<customcolors>
  <colorlist>
    <color>0,0,0</color>
    <color>255,255,255</color>
    <color>100,0,0</color>
    <color>0,0,100</color>
  </colorlist>

  <board>
    <row>1,1,3,3,3,1,1</row>
    <row>1,3,2,1,2,3,1</row>
    <row>3,2,1,2,1,2,3</row>
    <row>2,1,2,1,2,1,2</row>
    <row>1,2,1,1,1,2,1</row>
    <row>2,1,1,1,1,1,2</row>
    <row>1,2,1,1,1,2,1</row>
    <row>2,1,2,1,2,1,2</row>
    <row>1,2,1,2,1,2,1</row>
    <row>2,1,2,1,2,1,2</row>
    <row>1,2,1,1,1,2,1</row>
    <row>2,1,1,1,1,1,2</row>
    <row>1,2,1,1,1,2,1</row>
    <row>2,1,2,1,2,1,2</row>
    <row>4,2,1,2,1,2,4</row>
    <row>1,4,2,1,2,4,1</row>
    <row>1,1,4,4,4,1,1</row>
  </board>
</customcolors>
```

```
<specials>
  <move>
      <check cell="11,3">ispieceP</check>
      <check cell="11,3">isplayer1</check>
      <check cell="14,3">isempty</check>
      <movepiece>11,3-14,3</movepiece>
  </move>
  <move>
      <check cell="10,4">ispieceP</check>
      <check cell="10,4">isplayer1</check>
      <check cell="15,4">isempty</check>
      <movepiece>10,4-15,4</movepiece>
  </move>
  <move>
      <check cell="11,5">ispieceP</check>
      <check cell="11,5">isplayer1</check>
      <check cell="14,5">isempty</check>
      <movepiece>11,5-14,5</movepiece>
  </move>
  <move>
      <check cell="7,3">ispieceP</check>
      <check cell="7,3">isplayer2</check>
      <check cell="4,3">isempty</check>
      <movepiece>7,3-4,3</movepiece>
  </move>
  <move>
      <check cell="8,4">ispieceP</check>
      <check cell="8,4">isplayer2</check>
      <check cell="3,4">isempty</check>
      <movepiece>8,4-3,4</movepiece>
  </move>
  <move>
      <check cell="7,5">ispieceP</check>
      <check cell="7,5">isplayer2</check>
      <check cell="4,5">isempty</check>
      <movepiece>7,5-4,5</movepiece>
  </move>
</specials>
</board>
```

Ilustración 9 - Ejemplo de un tablero personalizado en XML. Entre otras cosas, destaca el uso de la pieza mostrada en la Ilustración 8 y un tablero en forma de ocho.

El sistema guarda un registro de todas las piezas y todos los tableros que intenta importar, anota todos los errores que se han cometido en el formato de estos y los escribe en un archivo para informar al usuario de los errores que ha cometido.

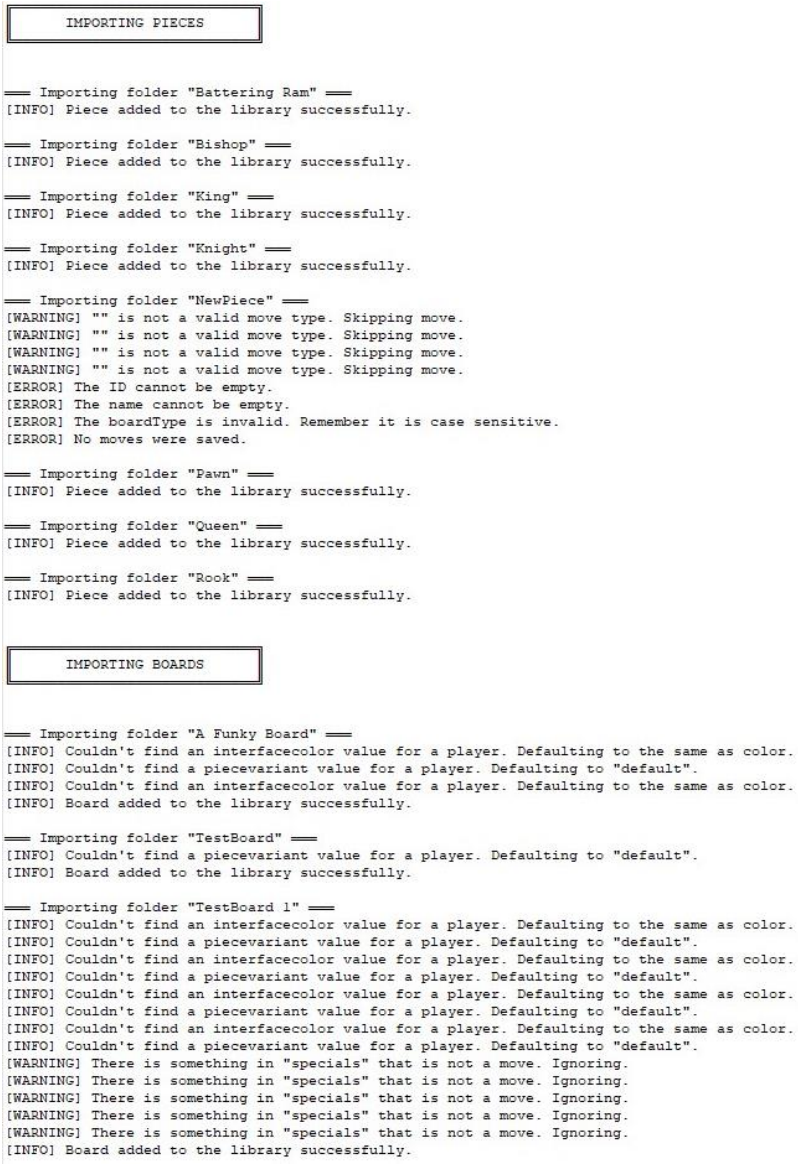

Ilustración 10 - Ejemplo de registro de errores de una ejecución del videojuego.

Todo este proceso se facilitó mucho gracias al lector de ficheros XML que viene incorporado en C#, dado que extraer y navegar la estructura del fichero se simplificaba a unas pocas líneas de código.

5.2. Sistema de librerías

Para guardar y mantener la librería de piezas y tableros, el mismo objeto que lee todos los archivos XML se encarga de mantenerlos guardados. Para ello, se le asigna una propiedad especial de Unity para que este se mantenga cargado en todo momento para poder acceder a su contenido desde cualquier escena.

Para que el usuario pueda consultar dichas librerías, se han creado dos escenas para poder consultar las piezas por una parte y los tableros por otro de una manera visual. En la escena de las piezas se puede ver una lista con todas las piezas que tiene el sistema mostrando la imagen que representa a cada pieza.

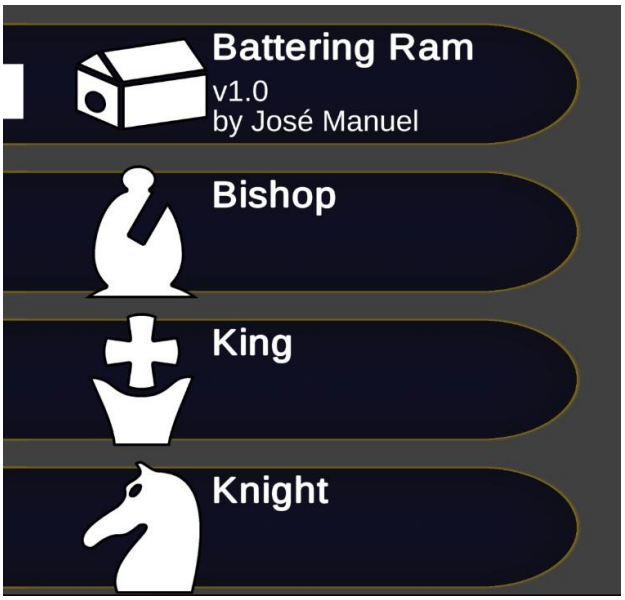

Ilustración 11 - Lista de piezas dentro del juego. Cabe destacar que las piezas que no muestran versión o autor es porque no se le ha especificado uno en el XML.

En la escena de los tableros se muestra una lista con el nombre, icono, numero de versión y autor.

Ilustración 12 - Listado de tableros disponibles. Los tableros "MiniChess" y "2v2 Chess" no tienen una imagen y, por tanto, cargan la imagen predeterminada.

Además, la escena con los tableros se utiliza como punto de entrada para crear una partida con el tablero deseado: seleccionando un tablero y presionando el botón de "jugar" se mostrará una lista con todos los jugadores involucrados en la partida, donde se podrá configurar el nombre, tiempo inicial, tiempo añadido al realizar un movimiento y tiempo de retardo al principio del turno antes de que empiece a descontar tiempo. Todos estos tiempos se configuran en segundos.

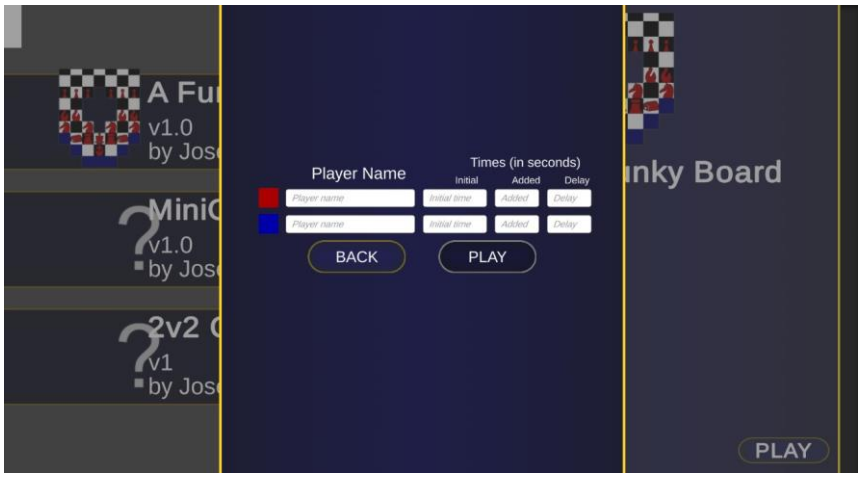

Ilustración 13 - Menú de configuración de la partida.

Si todos los tiempos son 0, se le aplica tiempo infinito al jugados. El único campo obligatorio es el de nombre, si algún tiempo está vacío, se asume tiempo de 0 segundos.

5.3. Sistema en partida

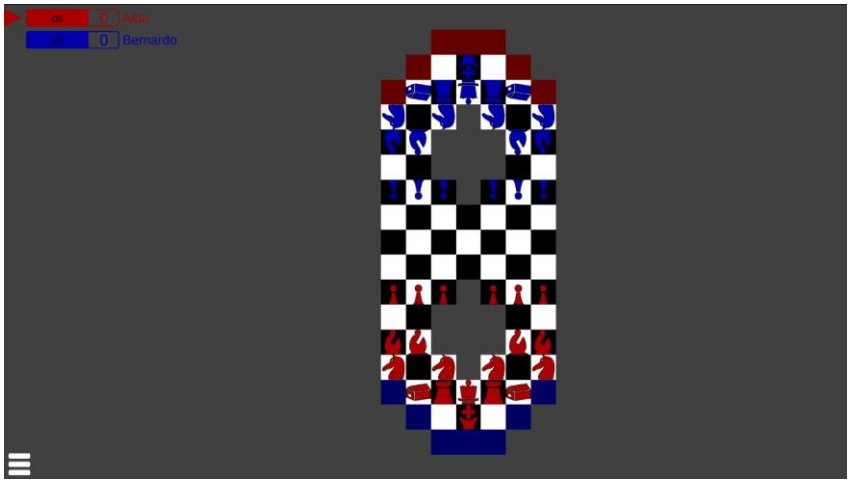

Ilustración 14 - Interfaz dentro de una partida. En este caso, el tablero cargado es el tablero descrito en el archivo XML de la Ilustración 9 en su posición inicial.

En cada subapartado se hablará de las diferentes funciones que ofrece el juego dentro de partida y cómo funcionan a nivel de código.

5.3.1. Preparación del tablero

Al iniciar una partida, el generador de tableros sigue una serie de etapas usando los datos recibidos por la librería acerca del tablero seleccionado, la primera de ellas siendo extraer toda la información referente a las piezas utilizadas, tanto de la librería de piezas para obtener su lista de movimientos, sino también la letra que representa a dicha pieza dentro de este tablero.

La segunda etapa es obtener la información de los jugadores que participan en la partida para posteriormente inicializar el controlador de turnos.

En la tercera etapa es en la que se genera el tablero. Para ello, se revisan todas las casillas en la información del tablero y se generan casillas en todas las posiciones donde no se haya especificado que no deba hacerse y piezas en todas ellas donde haya indicada el numero de un jugador y la letra de una pieza. Una vez cargado todo el tablero, se les pide a las piezas que realicen una primera lista de posibles casillas a las que se puedan desplazar.

5.3.2. Mover piezas

En partida, todas las casillas contienen una caja de colisión con la que detectan si el cursor entra o sale de estas y van informando de ello al controlador de entrada del usuario para que cuando se presione o suelte se conozca la casilla sobre la que se ha realizado la acción. Con esta información, el coordinador acepta dos tipos de movimientos: el movimiento a través de dos clics o a través del clic y arrastre. Además, todas las piezas almacenan su posición en el tablero, los movimientos que pueden realizar obtenidas de la librería de piezas y las casillas a las que se pueden mover en ese turno según estos movimientos y la posición actual del tablero. Hablaremos de cómo se calculan los movimientos disponibles en el apartado 5.3.3.

Cuando el usuario presiona el botón del ratón, se guardan las coordenadas de la casilla sobre la que ha realizado la acción y se espera a que lo suelte. En caso de que se haya presionado sobre una pieza, se le dice a esta que está siendo “arrastrada” para que se mueva con el ratón hasta que se haya soltado el clic para dar al usuario una respuesta visual en caso de que este realmente arrastrando. Cuando se suelta el clic del ratón, se le dice a la pieza que vuelva a su posición y se procede a comparar la nueva posición con la posición guardada anteriormente. En caso de que ambas posiciones sean idénticas, se considera que el usuario ha realizado un solo clic, se guardan la posición y se espera a que se realice un segundo clic, con el que se intentará realizar el movimiento. En caso de que las posiciones sean distintas, se intenta realizar el movimiento directamente y, en caso de que se haya realizado un clic con anterioridad, se borra la posición de dicho clic.

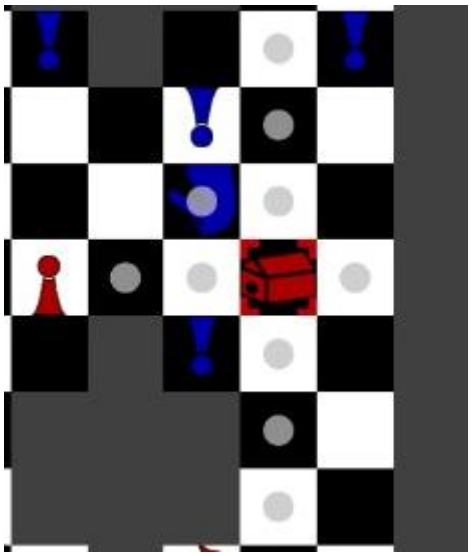

Ilustración 15 - Ejemplo de los movimientos que puede realizar la pieza descrita en la Ilustración 8 al seleccionarla.

A la hora de intentar realizar un movimiento, simplemente se comprueba si el propietario de la pieza es el mismo jugador que el que tiene el turno en ese momento y, en caso afirmativo, si la casilla objetivo está en la lista de casillas a la que se puede mover la pieza seleccionada.

5.3.3. Sistema de turnos

Al inicio de la partida, el controlador de turnos obtiene el nombre de los jugadores y los tiempos de cada jugador. Los tiempos están compuestos de tres números: el primero indicando el tiempo total que tiene dicho jugador, el segundo indicando el incremento de tiempo que reciben después de realizar un movimiento, y el tercero indicando el tiempo de retardo antes de descontar tiempo al jugador. Cabe destacar que estos números pueden ser diferentes para cada jugador, ya que es habitual que, si un jugador es notablemente superior al resto, este se ponga restricciones a sí mismo para intentar compensar, siendo la más común reducir el tiempo que tiene. Además, se le asigna el turno al primer jugador de la lista.

El tiempo del jugador va siendo descontado durante su turno hasta que el controlador principal de la partida informa al controlador de turnos de que el jugador ha realizado un movimiento, en cuyo caso se le incrementa el tiempo indicado por el tiempo de incremento del jugador y se pasa al siguiente turno. En caso de que se le acabe el tiempo, el jugador es marcado como eliminado y se pasa al siguiente turno.

Ilustración 16 - Interfaz que muestra el orden de los turnos, el turno actual y el tiempo restante de cada jugador. En este caso es el turno del jugador azul, la jugadora amarilla ha sido eliminada y el jugador verde está jugando con tiempo ilimitado.

Al pasar al siguiente turno, el coordinador realiza una serie de pasos antes de permitir al siguiente jugador poder realizar su movimiento. Estos pasos son los siguientes:

- Primero se comprueba si alguna pieza de tipo peón ha alcanzado una casilla donde puede ser promocionada a otra pieza. En caso afirmativo, se muestra una pantalla con todas las piezas a las que se puede promocionar dicha pieza para que el jugador pueda elegir a que pieza promocionar. Una vez seleccionada, el sistema elimina la pieza promocionada y crea una nueva pieza en su lugar que corresponde a la pieza seleccionada por el usuario.
- Después, se le informa al coordinador de turnos que pase al siguiente turno. Este va iterando la lista de jugadores buscando el siguiente jugador disponible en la lista. En caso de que solo haya un único jugador disponible, este es proclamado el ganador de la partida.
- En caso de que la partida no haya acabado, se calculan los movimientos que puede realizar el jugador, indiferentemente de la situación del rey o reyes del jugador después de haber realizado dicho movimiento.
- Por último, se comprueban todos los movimientos posibles del jugador y se eliminan todos aquellos que dejen al rey expuesto a ataques enemigos. Si después de eliminar todos los movimientos que dejen expuesto al rey este jugador no tiene ningún movimiento disponible, el jugador es eliminado de la partida y se pasa el turno al siguiente jugador disponible, repitiendo todos los pasos aquí mencionados.

5.3.4. Marcas y flechas

El mismo controlador de entrada del que hemos hablado en el apartado 5.3.2 también se utiliza para comprobar cuando el usuario hace clic derecho, informando al controlador principal sobre ello. Cuando el controlador principal recibe dicha información de una manera muy similar a la explicada en el apartado 5.3.2, utilizando la información de la posición del ratón en el tablero al pulsar y soltar el botón secundario del ratón para deducir si se trata de marcar una casilla en el caso de que ambos puntos sean en la misma casilla o de dibujar una flecha entre dos casillas en caso contrario. Una vez creada dicha marca o flecha, se almacena en una lista, para que, la próxima vez que se quiera marcar una casilla o dibujar una flecha, se compruebe si ya existe una con la misma posición y, si es así, borrar dicha marca o flecha en vez de dibujar una nueva por encima.

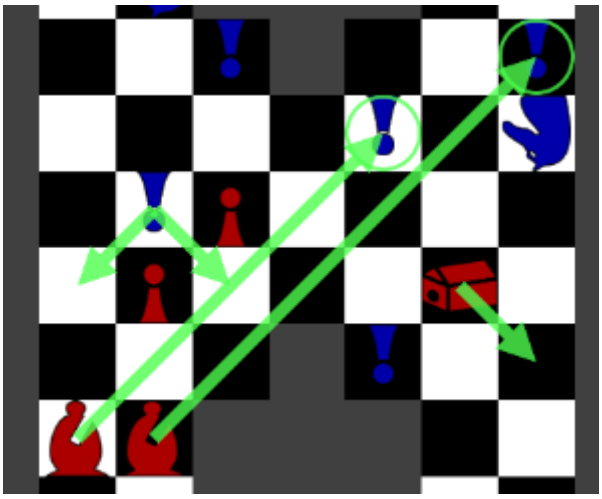

Ilustración 17 - Ejemplo del uso de las marcas y flechas. Cabe destacar que las flechas dibujadas no dependen de las piezas y que no las restringen de ninguna manera.

6. Resultados

En este apartado se compararán los resultados obtenidos con los objetivos al principio del desarrollo del proyecto, analizando si se han cumplido satisfactoriamente, si se ha cambiado durante el desarrollo o si se ha tenido que dejar para trabajos futuros.

6.1. Creación de una sintaxis para los archivos XML

Lo primero era diseñar una sintaxis para los archivos XML, haciéndolos no solo lo más intuitivos posibles, si no también resistentes a cambios y evoluciones de estos para permitir el añadido de nuevas funcionalidades. Además, estos archivos XML deben introducirse dentro del sistema de una manera sencilla, también resistente a cambios y que permita la inclusión de muchos otros archivos multimedia que permitan personalizar nuestras variantes lo máximo posible.

Resultados

Esta sintaxis ha sido creada satisfactoriamente, dividiendo los archivos XML en piezas y tableros para facilitar el desarrollo de variantes y la reutilización de código, permitiendo un grado de personalización estética de estas variantes suficiente como para dar temáticas a los tableros, y dejando espacio a la inclusión de muchas más funciones y personalización estética más profunda en un futuro.

En el anexo 10.1 se explica por completo el funcionamiento de esta sintaxis.

6.2. Leer archivos de piezas

Se necesita obtener información básica de una pieza como su nombre o su ID y un listado con los movimientos posibles desde su fichero XML, además de obtener las imágenes representativas de esta para imprimirlas en pantalla dentro de la partida. A partir de esta información, el sistema debe poder identificar a la pieza una vez se llame desde un tablero, mostrar la imagen solicitada por este como representación de la pieza dentro de la partida y poder obtener los movimientos posibles de la pieza a partir del listado de movimientos y del estado actual de la partida.

Resultados

La carga de los archivos XML y las imágenes de los archivos es satisfactoria, además de que el sistema detecta la gravedad de los errores que se pueden cometer por parte del usuario para decidir si ignorar el problema y proseguir como si esa parte errónea no existiera o si directamente cancelar la carga de la pieza.

Por otra parte, todos los tipos y estilos de movimientos descritos en el apartado 10.1.2 que se pueden introducir en el XML se han testeado por separado y en una variedad de escenarios y todos se han movido como se esperaba en todo momento.

6.3. Leer archivos de tableros

Se necesita obtener información básica de un tablero como su nombre, las piezas necesarias para jugar, los jugadores involucrados y la disposición inicial del tablero desde su fichero XML, en el que también se puede especificar los colores con los que se deben pintar las casillas e incluso la imagen que debe renderizar. Además, se debe poder cargar tanto la imagen representativa del tablero para los menús como las imágenes de las casillas que se quiere utilizar.

Resultados

La carga de los archivos XML y la imagen representativa del tablero es satisfactoria, además de que el sistema detecta la gravedad de los errores que se pueden cometer por parte del usuario y decidir si proseguir la carga como si el error no existiese o directamente cancelar la carga del tablero.

Respecto al coloreado de casillas, se ha probado las diferentes opciones que hay, que son la de dejar al sistema colorear usando los colores predeterminados, el colorear usando el mismo sistema que el predeterminado, pero con una paleta de colores personalizada y el personalizar que color de la paleta va a cada casilla, y todos pintan las casillas de la manera esperada.

Finalmente, el apartado de permitir al usuario personalizar las imágenes de las casillas no se ha implementado, por lo que se deja como trabajo posible que se puede hacer en un futuro.

6.4. Detectar errores en los archivos

A la hora de cargar los ficheros XML y las imágenes tanto de las piezas como de los tableros, el sistema debe detectar los diferentes errores que puede cometer el usuario a la hora de introducirlos e informar de estos al usuario, además de poder identificar la gravedad de estos para saber si se puede continuar con la carga de la pieza o tablero actual o no.

Resultados

Como se ha comentado en los apartados 6.2 y 6.3, el sistema detecta un gran abanico de posibles errores que el usuario pueda cometer a la hora de crear sus propias piezas y tableros e introducirlos al sistema, detectando la gravedad de estos para decidir si continuar con la carga o no. Además, el sistema recopila todos estos errores y los imprime en un fichero al que el usuario puede acceder para saber que debe corregir para asegurarse del correcto funcionamiento tanto de piezas como de tableros. Este archivo describe claramente no solo los errores cometidos por el usuario, si no también si estos errores han sido bloqueantes o no.

6.5. XML para las reglas de juego

El sistema debe cargar unos ficheros XML para modificar el reglamento el juego, permitiendo modificaciones desde cambios al sistema de turnos como a los objetivos de la partida. Estos cambios permiten crear una gran variedad de variantes incluso utilizando el mismo tablero para todas.

Resultados

Finalmente, no se ha podido implementar estos tipos de ficheros, pero se ha preparado el apartado de ficheros para tableros para permitir incluir dentro de este todo tipo de cambios en un futuro.

6.6. Asignar reglas a las piezas

El sistema debe poder asignarle una serie de reglas a las piezas que permitan modificar su comportamiento ante diferentes situaciones en el tablero como, por ejemplo, los movimientos que tiene disponibles o a que piezas puede capturar, entre otros.

Resultados

De momento, todo tipo de variación del comportamiento de las piezas viene incluido dentro del fichero XML del tablero, no de el de fichas, bajo el sistema de movimientos especiales.

Estos movimientos especiales permiten introducir una serie de movimientos resultado que se ejecuten al realizar el movimiento especial y una serie de condicionales que deben cumplirse para que estos movimientos sean posibles. Todas las condiciones se han probado de manera satisfactoria por separado, al igual que los resultados a estas.

Aun así, una ampliación de este sistema es posible, añadiendo tanto nuevos resultados a nuestros movimientos como nuevos tipos de condiciones que se deben cumplir.

6.7. Personalizar visualmente las piezas y tableros

El sistema debe permitir al usuario poder personalizar las piezas y tableros que crea con el mayor detalle posible, ya sea incluyendo aspectos diferentes a las piezas, cambiando los colores o incluso las imágenes de las casillas, o incluso cambiando el fondo de pantalla para una posible ambientación del tablero.

Resultados

Hasta el momento, el sistema permite la creación de tableros con las formas y tamaños que uno desee, algo que puede ayudar a la ambientación de los tableros si se usa correctamente, permite el uso de imágenes alternativas para las piezas, lo cual permite al usuario ambientar estas piezas al gusto reutilizando el fichero XML, y permite colorear las casillas del tablero al gusto.

Aunque estos sistemas ya permiten un grado de personalización estética aceptable, aún se pueden implementar muchos más sistemas que permitan un mayor nivel de detalles. De algunos ejemplos se habla en el apartado 8.3.

6.8. Mover piezas

El sistema debe saber interpretar la entrada del usuario para saber que movimientos quiere realizar, ejecutarlos en caso de que dicho movimiento sea posible y verse reflejados en el tablero una vez realizados.

Resultados

El sistema sabe identificar la entrada del jugador e interpretar no solo si se trata de un movimiento válido o no, sino también de si la entrada realizada se trata de un movimiento en sí o simplemente de la selección de una casilla en una primera instancia, permitiendo el movimiento de piezas a través de dos clics y no solo arrastrando. Este sistema fue probado múltiples veces durante el proyecto a medida que se iban introduciendo cambios, dado que es la base para que el resto del juego funcione, y no ha dado fallos en ninguna etapa.

6.9. Detectar movimientos legales

A la hora de calcular cuales son los movimientos los movimientos disponibles por el jugador, el sistema debe descartar aquellos movimientos que permitan al rival capturar el rey (o equivalente en el tablero actual) en su próximo turno. En caso de que, al descartar dichos movimientos, el jugador no tenga movimientos disponibles, el jugador debe ser eliminado de la partida.

Resultados

En un principio, se creía que este sistema sería uno de los más complejos de implementar, pero finalmente fue más fácil de lo esperado: una vez calculados los posibles movimientos, se simulan para comprobar si estos dejan expuesto al rey, y se eliminan de la lista de posibles movimientos en caso afirmativo. De esta manera, se cubren todo tipo de posibilidades, como evitar que el rey se mueva a casillas bajo ataque, mover cualquier pieza mientras el rey está bajo ataque si este movimiento no es para evitarlo o mover una pieza que cubría al rey de un ataque dejándolo expuesto.

6.10. Sistema de turnos

El sistema debe incluir un sistema de turnos que vaya alternando entro los diferentes jugadores activos al realizar un movimiento, además de calcular el tiempo restante que estos tienen en caso de jugar con tiempo y actualizar estos tiempos acorde al reglamento de tiempo utilizado.

Además, este sistema debe tener en cuenta los jugadores eliminados para saltarles el turno, así como eliminar a aquellos jugadores que se queden sin tiempo en su turno.

Resultados

El sistema sabe identificar correctamente de quien es el turno actual, limitando los movimientos de las piezas única y exclusivamente al jugador que tiene el turno actual, así como descontándole el tiempo que pase entre que empieza su turno y realiza un movimiento y eliminándolo en caso de que se le acabe el tiempo, pasando así al siguiente jugador. Al acabar el turno, el sistema busca satisfactoriamente quien es el próximo jugador que debe jugar, ignorando a los jugadores que ya se encuentren eliminados.

6.11. Dibujar flechas y marcar casillas

El sistema debe permitir al jugador dibujar sobre el tablero dibujar flechas y marcar casillas para poder indicar posibles jugadas, así como poder compartir ideas de movimientos con los compañeros de equipo en el caso de partidas de equipo. Estas marcas deben ser independientes a la disposición actual del tablero.

Resultados

El sistema puede dibujar satisfactoriamente una serie de flechas y marcas que sirvan al usuario como soporte visual al análisis que realice. Además, el sistema guarda un listado de estas flechas y marcas para que, en el caso de que el usuario intente volver a dibujar una flecha o marca ya existente, en vez de eso la elimine.

Dado que aún no se han implementado ningún sistema de juego en línea, no se ha podido implementar el poder compartir estas flechas y marcas con los compañeros de equipo.

7. Conclusiones

El trabajo ha sido una carga de trabajo constante pero no excesiva. A medida que se iban diseñando y desarrollando funcionalidades se iban encontrando nuevas necesidades o ideas que implementar en el juego o se iban descubriendo que algunas funciones no se podrían implementar tanto por falta de tiempo o por falta de experiencia con las herramientas. Esto se ha facilitado y reforzado gracias a la herramienta Trello, que ayudaba a organizar esas ideas, detallarlas y ordenarlas por el estado en el desarrollo de estas. El rol que ha tenido esta herramienta en el desarrollo se puede considerar incluso imprescindible en algunas etapas.

Respecto al resultado final del videojuego, solo se puede decir que, aunque no es el que estaba planificado, se está satisfecho con el alcance obtenido. La decisión de no dejarse presionar por las fechas ha permitido realizar un desarrollo sosegado de la aplicación siempre teniendo en mente las posibles ampliaciones que se le pueden aplicar al código e implementado de manera que estas ampliaciones supongan el menor esfuerzo posible. A esto ha ayudado el tener siempre en mente una idea de cómo podría o debería quedar la aplicación final, viendo con mas claridad las posibles ampliaciones al código que se está desarrollando.

Además, este desarrollo ha permitido familiarizarse aun más con la herramienta Unity y el gran abanico de posibilidades que esta ofrece, además de explorar los métodos y estructuras utilizadas en los videojuegos.

7.1. Relación con los estudios cursados

Este trabajo ha tenido una gran cantidad de puntos relacionados con lo estudiado en la carrera. Por ejemplo, en este se ha podido aplicar algunos patrones estudiados en la carrera, como por ejemplo el patrón de singleton para el sistema de librerías, y también se han aplicado los conocimientos obtenidos acerca de la organización y planificación de proyectos. Además, se han podido aplicar en mayor profundidad los conocimientos obtenidos en las asignaturas optativas referentes al diseño y desarrollo de videojuegos.

8. Trabajos futuros

A partir de esta versión inicial del videojuego se puede extender o mejorar el funcionamiento de muchas formas distintas. En este apartado se procederá a describir una lista de algunas ampliaciones que se pueden realizar al videojuego y como se pueden aplicar. Estas ampliaciones serán divididas en apartados en relación con el tipo de ampliación, pero cabe recordar que el orden en el que se listan no implica ningún orden de importancia ni prioridad.

8.1. Modificaciones del reglamento

Las modificaciones del reglamento son inclusiones de reglas opcionales que modifican reglamentos como nuevas condiciones de victoria o cambios en el funcionamiento de los turnos entre otros. Todos estos cambios deberían ser incluidos como entradas en la lista de metadatos del tablero donde se quiera aplicar dicho reglamento.

8.1.1. Condiciones de victoria

La primera de las múltiples modificaciones que se pueden hacer al reglamento es el a las condiciones de victoria. Este tipo de modificaciones deberían ser un campo nuevo en los metadatos de los tableros que indiquen que condición de victoria se está usando.

Por ejemplo, una de las condiciones de victoria que se pueden incluir es la usada en la variante de Antichess, donde para ganar se debe perder todas las piezas, pero estás forzado a capturar si se puede. Esto implicaría crear algoritmos alternativos tanto en la función que calcula los movimientos disponibles para forzar al jugador a capturar si es posible como en la función que calcula cuando un jugador es eliminado para darle la victoria este jugador en vez de eliminarlo.

Otro ejemplo, algo más sencillo que el anterior, es que elimine la necesidad de defender al rey o pieza equivalente. En este caso solo habría que indicarle al coordinador que no compruebe que movimientos dejan expuesto al rey.

8.1.2. Formato de los turnos

Otra posible modificación al reglamento de los tableros es modificar el funcionamiento de los turnos. En este caso, las modificaciones pueden ir desde simplemente indicar cuantos movimientos se pueden realizar por turno, pasando por alterar el orden de estos o incluso una combinación de ambas detallando que se puede hacer en cada uno de los turnos.

Por parte de configuración en el XML, dependiendo de la complejidad de lo que se quiera implementar, estos cambios pueden ir desde una simple entrada en el listado de metadatos hasta una entrada entera de manera similar al sistema de coloreado de casillas.

Dentro del juego simplemente se debería modificar la función del coordinador en partida que controla el paso de los turnos para adaptarse a las diferentes configuraciones que se quiera soportar.

8.1.3. Otras modificaciones menores

Algunas de las modificaciones que se pueden incorporar se podrían hacer a nivel de juego y no de XML.

Un ejemplo de esto es que se puede incluir un sistema de niebla de guerra, donde solo sean visibles las casillas a las que se puede mover al menos una de las piezas del jugador. Para hacer un cambio así se debería incluir un sistema para ocultar las casillas a las que no se pueda mover ninguna pieza una vez calculados los movimientos posibles.

Otro ejemplo sería la implementación de un sistema que, al empezar la partida, todas las piezas que no tengan el tipo peón sean colocadas de manera aleatoria entre las casillas que ocupan de manera habitual, similar al ajedrez 960 del que hablamos en el apartado 2.2.2. En este caso, dentro del generador de tableros, simplemente se deberían guardar las casillas donde deberían aparecer y la cantidad de piezas que aparecen de cada tipo y, una vez finalizada la creación del tablero y del resto de las piezas, colocar las piezas en el tablero de manera aleatoria.

Otro ejemplo más puede ser incluir la posibilidad de colocar las piezas capturadas de nuevo en el tablero bajo el control del jugador que las ha capturado, como si se tratase del ajedrez Crazyhouse del que se habló en el apartado 2.2.1. Aquí solo sería necesario llevar un seguimiento de las piezas capturadas e implementar una interfaz que permita al jugador colocar de nuevo estas piezas en el tablero, llamando al generador de tableros para que coloque a las nuevas piezas.

Nótese que estas modificaciones no solo no necesitarían de ninguna modificación por parte del usuario en los archivos XML, si no que serían compatibles entre ellos, por lo que se podría incluir una serie de botones para activar y desactivar estas modificaciones desde la pantalla de configuración de la partida.

8.2. Nuevos formatos de tablero

En este trabajo solo se han implementado los tableros en dos dimensiones de casillas cuadradas, pero se ha implementado de manera que poder soportar otros tipos de tableros sea lo más fácil posible y reutilizando la mayor cantidad de código posible.

8.2.1. Nuevas formas de casillas

Como se ha comentado, una de las modificaciones que se pueden hacer al tablero es cambiarle la forma de las casillas.

La forma más sencilla de modificar las casillas es haciendo que todas las casillas del tablero utilicen la misma forma. La única restricción que implica esto es que estas casillas deben poder encajar entre ellas, y las únicas formas geométricas que consiguen esto a parte de los cuadrados son los triángulos y los hexágonos. Ambas formas ya se

han explorado para crear variantes del ajedrez como se ha mencionado en el apartado 1, y en ambos casos se ha necesitado reimaginar como se mueven las piezas del juego, dado que, sin contar diagonales, las casillas cuadradas admiten 2 ejes de movimiento para un total de 4 sentidos y dos direcciones, mientras que tanto las hexagonales como las triangulares admiten 3 ejes o direcciones para un total de 6 sentidos.

Aun así, para poder incorporar estos nuevos tipos de casillas en el juego se puede reutilizar todo el código referente a como se guarda el tablero internamente, pero requeriría cambiar todo el código referente a como se mueven las piezas, y dichos cambios requerirían cambios hasta en como el usuario los introduce en el XML. Por ejemplo, en el caso de los tableros hexagonales, aunque los tableros solo necesitan dos ejes, los movimientos de piezas necesitarían 3, uno para el vertical (u horizontal, dependiendo de la orientación de los hexágonos) y los otros dos para los dos ejes diagonales. El trabajo más extenso sería la parte de como mostrar en pantalla estos nuevos tableros.

8.2.2. Nuevas topologías y conexiones hiperdimensionales

Otra opción es la de conectar desde casillas individuales hasta laterales enteros del tablero entre ellas. Esta modificación implicaría que una pieza pueda moverse entre estas piezas como si fueran contiguas entre ellas y con la posibilidad de especificar direccionalidad, pudiendo especificar desde caminos de una sola dirección hasta portales entre casillas, dando la posibilidad de simular tableros que un tablero plano no permitiría, como por ejemplo un tablero 2d con forma de cilindro, donde los bordes izquierdo y derecho se conectan entre ellos.

Este cambio implicaría una nueva sección dentro de los archivos XML de tableros para poder definir estas conexiones entre casillas, con los cambios en el sistema de lectura de los XML de tableros y su posterior almacenamiento en la librería de piezas. Además, se debe modificar el algoritmo de cálculo de movimientos para tener en cuenta estas conexiones de casillas y analizarlas.

Este cambio puede también abrir la posibilidad a mezclar varias formas de casillas distintas y conectarlas entre ellas de diferentes maneras. En este caso existen dos opciones para implementar estos tipos de tableros.

Una de las opciones es de dejar que los usuarios aprovechen tanto las inclusiones descritas en este apartado como el uso de imágenes de casilla personalizadas descrito en el apartado 8.3.1 para, aunque siga usando un tablero estándar, parezca que se están juntando casillas de diferentes tipos.

Otra opción es implementar un sistema que permita la mezcla de tipos de casillas desde el propio XML. El problema es que esta aproximación requeriría mucho trabajo en el diseño de estos XML, y los cambios que implicaría esto a nivel de código depende tanto de el qué se implemente como de cómo se implemente exactamente.

8.2.3. Mayor número de dimensiones

La otra posibilidad para cambiar el tipo de tablero es cambiando el número de dimensiones de este. Esta modificación no es nada nueva, ya se han visto variantes de tres dimensiones e incluso de cuatro dimensiones.

En este caso, se puede reutilizar el código ya existente y simplemente añadir un eje extra tanto a la representación del tablero en sí como a la representación de los movimientos de las piezas. Además, este cambio no tendría un límite de por sí a cuantas veces se puede aplicar, haciendo el número de dimensiones a las que se pueden llegar teóricamente infinito.

8.3. Cambios y opciones estéticas

La personalización de una variante no solo se trata de las reglas o de la forma del tablero, también pasa por cómo se ve ese tablero. Hasta ahora, los usuarios pueden modificar tanto las imágenes que representan a las piezas y los colores de las casillas del tablero, pero aún se pueden incluir muchas más opciones para el usuario a la hora de personalizar su tablero visualmente lo máximo posible. En este apartado se plantean algunas ideas para ampliar el repertorio de herramientas que tienen los usuarios.

8.3.1. Permitir imágenes personalizadas por tablero

Una de las posibilidades es la de permitir usar imágenes tanto para decorar las casillas como para modificar el fondo dentro de partida.

En el caso de cambiar el fondo, simplemente sería importar una imagen de la misma manera que lo hacemos con el icono del tablero y lo cargamos en el fondo de la escena de la partida, ya sea poniendo una imagen en la parte de atrás de la escena o modificando el fondo que renderiza la cámara con la imagen cargada.

El caso de las casillas es algo más complejo. En un principio, el funcionamiento debería ser el mismo que con el coloreado de casillas que hay ahora mismo, pero con imágenes en vez de colores y cambiando la imagen predeterminada que se usa para las casillas con la nueva imagen o incluso modificando la función que crea las casillas para crearlas todas indiferentemente de si el tablero tiene marcadas dichas casillas como inexistentes o no, y simplemente deshabilitando los algoritmos que detectan la posición del ratón para admitir casillas como muros. En este caso, habría que diseñar como el usuario introduce las imágenes dentro del sistema, que restricciones tienen o como indica el usuario en el XML que imágenes usar en cada casilla, entre otros.

8.3.2. Permitir uso de modelos 3D como piezas

Otra opción que se le puede ofrecer al usuario para la personalización de sus variantes es el admitir el uso de modelos 3D para las piezas.

Aunque leer y almacenar el modelo es similar a como se lee y almacena las imágenes alternativas usando un formato de archivo de modelo compatible, el poder utilizar estos modelos implica cambiar por completo como se muestran en pantalla los tableros para admitir estos modelos y, con ello, como se detectan las entradas del jugador. Además, sería necesario especificar en el tablero si se debe renderizar en 2D o 3D para saber si usar las imágenes o los modelos.

8.4. Nuevos sistemas y calidad de vida

No todas las modificaciones que se pueden realizar se restringen a mecánicas del juego, algunas modificaciones posibles se aplican a otros elementos del videojuego, en algunos casos en relación indirecta con el juego, en otros simplemente en los menús y en algunos casos solo a los XML. A continuación, se propone unas pocas ideas de las muchas modificaciones que se pueden aplicar, tanto dentro del juego como incluso fuera de él.

8.4.1. Multijugador en línea

La primera opción, y probablemente la más importante, es la inclusión de un sistema multijugador en línea usando las librerías que para ello ofrece Unity.

Por la parte del controlador en partida solo sería necesario limitar los jugadores que el usuario puede mover únicamente a los que le representan en la partida, además de incluir algunas funciones para poder comunicar al rival el movimiento realizado y recibir dicha información para reflejarla en el tablero.

Por otra parte, sería necesario crear una serie de menús para la búsqueda de partidas en línea y la configuración de estas, aparte de diseñar no solo cómo se comunican los diferentes jugadores, si no como controlar las posibles desconexiones y desincronizaciones que pueden suceder.

8.4.2. Soporte para nuevas entradas

Otra opción que puede ser prioritaria es la inclusión de nuevos esquemas de controles que permitan al usuario jugar al videojuego con la mayor variedad de controles posible. Por el momento solo se ha implementado la entrada por ratón, pero debería ser posible, por ejemplo, la entrada por teclado, la cual te permita mover un cursor entre las casillas usando las flechas de dirección.

Esto se consigue creando dentro del controlador de entradas del usuario una serie de sistemas que detecten estas entradas de teclado o cualquier otro tipo de entrada que se quiera implementar. En el caso de la entrada por teclado, se puede crear un cursor virtual que se mueva entre las casillas respondiendo a las entradas del usuario y que utilice las mismas funciones que con el ratón cuando el usuario quiera seleccionar o deseleccionar una casilla.

8.4.3. Ampliación del sistema de movimientos especiales

Dentro de los ficheros XML de tableros, existen una serie de movimientos llamados movimientos especiales, que son movimientos que solo se pueden realizar en caso de que se cumplan una serie de requisitos. (Más información en el apartado 10.1.3)

Aunque el sistema actual ya permita una gran variedad de posibilidades con las opciones actuales, siempre se puede ampliar estas opciones con nuevas condiciones para perfilar aún más bajo que circunstancias se pueden realizar estos movimientos como con nuevos resultados a estos movimientos.

Para poder incluir nuevas posibilidades, se deben realizar cambios en el lector de XML de tableros para poder cargar estas nuevas posibilidades, como el sistema de movimientos especiales para implementarlas.

8.4.4. Oponentes con inteligencia artificial

Otra opción es la inclusión de una inteligencia artificial que pueda analizar el tablero, los movimientos disponibles y realizar movimientos sin la necesidad de que sean introducidos por el usuario.

A parte de la adicción de los algoritmos necesarios para la creación de esta inteligencia artificial, las únicas modificaciones necesarias serían al coordinador para que este pueda comunicar el estado completo del tablero, los movimientos posibles y que la inteligencia artificial pueda mover piezas a nivel de código.

8.4.5. Detalles de una pieza o tablero

En los listados de piezas y de tableros se han dejado preparadas unas interfaces para poder mostrar detalles de estos para que el usuario pueda conocer los detalles de estos sin necesidad de conocer el funcionamiento de los XML ni entrar dentro del archivo.

En este caso solo se debería diseñar la interfaz con la información que se quiere mostrar al usuario y conectar esta interfaz con el controlador de la librería. El sistema que detecta de que pieza o tablero se quiere conocer información ya está implementado y, en caso de que se quiera cambiar como se listan las piezas o tableros, se puede reutilizar la función que manda la pieza seleccionada a la interfaz.

8.4.6. Editor gráfico para XMLs

Un editor gráfico para crear piezas y tableros sería un trabajo que facilitaría a los usuarios menos experimentados con el uso de archivos XML el poder crear sus propias piezas y tableros personalizados sin necesidad de conocer el funcionamiento de estos.

La idea sería desarrollar un programa que permita crear piezas y tableros de manera visual y con herramientas que te permitan, entre otras muchas cosas, ver en tiempo real los cambios realizados y como afectan al resultado final.

Aunque lo ideal sería que este editor viniese incorporado dentro del propio videojuego para contener todas las funcionalidades dentro del propio juego, al tratarse de un programa que genera archivos XML, realmente no es obligatorio que así sea, ya que se podría crear una interfaz gráfica con cualquier otro lenguaje y que el archivo XML que este programa genere traspasarlo a la carpeta correspondiente.

9. Referencias

1. **Williams, Andrew.** *History of Digital Games: developments in art, design and interaction.* Boca Raton : Taylor & Francis Group, 2017. ISBN 9781138885530.

2. **Lowood, Henry.** Game Engines and Game History. *History of Games International Conference Proceedings.* Enero de 2014. ISSN 1916-985X.

3. **Bird, H. E.** *Chess History and Reminiscences.* 10ª Edición. s.l. : Project Gutenberg, 2004.

4. **Rogers, Scott.** *Level Up! : the guide to great video game design.* Segunda edición. Chichester : Wiley, 2014. ISBN 1118877217.

10. Anexos

10.1. Estructura de los archivos XML

En este anexo se explicará en profundidad la estructura de archivos necesaria para poder crear tus propias piezas, utilizarlas en tus propios tableros personalizados y poder jugar a estos tableros dentro del videojuego, además de explicar la estructura de los archivos XML para describir las posibilidades que ofrecen.

10.1.1. Estructura de Archivos

En la raíz de los archivos del videojuego se encuentra una carpeta llamada “UserContent” y, dentro de esta, hay dos carpetas, una para piezas y otras para tableros. Dentro de cada carpeta se deberían crear subcarpetas, una para cada pieza o tablero. Estas subcarpetas pueden tener cualquier nombre y no afectan en ningún aspecto en las piezas o tableros que contienen. Las únicas excepciones de nombres de carpetas son “TemplatePiece” en el caso de las piezas y “TemplateBoard” en el caso de los tableros, ya que estas carpetas contienen plantillas que usar como punto de partida.

Dentro de una carpeta de una pieza son necesarios por lo menos dos archivos:

- **piece.xml**
  Este archivo es el que va a contener toda la información interna de la pieza.
- **default.png**
  Este archivo es la imagen que va a representar la pieza dentro del juego.
  Uno de sus lados debe ser exactamente 500 píxeles y el otro debe ser de 500 píxeles o menos.

Además de estos archivos, se pueden incluir cualquier cantidad de archivos PNG con imágenes alternativas a estas piezas. Estas imágenes se pueden usar a petición de los tableros para sustituirlas. Estas imágenes tienen las mismas restricciones que el archivo "default.png" respecto a el tamaño en píxeles.

Por otra parte, dentro de una carpeta de un tablero es necesario un archivo "board.xml" que contendrá toda la información relativa al tablero. Además, se puede incluir un archivo llamado "icon.png" como icono que represente ese tablero. Esta imagen debe ser de exactamente 1280 píxeles de ancho por 720 de alto.

10.1.2. Piezas

El archivo XML de las piezas debe estar todo dentro de un elemento <piece> y dentro definir dos secciones con sus correspondientes elementos, un elemento <metadata> y un elemento <moves> para definir los metadatos de la pieza y los movimientos que puede realizar respectivamente.

En los metadatos deberemos poner una serie de elementos para definir información acerca de la pieza. Para cada dato deberemos definir un elemento con el nombre del campo. Los campos que se pueden o deben definir son los siguientes:

- **id**
  Identificador de la pieza dentro del sistema. Este identificador será el que se utilicen en los tableros para indicar que pieza usar. Este campo es obligatorio, puede ser cualquier cadena de caracteres, y debe ser único entre todas las piezas.
- **name**
  Nombre representativo de la pieza que se mostrará al usuario en el listado de piezas. Este campo es obligatorio y puede ser cualquier cadena de caracteres.
- **version**
  Versión de la pieza que se mostrará al usuario. Este campo es opcional y puede ser cualquier cadena de caracteres.
- **author**
  Nombre del autor de la pieza. Este campo es opcional y puede ser cualquier cadena de caracteres.
- **boardType**
  Sobre que tipo de tablero se ha diseñado la pieza. Este campo es importante ya que, aunque un set de movimientos contenga el mismo número de coordenadas en varios tipos de tablero distintos, estas coordenadas no significarían lo mismo. Este campo es obligatorio, y actualmente solo soporta la entrada

"Square2D" para los tableros de dos dimensiones con casillas cuadradas, pero a medida que se incluya soporte para nuevos tipos de tableros, este campo permitirá más entradas distintas para estos nuevos tipos de tablero.

En la sección de <moves> es donde listaremos todos los movimientos posibles que puede realizar la pieza. Cabe recordar que todos los movimientos que requieran que se cumplan una condición, como por ejemplo el doble movimiento del peón, no van aquí, si no en el XML del tablero.

Los movimientos que se pueden definir entran dentro de dos categorizaciones, dependiendo del estilo en el que se mueven y del tipo de movimiento que son.

El estilo define si se trata de un salto o de un movimiento, y de si este movimiento se realiza solo una vez o se puede repetir. Los cuatro valores que se pueden asignar son los siguientes:

- **infinite**
  El movimiento necesita que el camino entre el origen y el destino esté libre y podrá ser repetido hasta que este quede bloqueado. Ejemplo: La torre o el álfil.
- **finite**
  El movimiento necesita que el camino entre el origen y el destino esté libre y solo podrá ser realizado una vez. Ejemplo: Si el peón pudiese moverse dos casillas sin necesidad de ser su primer movimiento.
- **jump**
  El movimiento ignora si el camino está bloqueado o no y solo se realizará una vez. Ejemplo: El caballo.
- **infinitejump**
  El movimiento ignora si el camino está bloqueado o no y se podrá repetir hasta que el destino esté bloqueado. Ejemplo: Si el caballo pudiese repetir sus movimientos indefinidamente mientras no cambie de dirección.

Y los tipos de movimiento indican si el movimiento se puede realizar solo capturando, solo sin capturar o de manera indiferente. Los tres valores que se pueden asignar al tipo de movimiento son los siguientes:

- **move:** solo puede moverse sin capturar.
- **capture:** solo puede moverse capturando.
- **both:** se podrá mover independientemente de si es capturando o no.

Para definir un movimiento, se deberá añadir dentro del elemento <moves> un elemento con el nombre del estilo de movimiento que se quiere y añadiéndole un atributo de nombre "type" y con valor el tipo de movimiento. El movimiento se indica dentro de este elemento con un set de coordenadas que indica cuantas casillas se debe mover en cada eje de manera relativa a la posición actual de la pieza. La dirección que representa cada eje de coordenadas variaría entre los distintos tipos de tableros, pero se debería mantener una regla general con las siguientes premisas:

- El orden de los ejes es siempre empezando por el que indique la última dimensión de todas, seguida de la penúltima y así hasta que solo quede un plano. Las siguientes coordenadas son primero la que indique la dirección mas hacia arriba posible y siguiendo las agujas del reloj.

- El sentido positivo de un eje siempre apunta lo mas hacia arriba, adelante y a la derecha posible.

Una restricción a tener en cuenta es que si se crea un movimiento de estilo “infinite” o “infinitejump” y de tipo “capture” funcionarán de la misma manera que si se hubieran creado de estilo “finite” o “jump” respectivamente.

Un ejemplo de cómo se incluiría el peón en un tablero bidimensional de casillas cuadradas es el siguiente:

```
<moves>
      <finite type=”move”> 1,0 </finite>
      <finite type=”capture”> 1,1 </finite>
      <finite type=”capture”> 1,-1 </finite>
<moves>
```

10.1.3. Tableros

Al igual que las piezas, todos los datos de los tableros deben estar incluidos dentro de un elemento <board>. Dentro se deben definit los componentes <metadata>, <players>, <pieces>, <position> y <specials>, además de poder definir un componente adicional con el nombre <customcolors>.

En los metadatos, al igual que en las piezas, tendremos que poner una serie de datos sobre el tablero. Para cada dato deberemos definir un elemento con el nombre del campo. Los campos que se deben o pueden definir son los siguientes:

- **name**
  El nombre con el que se identifica al tablero. Este campo es obligatorio, puede ser cualquier cadena de caracteres y debe ser único entre todos los tableros.
- **version**
  Versión del tablero que se mostrará al usuario. Este campo es opcional y puede ser cualquier cadena de caracteres.
- **author**
  Nombre del autor del tablero. Este campo es opcional y puede ser cualquier cadena de caracteres.
- **boardType**
  El tipo de tablero. Este campo es importante para que el sistema sepa que funciones debe usar para crear el tablero y que piezas son compatibles con el tablero. Este campo es obligatorio, y actualmente solo soporta la entrada “Square2D” para los tableros de dos dimensiones con casillas cuadradas, pero a medida que se incluya soporte para nuevos tipos de tableros, este campo permitirá más entradas distintas para estos nuevos tipos de tablero.

Dentro del componente de jugadores, se debe introducir una lista con un elemento <player> por cada jugador que se quiera crear en partida. Actualmente existe una restricción de 9 jugadores máximos por tablero. El orden en los que sean introducidos será el mismo orden en los que se turnaran dentro de la partida. Dentro de cada elemento <player> introduciremos una serie de elementos con los que definiremos algunos detalles sobre estos jugadores. Los campos que se deben o pueden definir son los siguientes:

- **team**
  Este es el equipo del jugador. Todos los jugadores con el mismo equipo no podrán capturarse las piezas mutuamente, no amenazarán a los reyes de los demás y ganarán de manera conjunta, incluso si uno de ellos fue eliminado. Este campo es obligatorio y solo se pueden introducir valores numéricos. También se puede introducir un guion (-) como valor, lo cual hará que los jugadores no tengan equipo.
- **color**
  Este es el color en el que se tintarán las piezas del jugador. Este campo es obligatorio y se debe introducir un color en formato RGB, separando cada valor con una coma (,).
- **interfacecolor**
  Este es el color de todos los elementos de la interfaz que representan al jugador. Este campo es útil cuando las imágenes de las piezas están pensadas para no ser coloreadas, ya que puedes poner el valor de color a blanco y usar este campo para dar un color representativo al jugador. Este campo es opcional y se debe introducir un color en formato RGB, separando cada valor con una coma (,). En caso de que no se introduzca ningún valor, se usará el mismo valor que el introducido en color.
- **direction**
  La dirección a la que se mueven las piezas del jugador. Esto permite hacer que las piezas se muevan a la dirección deseada sin necesidad de crear nuevas piezas que tengan en cuenta dichas direcciones. Este campo es obligatorio, y debe ser un más (+) o un menos (-) seguido de un número. El número representa el eje por el que se mueven las piezas del jugador, y el símbolo usado indica si se mueve hacia adelante o hacia atrás en dicho eje. En caso de que no se introduzca ningún símbolo, se supone un más (+).
- **imagevariant**
  La imagen alternativa que utilizarán las piezas del jugador. Aquí se debería introducir el nombre de la imagen PNG adicional que se incluyó en la carpeta de las piezas. Este campo es opcional y puede ser cualquier cadena de caracteres. En caso de que no se utilice este campo o una pieza no tenga un PNG con el nombre indicado, se utilizará la imagen predeterminada de esta.

En el componente de piezas, se debe crear un elemento <piece> por cada pieza que se quiera utilizar en el tablero y, al igual que con los jugadores, debemos introducir una serie de elementos dentro de cada <piece> para describir a esa pieza que se quiere usar. Los campos que se deben definir son los siguientes:

- **id**
  El id de la pieza que se quiere usar. El campo es obligatorio, puede ser cualquier cadena de caracteres y debe coincidir con el campo id de los metadatos de la pieza que quiere usarse en el tablero.
- **value**
  El valor de la pieza. Actualmente, simplemente es un valor simbólico y solo se utiliza para mostrar la ventaja que tienen los jugadores sobre el jugador con menor valor, pero en un futuro se puede utilizar para condiciones de victoria alternativas. El campo es obligatorio, y debe ser un número positivo.
- **char**

Un carácter que represente a esta pieza dentro de este tablero. Puede usarse cualquier carácter a excepción de números, una barra baja (_) o un doble punto (:), aunque se recomienda que se utilicen caracteres del alfabeto. Debe ser único entre las piezas utilizadas en el tablero.

Además, a cada elemento <piece> se le puede agregar un atributo llamado “type” con uno de los siguientes valores:

- **king**
  Este valor indica que la pieza es el objetivo que capturar en la partida. Debe haber una pieza y solo una pieza que contenga este tipo. Un jugador puede tener en el tablero varias copias de una pieza tipo rey, pero con solo perder una de esas piezas el jugador es eliminado.
- **pawn**
  Si la pieza puede ser promovida a otra pieza al alcanzar una casilla de promoción del jugador. Puede haber cualquier cantidad de piezas con este tipo.
- **upgrade**
  Si la pieza se puede obtener promoviendo una pieza de tipo “pawn”. Puede haber cualquier cantidad de piezas con este tipo.

Dentro del componente <position> se debe describir el tablero en su posición inicial. El cómo se debe definir el tablero dependerá de cada tipo de tablero, pero en este documento se describirá como se debe rellenar el componente <position> para un tablero de dos dimensiones y casillas cuadradas ya que es el único implementado en la actualidad.

Dentro de <position> se debe crear un elemento <row> por cada línea que queramos poner en el tablero. Cada línea se debe escribir siguiendo estas instrucciones:

- Debes separar cada casilla de la línea con una coma (,).
- Para indicar una casilla vacía, hay que poner un cero.
- Para indicar que una casilla no existe, hay que poner una barra baja (_).
- Para indicar que una casilla contiene una pieza, hay que poner primero el número del jugador que controlará esta pieza, siendo 1 el primer jugador de la lista de jugadores en el elemento <players>, seguido del carácter representativo de la pieza que se quiere usar.
- Para indicar que una casilla es la casilla de promoción de un jugador, después de definir el contenido de esa casilla, se debe poner un doble punto (:) seguido del número del jugador que puede promover piezas en esa casilla. Se puede repetir el proceso para añadir varios jugadores en la misma casilla.

Un ejemplo de cómo implementar la línea posterior del jugador negro en el ajedrez clásico sería el siguiente:

```
<position>
	<row>2R:1,2N:1,2B:1,2Q:1,2K:1,2B:1,2N:1,2R:1</row>
</position>
```

Se debe comprobar que todas las líneas contengan exactamente la misma cantidad de casillas o el tablero no cargará.

En caso de incorporar un elemento <customcolors>, existen dos maneras de utilizarlo: se puede incluir una lista de colores que sustituya la lista de colores estándar del sistema para colorear las casillas o se puede incluir, a parte de la lista de colores, un tablero donde se indique cada casilla de qué color se debe pintar.

Para introducir una lista de colores, dentro del elemento <customcolors> hay que incluir un elemento <colorlist> y, dentro de este último, un listado de elementos <color> con los colores que se quieren utilizar. Estos colores se den introducir en formato RGB separando cada valor con una coma (,).

En caso de querer indicar el color de cada casilla individualmente, se debe crear un elemento <board> dentro del elemento <customcolors> donde se debe introducir un tablero de manera similar al elemento <position>, pero cada casilla debe contener un número indicando que color de la lista de colores del elemento <colorlist> se debe usar, siendo 1 el primer color de la lista.

En el elemento de <specials> se incluirá un listado con todos los movimientos especiales del tablero. Un movimiento especial es todo aquel movimiento que requiera de que se cumpla una serie de requisitos para poderse realizar. Por cada movimiento especial que se quiera crear, se debe introducir un elemento <move> y dentro de este elemento una serie de elementos indicando tanto las condiciones que se deben cumplir para poder realizar el movimiento especial como los resultados de dichos movimientos especiales.

Los elementos condicionales sirven para describir qué condiciones se deben cumplir para poder realizar el movimiento. Debe existir al menos un elemento condicional para poder realizar un movimiento especial, y todas las condiciones se deben cumplir para que el movimiento esté disponible. Los posibles elementos condicionales son los siguientes:

- **check**
  Este elemento comprueba que se cumpla una condicional dentro del estado actual del tablero. Un elemento <check> siempre debe contener un atributo "cell" indicando la casilla sobre la que se quiere comprobar esta condición. Dentro de este elemento, se debe describir que comprobación se quiere realizar. Las comprobaciones que se pueden hacer son las siguientes:
    - hasmoved: Comprueba si la pieza en la casilla indicada se ha movido en algún momento de la partida. Si la casilla está vacía, esta comprobación será falsa.
    - hasnotmoved: Comprueba si la pieza en la casilla indicada no se ha movido en ningún momento. Si la casilla está vacía, esta comprobación será falsa.
    - ispiece**P**: Comprueba si la pieza en la casilla indicada es la pieza indicada. Se debe cambiar "P" por el carácter de la pieza que quiera comprobar. Si la casilla está vacía, esta comprobación será falsa.
    - isplayer**P**: Comprueba si la pieza en la casilla indicada es propiedad del jugador indicado. Se debe cambiar "P" por el número del jugador que se quiera comprobar. Si la casilla está vacía, esta comprobación será falsa.
    - isteam**T**: Comprueba si la pieza en la casilla indicada es propiedad de uno de los jugadores del equipo indicado. Se debe cambiar "T" por el

número del equipo que se quiera comprobar. Si la casilla está vacía, esta comprobación será falsa.
    - isempty: Comprueba si la casilla indicada está vacía.
    - isnotempty: Comprueba si la casilla indicada contiene alguna pieza.
    - isattacked: Comprueba si la casilla indicada está siendo atacada por alguna pieza enemiga al jugador actual.
    - issafe: Comprueba si la casilla indicada no tiene ninguna pieza enemiga atacándola.
- **lastmove**
  Este elemento comprueba el último movimiento realizado por un jugador específico o cualquiera de los jugadores. Dentro de este elemento debes especificar dos sets de coordenadas, separando cada coordenada entre ellas con una coma (,) y cada set de coordenadas con un guion (-), siendo el primer set de coordenadas la casilla origen y la segunda la casilla destino.
  Esta condición será verdadera si el último movimiento de alguno de los jugadores coincide con el indicado. Se puede especificar que el movimiento lo tiene que haber hecho un jugador específico añadiendo un atributo “player” incluyendo el número del jugador como valor.

Los siguientes elementos son los que se pueden incluir dentro del elemento <specials> como resultados de los movimientos:

- **movepiece**
  Dentro se debe incluir dos sets de coordenadas, separando cada coordenada entre ellas por una coma (,) y cada set de coordenadas con un guion (-), siendo el primer set de coordenadas la casilla origen y la segunda la casilla destino. Debe existir al menos un elemento <movepiece>, que será el movimiento que tendrá que realizar el usuario para desencadenar el movimiento especial. Comprueba que la casilla origen tenga una pieza del jugador que quieres que pueda realizar el movimiento o el movimiento no se realizará.
- **createpiece**
  Dentro de este elemento, se debe incluir el carácter de la pieza que quieres crear, seguido de un guion (-) y este seguido de un set de coordenadas indicando en que casilla se quiere crear. Se debe tener en cuenta que crear una pieza reemplazará la pieza que se encuentre en esa casilla, indiferentemente del jugador propietario de esta.
- **removepiece**
  Dentro de este elemento, se debe especificar un set de coordenadas, separando cada coordenada entre ellas con una coma (,), indicando la localización de la pieza que se quiere eliminar. Hay que tener en cuenta que, si la casilla indicada está vacía, este elemento no tendrá efecto.