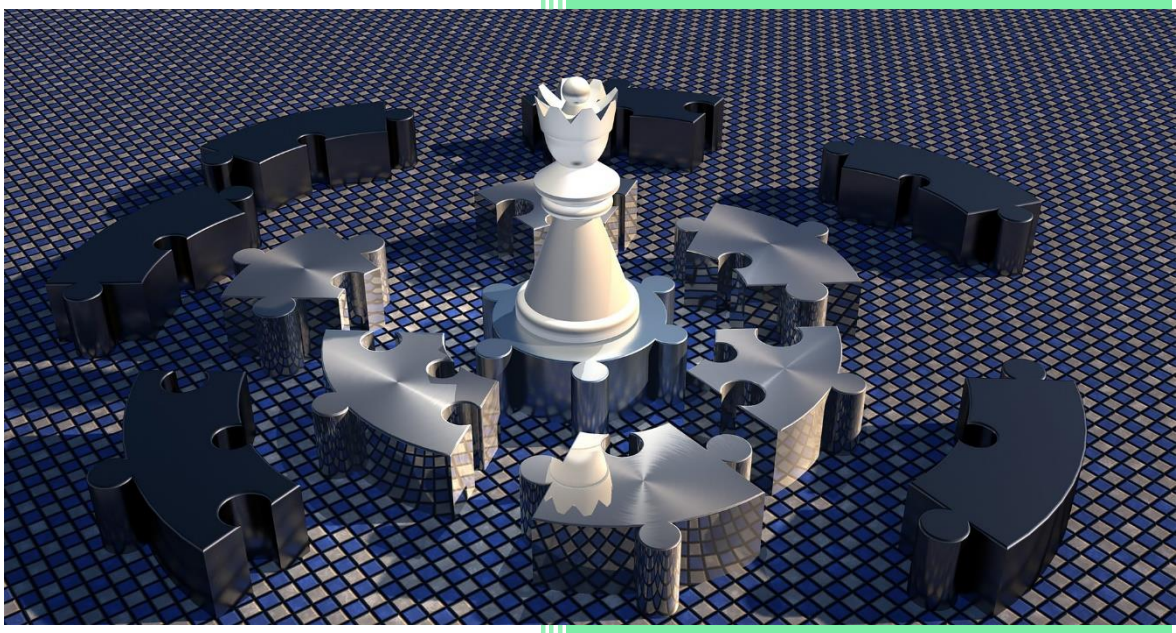
2019

Chess Battle Royale



Luis Pastor Abia

2° DAM B, I.E.S. Doctor Balmis

30-3-2019

ÍNDICE

1 INTRODUCCIÓN

En una ocasión, Bronstein tardó 40 minutos en mover ¡su primera pieza! Y luego ganó la partida, ¡un fenómeno! Cuando le preguntaron por qué, respondió mirando fijamente al tablero: "Estaba pensando donde había puesto las llaves de mi casa". ***En realidad la anécdota la cuenta el propio Bronstein pero tendría que buscar el libro donde lo cuenta así que permitidme que simplemente lo cite libremente…***

1.1 JUSTIFICACIÓN

Hay dos clases de hombres: los que se contentan con ceder a las circunstancias y juegan al whist; y aquellos que buscan controlar las circunstancias y juegan al ajedrez.

Edward James Mortimer Collins

La idea prendió en un descanso de clase. Medio en broma, medio en serio. Yo soy sólo culpable de recoger el guante, pensar más sobre ello y recopilar cosas que podrían (o no) funcionar. Y esto no deja de ser un primer acercamiento a una visión global que no sé cómo acabará. Pero ya lo dijo ***Lao-Tse***: *"Un viaje de mil millas comienza con un primer paso".* Caminemos

No voy a sorprender a nadie describiendo en qué consiste este proyecto. El mismo nombre y título son más que descriptivos. Pero, por si acaso, dediquemos en este apartado introductorio unos párrafos a la visión general de lo que se pretende conseguir.

Para los impacientes: sí, se trata de diseñar y realizar un video-juego conjuntando y sumando esas dos inspiraciones:

- Ajedrez
- Juegos Battle Royale

Así dicho parece más que sencillo (al menos en concepto) y que no requiere mucha más explicación. Pero vamos a empezar a darle otra vuelta de tuerca a la idea en sí.

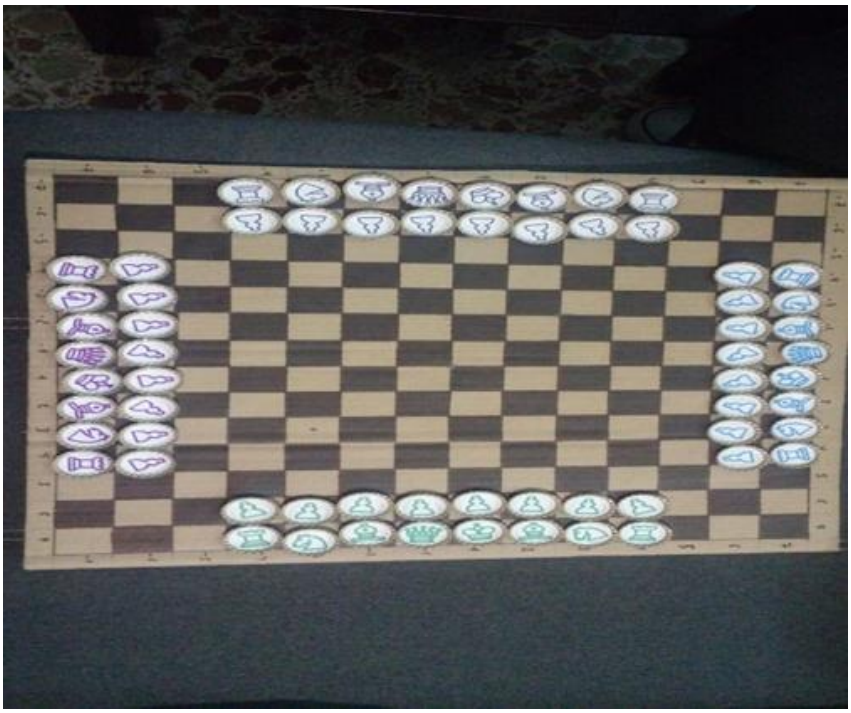
El ajedrez es un juego que tiene milenios a sus espaldas. Podría hablaros de su historia o de mi relación con él, de que es el juego-deporte-arte-ciencia al que se le han dedicado más literatura, de que es y ha sido uno de los primeros campos de batalla para la elaboración de programas de ordenador y de inteligencias artificiales (IA), de… tantas y tantas cosas. Parece complicado sacar algo nuevo sobre ello. Y, sin embargo, (y sin ponerme a aburriros con los datos ya que esto sigue siendo una introducción), sigue habiendo un mercado tanto de apps, bases de datos, canales de

streaming, programas de juego y aprendizaje, bastante maduro y competitivo.

Por otra parte, creo que hoy en día nadie puede dudar de la fortaleza del género de los Battle Royale (todos contra todos o como lo queráis ver). Empezó siendo la idea central de algún juego y otros copiando la idea y desarrollándose de mejor o peor manera han llegado a ser Juegos del Año para distintos *"Game Awards"* de la industria del video-juego. La premisa es bastante sencilla: mapa grande -> todos contra todos -> reducir mapa para forzar el enfrentamiento incrementando así paulatinamente la tensión y la emoción de una partida. Además de esos ejemplos hay numerosas franquicias de juegos que han caído más o menos en la tentación de tener entre sus ''modos'' de juego modos Battle Royale.

¿Es posible conjuntar ambos espíritus? Puede parecer a priori un poco complicado. Pero también dicen que el mundo es de los valientes...

Ya he mencionado que la idea surgió en un descanso de clase. Pero permitidme ahora que salte un poco adelante en el tiempo y muestre una imagen del prototipo que realicé cierto tiempo después. Miradla con ojos nuevos y pensad que os sugiere:



Vale, parece que es un simple ajedrez. De hecho ya hay ajedreces para 4 jugadores en el mercado. Nada nuevo bajo el sol, ¿no?

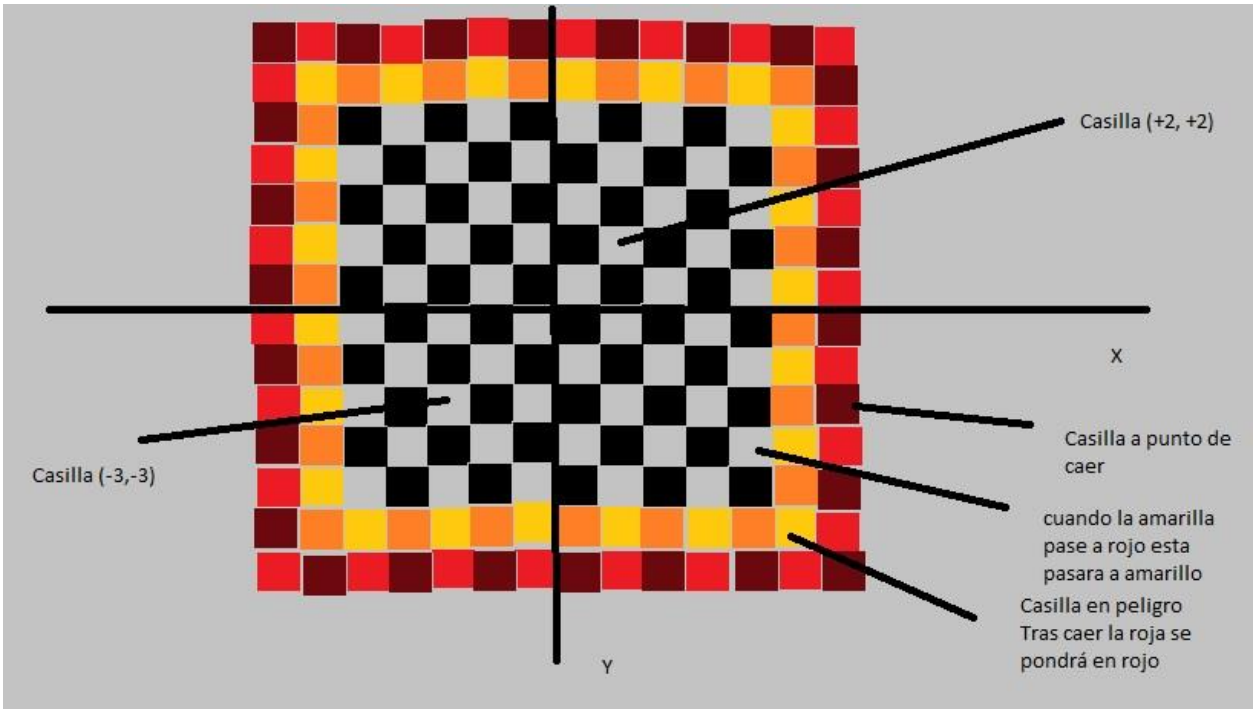
No obstante ahora quiero que imaginéis que comenzáis a jugar una partida (vale he de confesar que estas líneas las escribí antes de finalizar el proyecto con lo cual aún no se puede pero espero que de hecho cuando leáis estas líneas podáis jugar la partida).

Movemos una pieza, los rivales mueven las suyas. Movemos otra pieza. Y así discurre cierta parte de una partida. Curiosamente en algún momento notaremos que alguna pieza no se comporta como el ajedrez tradicional pero eso no es lo más importante.

Lo más importante es que de repente las casillas exteriores cambian de color. ¿Qué está ocurriendo?

Pasan unas jugadas y esas casillas se tornan rojas y las cercanas también cambian. Y tras un par de jugadas más esa parte del tablero desaparece llevándose con ellas las esperanzas de varias piezas que se encontraban en ellas y dejando un tablero más pequeño en el que ahora tienen que combatir las piezas supervivientes.

Y así poco a poco de un tablero gigante de 14x14 nos encontramos en un 12x12, 10x10, el tradicional 8x8 hasta llegar al 4x4 que preludia que pronto habrá un 2x2 y habrá acabado la partida.



Las oportunidades que surgen así son al menos inéditas (o eso me parece a mí). Por citar unas pocas que me vienen a la mente:

- Habrá momentos en los que los jugadores tengan que esforzarse por salvar las piezas que se encuentran en peligro y esa sea la mayor preocupación.
- Habrá momentos en los que habrá un espacio ilimitado que hará que las piezas mayores tengan una potencia inusitada al poder desplazarse de extremo a extremo.
- Habrá momentos en los que al menguar el tablero y lo mismo no desaparecer tantas piezas la concentración de ellas puede hacer que cualquier movimiento parezca opresivo.

- Tensión, novedad, tener que pensar en los movimientos de todos los jugadores implicados...

En resumen, sigue siendo un ajedrez, pero la influencia Battle Royale tanto en número de jugadores como en concentración de la acción lo hace bastante diferente.

1.2 OBJETIVOS

> *El ajedrez es una guerra sobre el tablero. El objetivo es aplastar la mente del oponente.*
>
> ***Bobby Fischer***

Existen distintos objetivos que se han querido cubrir con este proyecto. Y creo que lo mejor es clasificarlos en grupos:

1.2.1 Objetivos del juego/app en sí:

> *Don't remember where I was*
>
> *I realized life was a game.*
>
> ***Megadeth, A tout le monde***

1.2.1.1 El juego en sí.

Sí, puede parecer una perogrullada (y quizás lo sea) pero el principal objetivo es diseñar una app con una tecnología que se pueda exportar a distintas plataformas (la librería java LibGdx a usar cumple perfectamente este objetivo) que sea jugable. Aunque primero nos centraremos en que se vea en el PC.

1.2.1.2 Que el juego permita una posterior implementación de variantes y expansiones, y sirva como base para ellas.

Esta parte es bastante ambiciosa pero un objetivo personal es que el juego final permita la introducción y modificación de distintos parámetros y esté lo suficientemente bien diseñado con módulos independientes de tal manera que ir añadiendo variaciones sea una tarea relativamente sencilla y así se permita futuras versiones más elaboradas o incluso poder programar otros juegos tomándolo como base.

Sé que no voy a poder implementar en el desarrollo de este proyecto esas variantes, modificaciones y/o expansiones pero sí que espero que se sienten las bases para poder realizarlas.

1.2.1.3 Aplicación y uso de diversas tecnologías.

Usar distintas tecnologías simplemente por usarlas nunca puede ser un objetivo en sí. Pero, usar la tecnología adecuada (o una de ellas) para cualquier fin o aplicación e ir combinándolas estableciendo sinergias entre ellas sí que me parece un buen objetivo.

Y aunque el juego en sí utilice Java y LibGdX el proyecto del Chess Battle Royale (como en realidad cualquier proyecto que tenga como núcleo un juego) no es solamente el juego en sí, sino que hay numerosos componentes del sector productivo implicados. Creo que eso me dará la excusa perfecta para ir perfilando ciertos aspectos interdisciplinares relacionados con las aplicaciones informáticas y aunque no pueda dar más que una pincelada pues deben tener su representación.

Este objetivo casi enlaza perfectamente con el siguiente grupo de objetivos.

1.2.2 Objetivos formativos

La educación es el camino, no el objetivo.

A veces es complicado encontrar al autor de una frase…

1.2.2.1 Elaboración y diseño de la documentación del proyecto

Ya sé que es otro objetivo obvio pero uno de mis objetivos principales es que el proceso de recopilación de información, estructuración general del documento, elaboración de los guiones de trabajo y el trabajo en general queden bien documentados tanto en esta memoria como en los comentarios. A veces creo que se le da poca importancia a estos menesteres, y los veo de una importancia clara (y más como estoy comprobando en mis prácticas FCT que gran parte del trabajo que se desarrolla no es únicamente codificar sino estimar la duración, planificar las actividades, documentar y documentar las pruebas).

1.2.2.2 Documentar el ciclo de vida, la planificación del proyecto, control de versiones…

Siendo como es un trabajo individual, algunas de las herramientas que me hubiera gustado emplear quedan un poco descafeinadas (por citar el ejemplo más sencillo: aunque he estado usando tecnologías de control de versiones como el git, pues no he tenido que usar ramas y muchas veces casi me servía más como copia de seguridad en la red que la cantidad de usos que tiene como herramienta de trabajo colectivo). A pesar de ello creo que plasmar en esta memoria algunas de esas tecnologías empleadas dada la importancia que tienen en el desarrollo de cualquier proyecto informático pues no está de más (de hecho personalmente creo que debe ser una parte más importante de la formación de futuros programadores).

1.2.3 Objetivos "empresariales"

No conozco la clave del éxito, pero sé que la clave del fracaso es tratar de complacer a todo el mundo.

Woody Allen

Esta memoria y proyecto no deja de ser la memoria y proyecto que presento para la obtención del Título de Grado Superior de Formación Profesional de Desarrollo de Aplicaciones Multiplataforma; así que no pretende ser el germen de ninguna empresa (sería demasiado osado hasta para mí).

Pero lo que sí tiene como objetivo es realizar un somero análisis de dadas las oportunidades que se ofrecen en un sector como el diseño de videojuegos que anteriormente era monolítico y ahora hay una cuota de mercado para grupos y estudios más pequeños (conocidos como independientes) que inundan a menudo las *"stores"* de los dispositivos móviles.

1.3 Antecedentes (Análisis de lo existente)

Antes de entrar en un lugar, fíjate por dónde se puede salir.
Proverbio vikingo

Antes de centrarnos en el juego propuesto en sí hay que citar que intentos por hacer ajedreces con más de 2 personas ya ha habido con anterioridad. De hecho algunos historiadores postularon que una variante del Chaturanga (el Chaturaji) que se juega con 4 participantes era el predecesor del Chaturanga y éste del ajedrez [1]. Aunque sea a modo de anécdota pongo dos fotos de ajedreces de 3 y 4 personas.

Variaciones más o menos modernas y/o conocidas del juego del ajedrez también existen. Por citar algunas de ellas se puede ver una lista también en la Wikipedia [2] o consultar algunos libros donde se encuentran descritas [3]

El mundo de las variantes del ajedrez ha sido y es un tema con un creciente interés desde que los primeros programas de ajedrez comenzaban a luchar y vencer a grandes jugadores del mundo con cierta solvencia. De hecho alguno como el ajedrez aleatorio de Fischer se propuso como una alternativa que evitaba a esos ingenios electrónicos. La verdad es que aunque algunos de ellos han tenido hasta torneos internacionales con bastante bolsa en juego al final quizás no han alcanzado más status que el de variación curiosa. Aunque he de reseñar que en distintos sitios (p.e. en Lichess [4]) se puede jugar a estas variantes su éxito es menor que el ajedrez tradicional.

Desde las primeras eras de la era de computación han habido recreaciones del ajedrez en videojuegos. De hecho el esfuerzo para conseguir buenas inteligencias artificiales tuvo sus primeros combates en las 64 casillas.

Citaré por importancia histórica:

- Las historias del famoso autómata "El turco" [5] aunque más que inteligencia artificial en ese caso lo que había era trampa propia de trileros y no sea más que una cosa anecdótica.
- El superordenador de IBM Deep Blue que fue la primera computadora que ganó al campeón vigente del mundo de ajedrez en 1997 [6].
- Fritz de ChessBase [7]. Programa que dominó durante cierto tiempo los engines de ajedrez. Aunque personalmente creo que su principal relevancia fue la ChessBase [8] que siendo una base de datos completamente dedicada al ajedrez significó una pequeña revolución a la hora de preparar aperturas y al haber mayor circulación de información de las partidas a una mayor

preparación de las partidas por parte de los profesionales del ajedrez y de los amateurs que podían acceder a ello.

- AlphaZero [9] [10]. Creado por Google y que, en vez de usar como muchos motores algoritmos min-max para encontrar la mejor respuesta buscando entre millones de posiciones evaluadas gracias a la ayuda de expertos, usa redes neuronales y un algoritmo de aprendizaje genérico en unidades de procesamiento tensorial. Aunque hay cierta polémica sobre la fuerza comparativa al no poder competir con el mismo hardware (las TPUs son distintas a las CPUs convencionales y completamente diseñadas para las redes neuronales) lo cierto es que su irrupción en el campo de las IAs ha sido una especie de terremoto por sus buenos resultados.

Además existen otros motores de juego, videojuegos más orientados al amateur (algunos 3D, otros con batallas y animaciones entre las piezas en las capturas), algún juego de puzzles para enseñanza del ajedrez, cantidad de blogs, páginas, revistas, etc. Simplemente entrar en la tienda de Android y buscar chess o ajedrez puede dar una idea de la importancia que estos juegos casuales y de material web o de aprendizaje tiene. Hacer simplemente un recorrido por todos ellos sin llegar a ser siquiera exhaustivo puede ser…

Otro aspecto que me gustaría reseñar es la importancia del software de ajedrez para bases de datos donde hay que señalar el reinado comercial de ChessBase (aunque personalmente recomendaría otras bases de datos de licencia GPL como la Scid [11]).

Y dentro del ajedrez tradicional hay que tener en cuenta la importancia económica de los portales para jugar al ajedrez. Tanto PlayChess [12] de ChessBase, Internet Chess Club [13] (ambos con más de treinta mil suscriptores), FICS [14] (gratuito), Lichess [4] (corriendo bajo licencia AGPL o sea bajo software libre), Chess.com [15] (quizás el más visitado aunque también posiblemente no sólo por el portal para jugar tiene foro, web red social, artículos, etc.), Chess24 [16], etc.

Con respecto a otros juegos de tablero por computadora tengo que reseñar que aunque es complicado dar una estimación realista existe una extensa amplitud de propuestas tanto en la tienda de Steam como en los dispositivos móviles.

Hay que tener en cuenta que aunque propiamente no son de tablero (no han tenido su contraparte física por así decirla) numerosos juegos de estrategia o RPG por turnos pueden verse como juegos de tablero en los que dicho tablero usando las posibilidades que confiere el formato digital para enriquecer la experiencia del jugador.

Aún con todos estos antecedentes creo que sigue siendo un campo lo suficientemente inexplorado como para que existan posibilidades de

innovación (y no solamente en base a mejores gráficos y texturas) sin llegar a realizar clones como abundan en las susodichas tiendas digitales.

¿Existe algún antecedente de intentar hacer juegos de tablero por computadora con algunas similares a lo que se pretende mezclar en este proyecto, con variaciones en el tablero para hacer cambiar el campo de batalla?

Pues hay distinto tipo de franquicias La verdad es que lo desconocía pero el mundo de los videojuegos es inmenso y buscando buscando resulta que relativamente cercano en el tiempo (10/12/2018) ha salido a la venta en Steam un videojuego de Facepunch Studios titulado Clatter. Con bastantes diferencias sería el referente, aunque está muy por encima de mis posibilidades de este proyecto. (No deja de ser un estudio con un cierto número de integrantes y experiencia y le dedicaron una cantidad mayor al diseño y desarrollo de dicho juego).

Sin embargo, creo que el Chess Battle Royale al ser más cercano al espíritu del ajedrez puede tener un futuro más prometedor dentro de los deportes mentales.

2 NECESIDADES EMPRESARIALES PARA EL DESARROLLO

La mejor forma de vender algo: no venda nada. Gánese la confianza y el respeto de aquellos que podrían comprar

Rand Fishkin

2.1 ¿PLAN DE EMPRESA?

A veces los problemas se ven más grandes cuando vienen de camino que cuando llegan.

Leído por ahí y aunque desafíe a la física me sacó una sonrisa pesimista de las mías.

Aunque crea en las posibilidades económicas de Chess Battle Royale, soy realista y cuando finalice la entrega de proyecto este aún se encontrará en pañales y requeriría un esfuerzo mayor para comenzar a ser vendible. En este apartado quiero únicamente pincelar cuales serían las estrategias a seguir para la comercialización y puesta en marcha de este juego.

A pesar de todo, me he permitido la libertad de proponer un escenario que en algunos puntos quizás peca de demasiado ambicioso, (o de poco realismo según se mire), de que necesidades empresariales requeriría para llevar a último término este proyecto a un nivel más allá del educativo y didáctico y pasar a ser un proyecto "real".

2.2 RECURSOS HUMANOS

Al final, todas las operaciones de negocios pueden ser reducidas a tres palabras, gente, producto y beneficios. A menos que tengas un buen equipo, no tienes mucho que hacer con las otras dos.

Jack Welch Jr

La tarea más importante en este sentido es una correcta planificación del personal a un medio o medio-corto plazo ajustándose a las necesidades empresariales con el menor costo y número de personal posible al tratarse de una empresa nueva y que necesita introducirse en el mercado. Y eso no será sencillo y costará.

No obstante, hay que tener en cuenta que la interdisciplinariedad en el sector de los video-juegos más que algo extra o decorativo es una seña de identidad y, sin ella, no habría ninguna posibilidad de éxito con lo cual hay que buscar abarcar la mayor amplitud de miras posibles en distintos campos para que la tarea tenga éxito.

De esta forma, aunque no se deba olvidar en el desarrollo del software en sí, habrá que intentar sumar al proyecto (emplear) a gente entusiasta por otros campos. De esta manera creo que deberíamos contar con los siguientes socios o empleados (y de paso ponemos sus funciones de las que se encargarán):

- Una persona de perfil front-end y relaciones con el cliente/usuario que se ocupe del desarrollo y mantenimiento de una presencia en la web y en redes sociales, con un espacial interés e hincapié en programar/realizar/mantener acciones dentro del marketing digital, y otras actividades para mantener el juego vivo.
- Un comercial/relaciones públicas más tradicional que se encargue de buscar oportunidades y reuniones con distintos clubs de ajedrez y escuelas ofreciendo la plataforma (y posibles adaptaciones de ellas y colaboraciones con otras entidades tanto públicas como privadas) para hacer acciones publicitarias y divulgativas del juego. Conviene que tenga cierto saber en el juego en sí (y en el ajedrez tradicional).
- Un desarrollador de la aplicación en sí con fuertes conocimientos y deseos de experimentar en los aspectos más visuales y de front-end de la aplicación. Un hombre ha de conocer sus limitaciones y creo que necesitaría tanta ayuda en ese aspecto que delegar ese tipo de tareas a otro programador que no fuera yo creo que sería la mejor idea.
- Yo, que me encargaría de la dirección del proyecto en sí y de intentar mejorar la IA y los algoritmos back-end que usara la aplicación (y el posterior tratamiento de los datos para crear material que pueda incidir en un mejor aprovechamiento y divulgación del juego).
- Por último, aunque esto tendría que esperar más adelante, el siguiente responsable a ser seleccionado para formar parte de la empresa debería ser alguien que se ocupara tanto de la arquitectura de hardware necesario y del escalamiento necesario si de verdad se consigue una cierta ampliación de nuestro nicho de mercado o de nuestras posibilidades de ventas. Hasta que eso ocurra habría que ir trabajando con las limitaciones claras de usar lo más barato posible en todo ello.

Sumando hacen como mínimo un total de 4(+1) personas necesarias. Aunque se vayan a explorar distintas alternativas para la financiación del proyecto parece claro que es de un calado bastante grande. De hecho creo que la mejor alternativa sería plantearse la creación de una sociedad buscando socios que se involucrasen tanto económica como físicamente en el proyecto y fueran cubriendo esos puestos.

De esta manera las obligaciones frente a la Seguridad Social serían únicamente las personales y…

(preguntar a Lara… no sé si eso va por buen camino, la idea es que no coste tanto el monte de formar todo esto pero no sé si será legal…)

2.3 PREVENCIÓN DE RIESGOS LABORALES

Más vale prevenir con las manos en la masa y tendré que tener cuidado con no sufrir un ataque de nostalgitis de cuando era mucho más pequeño.

Yo mismo

2.3.1 Riesgos laborales

Si no se conoce la causa de los fenómenos, las cosas se manifiestan secretas, oscuras y discutibles, pero todo se clarifica cuando las causas se hacen evidentes.

Luis Pasteur

La mayoría de los puestos de la empresa consisten en realizar trabajos delante de pantallas de visualización (ordenadores). Así que voy a centrarme en realizar un resumen de los riesgos asociados al puesto de trabajo. Aunque un breve resumen de los riesgos asociados al lugar de trabajo siempre es pertinente.

En cualquier caso para un mayor detalle sobre estos menesteres se puede consultar la numerosa bibliografía que existe sobre los riesgos de trabajadores de oficina. Una parte de la cual hemos reflejado en el apartado de bibliografía. [17] [18] [19] [20] [21]

2.3.1.1 Riesgos asociados al lugar de trabajo

2.3.1.1.1 Caídas al mismo nivel

Las lesiones que se producen con mayor frecuencia por caídas son torceduras, esguinces, contusiones, heridas y fracturas.

Hay que vigilar que los suelos estén convenientemente limpios (pero no recién fregados o encerados), tener una iluminación adecuada, evitar obstáculos en pasillos y áreas de trabajo, cables, objetos que entorpezcan el paso o presencia de baldosas rotas o levantadas. También el trabajador debe evitar correr o cometer imprudencias que puedan dar lugar a un accidente, usar un calzado preferentemente con suela antideslizante.

2.3.1.1.2 Caídas a distinto nivel

Fundamentalmente por escaleras presentes en el lugar de trabajo tanto fijas como móviles.

Es importante el buen uso de estas de una manera responsable evitando distracciones de recorrerlas mientras se usa algún dispositivo móvil o papeles aparte que tengan unas características técnicas tanto los escalones como las barandillas adecuadas y que cumplan la normativa vigente.

2.3.1.1.3 Riesgos eléctricos

Es una obviedad pero la electricidad es algo importante cuando se trabaja con ordenadores y hay que tener en perfectas condiciones los enchufes (evitando entre otras cosas la sobreutilización de ellos con ladrones y el uso de alargadores innecesarios o en malas condiciones) y la buena organización (y estado) de los cables. Hay que evitar el contacto de electricidad y agua.

2.3.1.1.4 Golpes y cortes con material de oficina

Hay que tener cuidado con el uso de tijeras, grapadoras, cutters, papel, etc. Y con el orden en general evitando posibles cortes y golpes contra obstáculos, cajones, mobiliario movido de sitio, etc.

2.3.1.1.5 Incendios

La normativa para evitar y prevenir incendios y la forma de actuar frente a ellos y las medidas a tomar es bastante amplia y dependerá en gran medida en que lugar y que dimensiones tenga el posible lugar de trabajo. Por ello hasta no tener claro este punto sólo me queda decir que habrá que elaborar el correspondiente plan de emergencias y evacuación teniendo en cuenta la posibilidad de que existan miembros ajenas de la empresa, cumplir con la normativa vigente en cuestión de alarma, señalización, salidas de emergencia y extintores.

Como siempre hay que recordar que ante una situación de emergencia no se puede perder tiempo recogiendo enseres personales (o de cualquier tipo), que no se deben usar ascensores y cómo actuar en distintos escenarios (p.e. encerrados por el fuego cerrar puerta y tratar de tapar con trapos mojados, etc.).

2.3.1.2 Riesgos asociados al puesto de trabajo

Nos vamos a centrar en los riesgos derivados del uso de pantallas de visualización (o simplemente pantallas) y en otros factores asociados al trabajo como condiciones inadecuadas de temperatura y/o humedad, iluminación, ruido, la silla, mesa o mobiliario en general inadecuado, la organización del trabajo, la carga del trabajo en sí, el propio trabajador y sus compañeros, etc.

Los principales riesgos que aparecen por el uso de pantallas son correspondientes a la disminución de la capacidad física y mental después de realizar un esfuerzo por un tiempo, o en otras palabras; a la fatiga. Esta fatiga para los casos de trabajo en oficina se divide en fatiga postural (que puede dar lugar a trastornos musculo-esqueléticos), fatiga visual (que puede derivar en trastornos visuales y molestias oculares) y fatiga mental (con posibles alteraciones del sueño y emocionales y otros trastornos psicosomáticos).

2.3.1.2.1 Fatiga postural

Las principales causas de la fatiga postural es debida a las siguientes circunstancias:

- Distribución inadecuada y falta de regulación de los elementos de trabajo
- Deficiente iluminación.
- Hábitos inadecuados.
- Deficiente diseño de los elementos y/o mobiliario inadecuado.
- Movimientos repetitivos.
- Estatismo postural.

Este estatismo postural es muy importante y muchas veces es consecuencia de malos hábitos de trabajo inadecuados (no realizaciones de pausas, mala colocación en la silla o no poner de manera adecuada la pantalla, teclado, etc. El resultado puede ser provocar molestias y lesiones sobre todo en la zona cervical y dorsal.

2.3.1.2.2 Fatiga visual

Síntomas de fatiga visual puede ser pesadez de los ojos, somnolencia, borrosidad, dolores de cabeza, etc. Esta fatiga puede producirse por el esfuerzo visual de diferentes intensidades lumínicas y distancias, mala disposición de los elementos de trabajo y fallos en la calidad de la pantalla, reflejos, demasiado tiempo delante de ella.

2.3.1.2.3 Fatiga mental

Hay muchos factores que inciden en la carga mental de trabajo como las propias de las exigencias del trabajo (exceso de cantidad y complejidad de información, poco tiempo para realizarlo, tensión en la toma de decisiones). Todas ellas pueden ocasionar un estado de fatiga transitoria que si se mantiene en el tiempo puede llegar a desembocar en fatiga mental crónica de consecuencias graves tanto dentro del trabajo (baja de rendimiento, absentismo, etc.) como fuera.

2.3.2 Medidas preventivas

Es mejor encender una pequeña vela que maldecir la oscuridad.

Proverbio chino

2.3.2.1 Equipo de trabajo

2.3.2.1.1 Pantallas

- ✓ Buena resolución, imagen estable, ajuste de luminosidad y contraste, fondo de pantalla no demasiado oscuro, orientable e inclinable.
- ✓ Se recomienda que la distancia del ojo a la pantalla sea superior a 40 cm y que la parte superior quede a la altura de los ojos.
- ✓ Hay que evitar que el trabajador esté de frente o de espaldas a la ventana. Lo mejor es perpendicular a la ventana.

2.3.2.1.2 Teclado

- ✓ Inclinable e independiente de la pantalla para poder adaptar su posición y evitar posturas inadecuadas.

2.3.2.1.3 Ratón

- ✓ Su posición en la mesa debe permitir apoyar la muñeca sobre la mesa.

2.3.2.1.4 Portadocumentos

- ✓ Recomendable si es necesario trabajar con documentos impresos colocándose a la misma altura que la pantalla y mismo plano de visión para reducir problemas en el cuello y los cambios de acomodación visual.

2.3.2.1.5 Mesa

- ✓ Debe ser suficiente para colocar la pantalla y el teclado a la distancia adecuada y apoyar en ella manos y brazos estando a la altura de los codos y distancia a la silla para permitir libertad de movimientos.

2.3.2.1.6 Silla

- ✓ Debe ser estable (se recomienda de 5 apoyos (o ruedas) en el suelo y de altura regulable. Con respaldo reclinable (que de apoyo a la zona lumbar) y altura ajustable. Si tiene reposabrazos estos no deben impedir que se pueda acercar a la mesa.

2.3.2.1.7 Reposapiés

- ✓ Solo necesario cuando los pies no alcancen bien al suelo y la mesa no sea regulable en altura.

2.3.2.2 Factores ambientales

2.3.2.2.1 Iluminación

- ✓ Hay que garantizar unos niveles adecuados con su correspondiente mantenimiento y limpieza de las fuentes de luz.

2.3.2.2.2 Ruido

- ✓ Sería recomendable intentar minimizar el ruido y controlar el nivel de volumen de las conversaciones.

2.3.2.2.3 Temperatura

- ✓ Verano (23-26), Invierno (20-24) con humedad entre 45 y 65%.

2.3.2.2.4 Espacio

- ✓ Debe permitir cambios de postura y movimientos, situando las cosas más usadas a una correspondiente distancia para realizar el menor esfuerzo.

2.3.2.3 Trabajador

2.3.2.3.1 Medidas para evitar la fatiga postural

- ✓ Mantener hombros hacia atrás con la espalda recta y los riñones sujetos al respaldo evitando sentarse en el borde o mitad del asiento.
- ✓ Evitar giros frecuentes de la cabeza e inclinarla hacia atrás. El cuello debe estar lo más recto posible y con la mirada hacia delante con la cabeza levantada.
- ✓ Regular la silla y el monitor para tener una visión cómoda.
- ✓ Las manos deben descansar sobre la mesa en un ángulo brazo antebrazo de unos 90° mientras se teclea.
- ✓ Evitar giros e inclinaciones o estiramientos de brazo para ordenar el material. No agacharse doblando la espalda.
- ✓ Cambiar frecuentemente de postura de trabajo realizando pausas y levantándose de vez en cuando, estirando o caminando un poco.
- ✓ Realizar ejercicios de relajación y estiramiento de cuello, brazos y espalda.

2.3.2.3.2 Medidas para evitar la fatiga visual

- ✓ Pausas para relajar los ojos.
- ✓ Evitar deslumbramientos directos e indirectos.
- ✓ Adaptar la iluminación complementándola con luz artificial y/o localizada.
- ✓ Revisiones periódicas de la vista.

2.3.2.3.3 Medidas para evitar la fatiga mental

- ✓ Alternar tareas con algunas que demanden menores esfuerzos mentales.
- ✓ Pausas.
- ✓ Organizar el trabajo, aprender a gestionar el tiempo y planificarlo estableciendo prioridades de actuación.

- ✓ Dormir lo suficiente.

2.3.2.4 Organización del trabajo.

En el contexto laboral los factores psicosociales y de organización afectan y pueden dar lugar a la aparición de efectos negativos.

Ante ellos sería deseable poder alternar la tarea con otras más relajadas o que exijan un menor nivel de atención, tener cierta libertad y flexibilidad para tener cierta autonomía temporal de las pequeñas pausas y acometer el trabajo a un ritmo adecuado adaptando la carga permitiendo tiempo para la recuperación de la fatiga.

También sería deseable una comunicación fluida a todos los niveles para facilitar el intercambio de información y reducir la ambigüedad si no se dispone de la suficiente información al igual que conocer perfectamente las funciones y responsabilidades de cada puesto.

Utilizar correctamente los sistemas de información y participación en la empresa ayudan a solventar problemas y aumentan la motivación. Al igual que con las relaciones entre los compañeros.

2.4 INICIATIVA EMPRENDEDORA

No he fracasado. He encontrado 10.000 maneras que no funcionan.

T.A. Edison *(Aunque estoy seguro que todos lo hemos pensado en alguna ocasión)*

Casi todo tuyo Lara… Bueno en realidad lo tengo que hacer yo y lo haré yo, pero si puedes echarme una mano y no al cuello… Creo que es necesario dado lo que me he "inventado" que sea una mini-sociedad, lo que dudo es entre SL normal pero como eso es de capital y no tiene en cuenta trabajadores pues lo mismo una sociedad comanditaria o como se llame pues es mejor idea… no sé, cuándo lo leas me cuentas.

2.4.1 Forma jurídica

El talento gana partidos, pero el trabajo en equipo y la inteligencia gana campeonatos.

Michael Jordan

2.4.2 Trámites administrativos

Administración, s. En política, ingeniosa abstracción destinada a recibir las bofetadas o puntapiés que merecen el primer ministro o el presidente.

Ambrose Bierce

3 REQUISITOS FUNCIONALES DE LA APLICACIÓN

El diseño no es solo lo que se ve o lo que se siente. El diseño es cómo funciona.

Steve Jobs

Como ya he dicho se trata de que con reglas más o menos familiares a las del ajedrez hacer un tablero que irá disminuyendo conforme pasa el tiempo y así obligarse a pelearse en el centro. Tengo en mente varias variaciones más o menos simples del juego que elaboraré en las Especificaciones más adelante. Pero en general se trata de abarcar de manera global tanto el juego principal y apps asociadas como todo lo que se puede llegar a mover alrededor del juego en sí.

A nivel del juego habrá que hacer un juego de tablero 2D pero con un tablero "flexible" que permita su modificación y varias particularidades. Habría que retocar algunos movimientos de las fichas de ajedrez, al igual que algunas reglas. Todo ello llevará a una necesidad de creación de distintos apartados a modo de Tutorial (ver la página web que sigue a modo de ejemplo inicial http://luispivo.github.io/ChessBattleRoyale/tutorial.html).

En principio trabajaría en modo desktop para explorar las capacidades de pantallas más grandes debido a trabajar con más de 64 casillas. En cualquier caso como se trabajará con la **librería java LIBGDX** la portabilidad hacia otras plataformas no debería presentar mayores dificultades.

Al final el objetivo del proyecto inicial será tener un juego más o menos jugable dejando para posteriores implementaciones las distintas "expansiones" quedando en realidad para presentar una especie de fase pre-alpha.

3.1 EL JUEGO EN SÍ (EN UN PRIMER ESTADIO LOCAL Y SIN MOTORES DE IAS COMPLEJOS).

No podemos ser nada sin jugar a serlo.

Jean-Paul Sartre *(hablando del ser en sí y demás no pude contenerme y me sonó a existencialismo…)*

3.1.1 El tablero

Siempre habrá un campo de batalla.

(Bueno aquí al final va desapareciendo pero ya se me entiende…)

El tablero cobra una importancia capital en el ChessBattleRoyale y es lo que le da el picante al juego, pero al mismo tiempo es lo que hace más complicado la codificación del juego. Al tener que tener la flexibilidad de tener distintos tamaños y estado de casillas (no peligro de desaparecer, peligro de desaparecer, desaparecida, señalada por el jugador o no señalada, con pieza encima o no…). Pues lo hace más complicado que simplemente coger una matrix 8x8 y poco más como suele ser en el ajedrez tradicional.

Además si se programa cualquier IA ha de tener en cuenta estos cambios de tamaño aparte que si ya las cantidades de movimientos diferentes que hay en el ajedrez son una cantidad muy elevada, podéis imaginaros que ocurre cuando pasamos a tener 14x4 y el doble de piezas y jugadores en él. Ya los programas bien optimizados y realizados con equipos muy expertos a cuesta tienen problema pues para cuatro es aún más complicado (ya hablare de ello más profusamente cuando lleguemos al apartado de IAs).

Habrá que ir controlando cuando van desapareciendo las filas para que agregue emoción a las partidas (turnos y/o tiempo). Y eso será otro parámetro para ir modificando el tipo de juego.

3.1.2 Número de jugadores

Dos es compañía, tres multitud…

¿4 como en este juego que es como una concentración?

4 Jugadores. Creo que es un buen compromiso entre el ajedrez clásico y juegos de tablero (2 jugadores) y juegos todos contra todos. Aunque la complejidad es evidente que aumenta. De hecho hay estudios que indican la dificultad que conlleva tener que calcular y/o "prever" para la mente humana los movimientos de tanta gente. Y para la IA es todavía terreno aún más inexplorado.

3.1.3 Piezas y movimientos cambiados:

Aunque se quede abierto a la experimentación de otras posibles piezas creo que la base de las piezas del ajedrez y normas del ajedrez se pueden extrapolar perfectamente al Chess Battle Royale con (a priori) estas modificaciones:

3.1.3.1 Peones

Peones: ellos son el alma del Ajedrez; solos, forman el ataque y la defensa.

A. D. Philidor, siglo XVIII

¿No eran el alma del ajedrez? En Chess Battle Royale siguen siéndolo aunque hayan perdido en su mochila la oportunidad de coronar y convertirse en otras piezas, (al fin y al cabo no tiene sentido que lleguen al otro campo del tablero cuando este tablero cada vez es más pequeño). ¿Qué reciben a cambio para continuar siendo el alma y un aspecto sobre el que girará gran parte de la estrategia y del juego?

Nada más y nada menos que la posibilidad de retroceder y matar hacia atrás.

Puede parecer una modificación menor y más cuando su movimiento se limita (nada de 2 casillas de alcance y por ende nada de comer al paso) aún más y que dado que el tablero va disminuyendo... Pero aquellos que sois avezados y expertos jugadores de ajedrez sabéis lo que condiciona una partida las cadenas y estructuras de peones y podéis empezar a pensar el efecto muralla que pueden tener cuando además son capaces de defender a las piezas que se encuentran detrás. Y además añadidle la elasticidad de poder modificar dicha estructura con un retroceso en un momento dado. ¿Comenzáis a sospechar las posibilidades?

Como Tips adicionales podemos decir que habrá que estudiar como escudarse con ellos y cuando abrir la estructura detrás de ellos para dejar sacar la furia de las demás piezas. Un buen jugador de Chess Battle Royale ha de tener un fino juego de peones.

3.1.3.2 Caballos

Un caballo, un caballo, mi reino por un caballo!

Ricardo III, William Shakespeare

Su movimiento tan aclamado en L sigue igual. Su peculiaridad de saltar permanece. Su valor parece disminuir dado que sigue siendo la más "lenta" entre las piezas mayores. Habrá que ver como los cambios en los peones le afectan. Cuando estos consigan ser una formación compacta y enmarallar la partida el factor sorpresa de la caballería tras ellos puede ser más importante. Si las estructuras se abren y las murallas de los peones desaparecen quizás los caballos sólo sirvan de distracción y de escolta a otras piezas (aka el rey) que a diferencia del ajedrez tradicional tiene que tener un papel más activo en la partida. Al fin y al cabo si se limita a intentar esconderse tras las piezas y no moverse demasiado la disminución del tablero...

Todo ello hace que a priori los caballos sean una fuerza que ha de coordinarse con los peones y otras piezas para funciones defensivas o cuando el tablero se estrecha y estrecha. ¿No parece un poco al contrario que en el

ajedrez tradicional en el que el caballo siempre ha reinado más y mejor en las etapas iniciales de la contienda al poder desplegarse más rápidamente?

3.1.3.3 Alfiles

> *Alfiles malos protegen a los peones buenos.*
>
> ***M. Suba***

Otro que no cambia. Pero imaginar su potencia en un tablero libre 14x14... Y de la misma manera imaginar su desventura si la muralla de peones no se descompone... Pueden ser los cuchillos que apuñalen a los adversarios fácilmente en un puro ejercicio ofensivo. Y al mismo tiempo también tendrán un valor defensivo y coordinable con los peones al matar en diagonal también nada desdeñable. Como siempre una pareja de alfiles bien engrasada son una herramienta capaz de cambiar una partida.

3.1.3.4 Torres

> *Donde quiera que el ajedrez se menciona en crónicas antiguas o romances métricos, es con ocasión de algún acto de violencia o enemistad amarga. El gran tamaño de las primeras piezas de ajedrez, y el uso del metal en las tablas, deben haberlas convertido en armas tentadoras para un hombre enojado. Las torres, en particular, parecen haber sido utilizadas a menudo como héroes de Homero que emplean una gran piedra.*
>
> ***Howard Staunton.***

Al perder el enroque y con los cambios de los peones parece que será más complicada la tarea de ponerlas en funcionamiento ¿no? Seguramente será verdad. Además no cambian nada... Y sí, en tableros grandes la importancia de las líneas abiertas será más importante pero tampoco es que tengan un alcance mayor que los alfiles... Pero hay algo nuevo para las torres que creo que las beneficiará. No es algo tan aparente a primera instancia y menos cuando ves el Chess Battle Royale con los ojos de un ajedrecista.

Pero... ¿no se nota en la distribución inicial que ahora la torre no está confinada en la esquina? ¿Y eso que significa para una pieza que gusta de espacios y líneas abiertas? ¿Hace falta que lo diga?

3.1.3.5 Dama

> *El hombre va como los peones: de casilla en casilla sin poder atrapar a la dama.*
>
> ***Francisco de Quevedo.***

Quizás es la pieza que menos cambia su función. Pero ya era la pieza más poderosa del ajedrez tradicional y continua siéndolo. Así que no hay que preocuparse demasiado por ello. Combina los movimientos de alfil y torre y aunque no gana la libertad de la torre y será mucho menos deseable involucrarla en el nuevo papel defensivo de los alfiles pues sigue siendo versátil y capaz de movilizarse con facilidad. Así que mucho cuidado con esta pieza y su posibilidad de combinarse con todas las demás.

3.1.3.6 Rey

> *El Rey es una pieza de pelea. ¡Úselo!*
>
> ***W. Steinitz.***

En el Chess Battle Royale hay una pieza que no parece cambiar para ningún aspecto positivo. Y sí, es la pieza más delicada del tablero porque una vez que desaparece, desaparecen con ella toda la ilusión de la partida pues esta acaba. Y sí, en Chess Battle Royale el rey es aún más débil que en el ajedrez tradicional por los siguientes cambios:

Fuera enroque y formas para desaparecer a un lugar seguro del tablero...

Vale que los peones ahora le protegerán mejor pero ¿cuánto tiempo cree el pequeño rey que podrá esconderse tras ellos cuando la amenaza de desaparición del tablero le empuje una y otra vez hacia el centro y la batalla?

¿Serán esos cambios capaces de eliminar todas las estrategias y tácticas aburridas de reyes escondiditos y traerá los viejos tiempos de Steinitz o de los finales donde los reyes han de ir al frente a luchar o no? No lo sabemos y dependerá de la partida al final pero lo que está claro es que en una partida de Chess Battle Royale la habilidad de tener tacto con el rey será más importante que nunca.

3.1.3.7 Enroque

> *Sólo los cobardes enrocan.*
>
> *Enrócate pronto y a menudo.*
>
> ***Rob Sillars***

Ya lo hemos hablado al hablar del rey (y de su seguridad). Pero lo justo es que la eliminación de este movimiento especial al afectar a dos tipos de piezas tenga su propio mini-apartado para dejar claro que en el Chess Battle Royale no existe el enroque. Al menos en esta implementación pre-alpha.

3.1.3.8 Jaque mate

> *El Ajedrez moderno está demasiado preocupado con cosas como la estructura de Peones. Olvídenlo, el Jaque Mate termina la partida.*
>
> ***GM Nigel Short.***

De nuevo hay posibilidades de hacer cambios con esta regla en un futuro pero para esta implementación al que le coman el rey (o este desaparezca abruptamente por un costado del tablero) perderá la partida y desaparecerán el resto de las. Creo que es un buen compromiso entre el espíritu ajedrecístico natural y la complejidad de tener que ver si los otros contrincantes pueden (y/o quieren) salvar a un rey que supuestamente no puede hacer nada por salvarse en su siguiente turno. Siento la perdida caballeresca de no comerse nunca al rey y que se parezca más al juego de niños que uno se equivoca y el otro le quita el rey y se ríe pero… Digamos para sentirnos mejor que además es el convenio usual en partidas rápidas.

3.1.4 ¿Turnos o Estrategia a tiempo real (Expansión)?

> *La vida es demasiado corta para el ajedrez.*
>
> ***Lord Byron***

Esta primera versión del juego se realizará con turnos.

Aunque como en el ajedrez habrá que añadir el factor tiempo (relojes). Intentaré incluir distintos tiempos y que se puedan escoger. Aunque la primera idea es usar la tradicional forma de jugar al ajedrez con relojes (tiempo fijo para x movimientos) pero creo que la mejor primera aproximación es tiempo fijo corto para cada movimiento y si no se pierde turno o al menos un sistema de reloj como el que propuso D. Bronstein en una ocasión de tener siempre un mínimo de tiempo por jugada aparte de un tiempo para toda la partida.

Aún así puede que en el mundo contemporáneo la gente prefiera un juego más vivo e intenso propio del género de estrategia a tiempo real (¿pausable vs la IA?) permitiendo mover una pieza tras pasar una pequeña cantidad de tiempo. Como digo puede ser una vía de modificación futura el permitir tanto distintas formas de establecer el tiempo como la de tiempo real. Pero eso quedará para posteriores versiones del proyecto.

3.2 INTELIGENCIA ARTIFICIAL

> *¿Hay que tener inteligencia para jugar? No, el Ajedrez hace a la gente inteligente.*

Emanuel Lasker

Algo similar a la cita se podría decir históricamente del ajedrez y de la Inteligencia Artificial como ya analizamos en la sección de Antecedentes.

Pero como ya dijimos allí no es lo mismo el Ajedrez convencional con el ChessBattleRoyale y el tener más casillas en el tablero (y que este disminuya) y duplicar el número de jugadores hace más complicado desarrollar una buena IA (y mucho menos llevarla al nivel que algunos programas han conseguido con el ajedrez convencional).

Sin embargo, para un correcto juego de un único jugador, como para tener posibilidades de los primeros esbozos de usar el programa como primera versión para tener funciones de Base de Datos y de replays necesitaremos desarrollar algún motor de IA. Mis expectativas con respecto a esto van a ser completamente realistas y tampoco espero que su fuerza de juego sea brillante.

Una discusión un poco más teórica de las posibilidades para aplicar una IA se pueden ver en el apéndice C. En este apartado nos vamos a limitar a partir de ahora a describir como he implementado la primera IA del ChessBatleRoyale (la SillyIA).

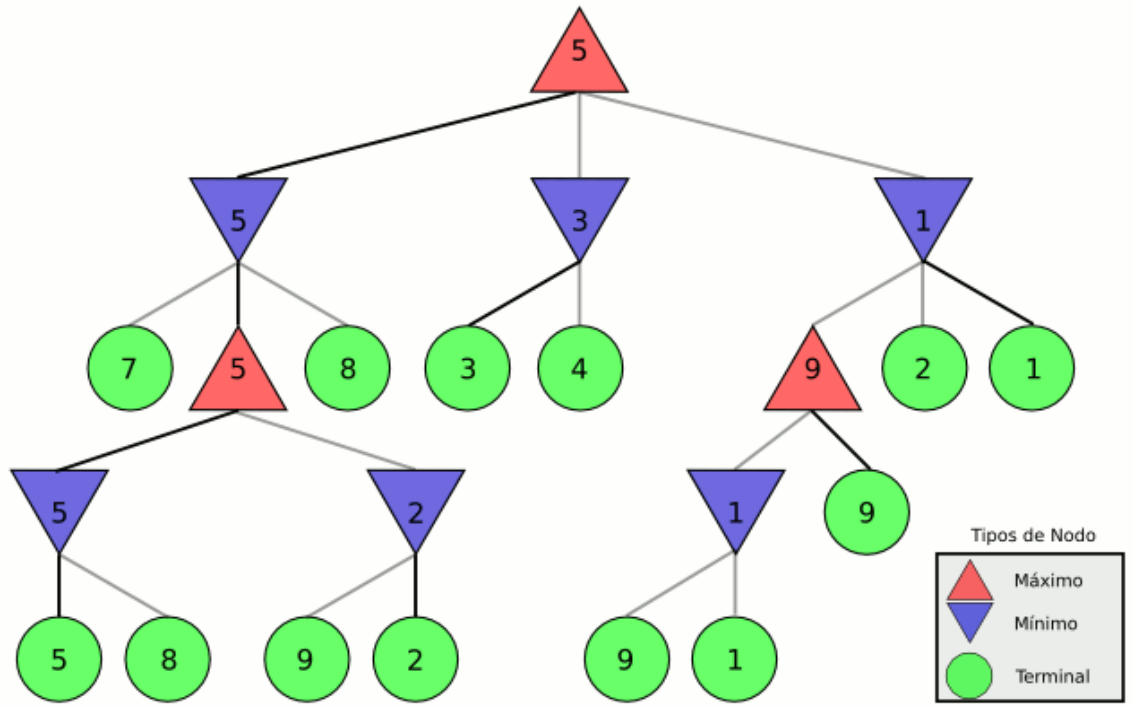
3.2.1 La SillyIA

3.2.1.1 El algoritmo MinMax

La teoría de juegos en la que se fundamenta este algoritmo dice que para juegos de suma cero (lo que alguien gana otro pierde) y con información perfecta (todo el mundo sabe todo de la posición o estado del juego) el método para alcanzar la solución consiste en minimizar la perdida máxima esperada por los contrincantes eligiendo el mejor movimiento para si mismo (y asumiendo que los demás escogerán el peor para ti), es decir, usar un algoritmo de minimax [22].

De hecho aunque las nociones matemáticas comienzan en la década de 1920 (Borel y Von Neumman) en el ajedrez no confiar que el rival va a hacer la peor jugada o caer en la celada es un hecho conocido. La dificultad de usar estos algoritmos con juegos tan complejos como el ajedrez (si os ponéis a calcular el espacio de posibilidades….) ha sido clásicamente un problema de memoria y de capacidad de computación no de que no exista esta solución.

El algoritmo básicamente funciona calculando el árbol del juego teniendo en cuenta todas las posibilidades posibles hasta una determinada profundidad (idealmente para resolverlo esta profundidad es hasta el final de la partida). Se calcula una valoración de la posición en cada nodo y entre las distintas se elige la de menor valor si la jugada la tiene que hacer el contrincante y la de mayor valor si es el propio. Sencillo ¿no?



El problema para aplicarlo al ajedrez es de dos tipos:

- Básicamente en el ajedrez la valoración debería ser 1 ganar, 0 tablas y -1 perder… Pero recordar que básicamente el ajedrez tiene muchas posibilidades (matemáticamente un espacio que crece exponencialmente) y el árbol es… Por lo tanto no se va a poder calcular hasta el final del juego y por eso lo que se hace es truncar el árbol a una profundidad y en vez de valorar con el resultado de la partida se valora con una función que evalué quien tiene ventaja, etc. Evidentemente la función a evaluar pues depende de las IAs y no son perfectas por esas cosas que conoce todo jugador de ajedrez de que a veces el valor de las piezas es relativo. A este problema lo llamaremos problema de la EVALUACIÓN. Aunque como habéis leído está emparentado con el problema del espacio de posibilidades.
- El problema del ESPACIO de POSIBILIDADES. Sólo en finales con pocas piezas se ha podido alcanzar soluciones basadas en este árbol de posiciones debido a que hay muchas jugadas y cada una de ellas abre otra infinidad de posibles jugadas, etc. Y aún así, en estos casos para alcanzar una valoración objetiva de una posición se requieren muchas jugadas y la valoración depende de un tiempo o movimiento con lo cual tradicionalmente las IAs de ajedrez no sabían jugar bien ni las aperturas (demasiadas posibilidades) ni los finales (demasiada profundidad) y bueno ni los medio juegos complicados.

Por ello para que jueguen mejor tradicionalmente se ha intentado añadirle libros de aperturas y tablas de finales. Esto, que no deja de ser lo

que en realidad hacemos los humanos aprendernos esos juegos y por memoria.., no deja de ser "tramposo" con los competidores humanos que no pueden tener una certeza cien por cien de recordar algo que para las máquinas de silicio está escrito en bytes. En el estado que vamos a intentar sacar las primeras versiones del ChessBattleRoyale se nos sale del scope.

Aparte se han intentado hacer variaciones del algoritmo minmax para simplificar el asunto (una de ellas la poda alpha-beta la usaremos en nuestras IAs, pero hay más como intentar evaluar primero las mejores jugadas candidatas con una cierto alcance de posibilidades no tan amplio, etc).

3.2.1.2 La poda Alpha-Beta

La poda Alpha-Beta consiste en intentar se basa en evitar el cálculo de ramas cuya evaluación final no va a poder superar los valores previamente obtenidos.

- α es el valor de la mejor opción hasta el momento a lo largo del camino para MAX, esto implicará por lo tanto la elección del valor más alto
- β es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, esto implicará por lo tanto la elección del valor más bajo.

Eso hace que el espacio a calcular se minimice porque muchas posibilidades se descartan automáticamente durante el proceso si salen de esos valores porque no van a mejorar el resultado ya obtenido.

Computacionalmente este algoritmo es en pseudocódigo:

```
función alfa-beta(nodo //en nuestro caso el tablero, profundidad, α, β,
jugador)
    si nodo es un nodo terminal o profundidad = 0
        devolver el valor heurístico del nodo
    si jugador1
        para cada hijo de nodo
            α := max(α, alfa-beta(hijo, profundidad-1, α, β, jugador2))
            si β≤α
                romper (* poda β *)
        devolver α
    si no
        para cada hijo de nodo
            β := min(β, alfa-beta(hijo, profundidad-1, α, β, jugador1))
            si β≤α
                romper (* poda α *)
        devolver β
(* Llamada inicial *)
alfa-beta(origen, profundidad, -infinito, +infinito, jugador_deseado)
```

Los resultadosse pueden mejorar ordenando el espacio de llamada (buscando las mejores jugadas posibles al principio (capturas, amenazas, cambios, etc.)) Para nuestra humilde SillyIA inicial no hemos tenido en cuenta todo esto.

3.2.1.3 Todo eso para el ajedrez pues vale bien nos lo creemos, pero, ¿y para el ChessBattleRoyale?

Pues las dificultades se incrementan…

Para empezar el espacio de posibilidades es mucho mayor (más piezas, más casillas, más jugadores). Y eso es lo que ha hecho que p.e. para el GO se hayan tenido que buscar otro tipo de algoritmos no MinMax sino p.e. una búsqueda MonteCarlo sobre el árbol de posibilidades. De hecho poner que profundice más de 2 jugadas (de todos los jugadores) tienta a la paciencia en el caso del ChessBattleRoyale tal y como está implementado actualmente.

Las funciones de evaluación también se complican porque el tablero cambia y casillas desaparece, porque hay que tener en cuenta más contrincantes, porque…

En resumen, que no estoy muy contento con mi SillyIA pero bueno es un primer paso.

Comentaremos las funciones para la evaluación empleadas cuando nos pongamos a comentar el código pero aunque hay que añadirle más factores lo que he intentado añadir es un peso distinto de las piezas según su distancia al borde que peligra del tablero (esto hará que la IA intente luchar por el centro como un buen BattleRoyale), según la movilidad que tengan (un alfil encerrado tiene mucho menos valor que un alfil capaz de recorrerse todo el tablero), según la pieza en sí.

Muchos de estos factores de las funciones se han tomado en una primera aproximación como constantes más o menos tomadas de sus equivalentes ajedrecísticos normales pero no me cabe la menor duda que serán parámetros a optimizar (a la "machine learning") en un futuro espero que no muy-muy lejano.

3.3 Escritura (Expansión cercana)

> *Si el cielo fuera papel y el océano un tintero no me alcanzaría para escribir las partidas, que jugar, yo quiero.*
>
> ***Andrea Lambrecht, Argentina***

Es muy importante que las partidas se puedan guardar como en el ajedrez en papel con el objetivo de poder reproducir partidas y que el juego siga teniendo un carácter deportivo. Esto ayudará además de para hacer replays para cuando haya que mejorar la IA y para tutoriales, guías, etc. y cuando haya que ponerse con guardar datos tanto de las partidas como de los jugadores.

Creo que se puede realizar mediante una simple transcripción de Escritura Algebraica a-lo ajedrez pero ampliada para tener en cuenta que el tablero es más grande y que hay cuatro jugadores. Pero en principio como sigue siendo un sistema 2D con casillas nombrando la casilla inicial y la final y un sistema para indicar cuando el tablero va disminuyendo o cambiando sus colores estaría todo hecho.

En cuestión funcional lo que vamos a requerir a la aplicación en sí es que además del modo de juego exista un modo Replay o de Repetición que ha de ser capaz de leer partidas anotadas y simplemente ir dejando reproducir las jugadas en un tablero.

Estas "plantillas" de partidas además serán la base para un guardado y persistencia de las partidas. Quizás por su estructura lo mejor será establecer las partidas como documentos json.

Nota para futuras ampliaciones: Y a la hora de realizar la ampliación del juego a multi-jugador habría que ver si es más rentable ir pasando la notación de la partida más los detalles de los relojes y jugadores o ir pasando el tablero. Quizás es mejor para un juego por turno con notación pero todas estas cosas se discutirán en su momento, sólo las incluyo aquí por reflejarlas como futuras vía de expansión.

3.4 Funciones preliminares de BD (Expansión futura)

> *El producto más valioso que conozco es la información.*
>
> ***Gordon Gekko,** Wall Street*

Uno de los requisitos para futuras expansiones será establecer como guardar datos de jugadores y de sus partidas en sus correspondientes sistemas bases de datos tanto para uso interno como para el usuario final.

No creo que sea necesario llegar al nivel de una base de datos ajedrecística como ChessBase (y menos en los primeros estadios de desarrollo) pero de alguna manera el programa debe poder recopilar, guardar y manejar tanto datos de las partidas en sí (con su notación) que nos vendrá bien para implementaciones de replays, guías, tutoriales, entrenamiento, etc. como datos de jugadores y datos administrativos para p.e. la realización de rankings y sistemas ELOs para relacionar la fuerza de los jugadores para competiciones y demás, o el acceso de futuras funcionalidades de notificaciones, competiciones, comunidad, etc. que se establezcan.

Para un guardado de las partidas quizás un formato json junto alguna base de datos no relacional como mongodb podría ser la solución más natural y quizás para los datos de los jugadores una base de datos relacional bastante sencillas sea la mejor posibilidad.

3.5 EXPANSIONES

> *El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad.*
>
> ***Victor Hugo***

Todas estas expansiones no estarán en este proyecto inicial pero quiero dejar constancia en este apartado sobre la funcionalidad de la aplicación algunas posibilidades que van a quedar momentáneamente en el tintero pero que se necesitarán hacer para cumplir los objetivos futuros y empresariales de la aplicación:

- Multi-jugador
- Puntuación/Ranking ELO
- Modo campaña con puzles
- Editor de problemas/estudios
- Acceso a información de guías y/o manuales dentro de la aplicación.
- Notificaciones de retos. Calendarios de competiciones.
- Creaciones de clubs y socialización.

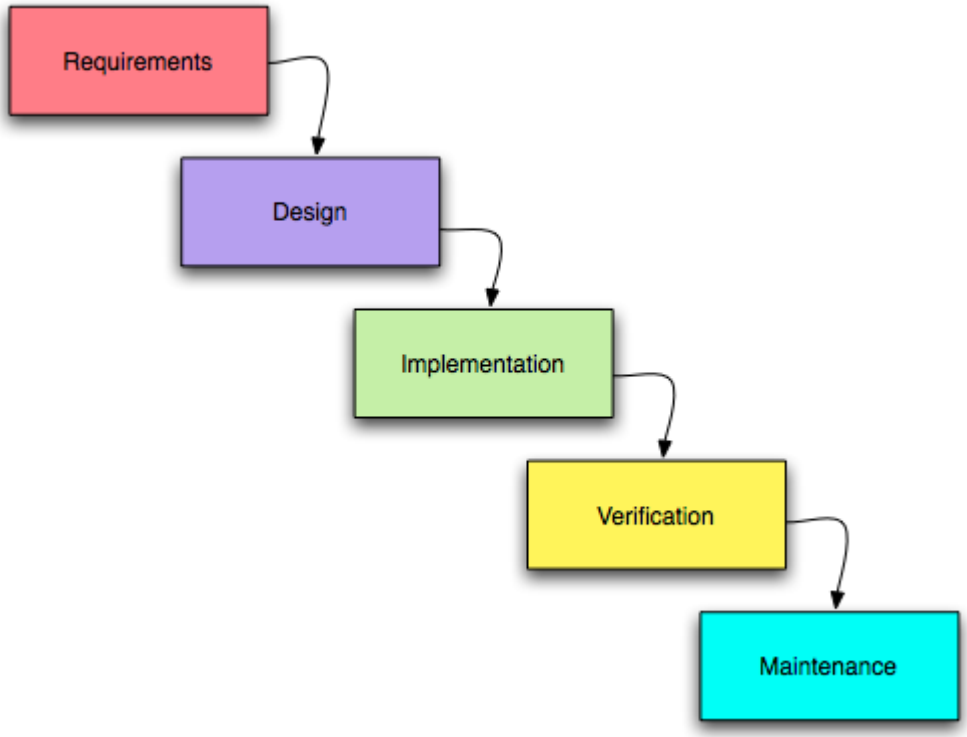
4 CICLO DE VIDA Y ESTRUCTURA DEL PROYECTO

4.1 CICLO DE VIDA

> *Cada momento de la vida nos ofrece algo diferente y, si no atravesamos los ciclos vitales cuando corresponde, podemos desarrollar un sentimiento de no pertenencia.*
>
> ***Helena Lopez-Casares***

Una vez se tienen claro los requisitos en un ciclo de vida clásico (o de cascada) viene que debería procederse a una fase de análisis y diseño. Así que ahora debería venir un capítulo 4 dedicado a ello. Permitidme que incluya aquí un breve apartado para discutir sobre los ciclos de vida del software y lo que he intentado ejecutar durante el desarrollo de este proyecto.

Tampoco quiero hacer una discusión teórica detallada. De hecho dado que el caso que nos ocupa (al menos inicialmente) es un proyecto en el que las especificaciones del "usuario" (y el usuario en sí) es uno mismo al ser el propio autor, las principales desventajas del método en cascada tradicional pues no afectan (los requisitos en principio debería el programador conocerlos perfectamente y la comunicación con uno mismo debería ser como mínimo sencilla) y, sin embargo, todas las ventajas de este ciclo teórico sí que están (un mayor control de la programación del desarrollo). Pero…



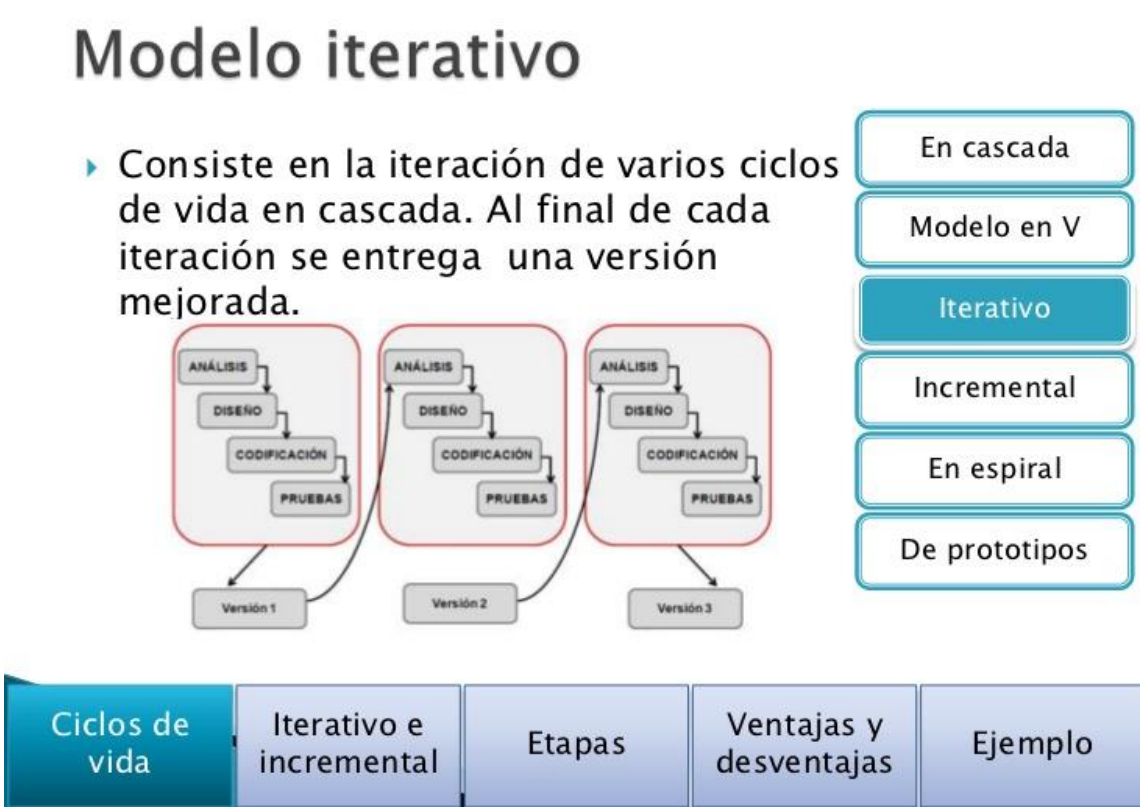
Tal vez yo sea el raro (o simplemente el inexperto) que al proponer este tema no sepa exactamente todos los requisitos que podré implementar en la fecha de entrega y cuáles no, aparte que cuando empecé con el proyecto tampoco tenía (ni tengo) completamente claro todo lo que pretendo que tenga el programa. Por otra parte, el usuario real no soy yo, sino que espero que sean los futuros jugadores de Chess Battle Royale, con lo cual tampoco sé enteramente lo que ellos pueden querer o no de la aplicación.

Por esos y otros motivos me ha parecido más interesante no intentar usar ese ciclo de vida clásico y, en cambio, usar algo más parecido a un ciclo de vida incremental e iterativo donde primero intentaré realizar unas versiones del Chess Battle Royale en sí e ir añadiendo funcionalidades y retoques hasta ir acercándonos a un producto final que sí que tuviera todas las funcionalidades y características deseadas.

Cuando llegue la fecha de entrega del trabajo veremos a que versión del software habremos llegado pero en principio las distintas versiones que se quieren realizar, con la funcionalidad más importante añadida, son:

- Versión Alpha-0 (Ruy López de Segura):
  - Estructura del tablero, piezas, movimientos, reglas del juego.
- Versión Alpha-1(Alessandro Salvio):
  - IA primitiva. Introducción del juego singleplayer.
- Versión Alpha-2 (Gioacchino Greco):
  - Partida visible. Todos los jugadores en la misma pantalla sin control de quien…
- Versión Alpha-3 (de Kermeur):
  - Notación: replays de partidas, guardado de ellas en formatos y BD de ello.
- Versión Alpha-4 (Philidor):
  - Screens de configuración de partida y login de jugadores (No multiplayer solo modificación de todas la interfaces de usuario para incluirlas en un futuro)
- Versión Alpha-5 (Louis de la Bourdonnais):
  - Inclusión de mayores funcionalidades de base de datos de partidas
- Versión post-Alpha-6 (Howard Staunton):
  - Primer esbozo de multijugador.
- Versión post-Alpha-7 (Adolf Anderssen):
  - Versiones y retoques para otras plataformas. Mejoras en todos los aspectos.
- Versión Beta (Paul Morphy):
  - Previa a cualquier versión estable, la primera para ser lanzada realmente.

Si para la fecha de entrega hemos llegado a la versión Legall o Philidor… estaré más que contento.



También hay que tener en cuenta que aunque intente partirlo por funcionalidades es más que probable que tenga que hacer rediseño de las anteriores durante alguna versión así que nunca será una forma de ciclo de vida prefecto y se mezclará según vea más factible.

Todas estas versiones alpha no verán la luz pública pero quería que estuviese aquí puesto el "organigrama" de como pensaba ir desarrollando como parando en cada una de ellas para un testeo más intensivo y un rediseño de requerimientos y diseño de la siguiente etapa.

Tampoco documentaré todas unas de estas versiones intensivamente por no convertir este documento en un mamotreto pero sí que dejaré algunos de los apartados que personalmente considero más interesante de algunos de ellos para los apéndices.

Salvo una excepción que creo que es lo suficientemente interesante como para tener su propio (el siguiente) capítulo.

4.2 Camino a la versión Ruy López de Segura

Voy camino Soria ¿Tú hacia dónde vas?

Allí me encuentro en la gloria Que no sentí jamás

Gabinete Galigari

Los detalles de análisis y diseño los dejaré ya unificados con otras versiones en el capítulo 6; ahora sólo quiero describir como he estructurado el proyecto. Además aunque las herramientas y plataformas utilizadas quizás correspondieran al capítulo de codificación, al no ser de la aplicación en sí sino que he decidido usarlas para enriquecer aspectos del proyecto pues creo que mejor les dedico este apartado.

4.2.1 Prototipado

Cuando se hace el análisis de requisitos (la primera etapa de cualquier ciclo de vida) tener un prototipo ayuda a probar la funcionalidad y la usabilidad del sistema antes de empezar las etapas de diseño y de desarrollar el programa en realidad. De esta forma se puede conseguir una mejor comunicación a la hora de presentar los diseños o incluso ahorrar tiempo y recursos dado que es significativamente más barato de desarrollar un prototipo que el producto final para ver si es lo que se quería (entre otras cosas te ahorras de programar todas las capas necesarias o tener datos completos, etc., etc.).

Este prototipado puede servir no sólo para la construcción de las interfaces de usuario sino que puede ayudar a la codificación en sí. De hecho en la versión α-0 no va a tener interfaz de usuario porque lo que quería era ir programando las distintas clases a emplear comprobando que funcionen simplemente escribiendo strings pero el diseño del prototipo del juego me fue totalmente necesario para realizarlo.

¿Y cómo puede ser el prototipo de esta versión sin interfaz gráfica? Pues nada más y nada menos que el juego en físico. Esta es una de las ventajas de partir de un juego de tablero. Se puede fabricar un prototipo y probarlo en vivo con gente y posibles jugadores para que te vayan dando un feed-back sobre las posibilidades del juego.

Y eso es lo que hice (bueno mandé hacer). Construí un Chess Battle Royale con cartón, chapas, pegamento, papel, rotuladores, una buena dosis de paciencia que tuvo que poner mi novia Lara sin la cual nada de esto sería posible. De hecho si habéis leído el documento ya lo habréis visto pues es la primera imagen que ilustra (tras la de la portada claro está).

La utilidad de este prototipo ha sido impresionante ya que además de su utilidad tradicional ha servido para comprobar distintos problemas a la hora de realizar la anotación de la partida, como eso afecta con la desaparición de casillas, visualización mental que ha llevado a modificar los movimientos tradicionales de las piezas que ya hemos mencionado, etc.

Además, ha sido divertido.

Y como extra (aunque tenga que ver poco con el ciclo, sí que tiene que ver con el mundo del videojuego en general) pues el documento histórico de cómo se creó ese prototipo está en (insertar el link a la página). Y esto puede parecer una tontería, pero es una característica general de todos los juegos

en la actualidad que vayan poniendo en internet (en diversas plataformas donde todas las redes sociales están alcanzando una gran importancia para este tipo de comunicaciones) los llamados blog de desarrollador. De hecho creo que hoy en día, esta clase de detalles para crear "hype" o de comunicación de los desarrolladores con los usuarios finales (más conocidos como "gamers") tiene una gran importancia.

Evidentemente, no voy a hacer un desarrollo completo de todas estas posibilidades externas que estarían dentro del desarrollo de los objetivos empresariales (más que nada por falta de tiempo por falta de tiempo para poder realizar el resto de tareas dentro del proyecto) pero como ejemplo me remito a ese vídeo.

Y además me hizo ver, algo que sí que tiene su interés en cuestión informática, que como todo buen juego su presencia en la red. Así que estaría bien el desarrollo de su página oficial del proyecto además que así también quedaría reflejado parte de lo aprendido sobre el ciclo sobre lenguajes de marcado.

De nuevo por demasiada amplitud del "scope" del proyecto (recordemos que según los recursos humanos que creo que se necesitarían para llevar a cabo hasta el final el proyecto necesitaría a una persona cuya responsabilidad fueran estas tareas) tampoco puedo llevar a cabo eso en su mayor extensión y la que creo que sería su mejor implementación pero mientras resolvía otra NECESIDAD (y con mayúsculas) para realizar el proyecto pues creo que distintas piezas encajaron…

Antes de cambiar de apartado me gustaría añadir que otras pantallas que sirvieron como prototipo de

4.2.2 Control de versiones

Usar un sistema de control de versiones es imprescindible cuando se realizan proyectos en equipo. En este caso al trabajar de una forma individual pues las ventajas son menos aparentes. De todas maneras creo que es una buena idea usarlo porque permite una mejor organización, sirve hasta en un momento dado para tener copia de seguridad, ayuda a acostumbrarse a usarlo y a aprenderlo. La plataforma escogida ha sido github [23] que usa git [24].

Aviso para que no os asustéis si intentáis mirar el repositorio: Aunque la forma habitual es usar una rama para las modificaciones e ir usando la main para versiones más o menos estables (o que al menos compilen), al trabajar de momento yo sólo pues no he seguido ese convenio sino que simplemente estoy poniendo todo en la rama principal y si el commit del día no ha llegado a tener algo decente pues así se ha quedado.

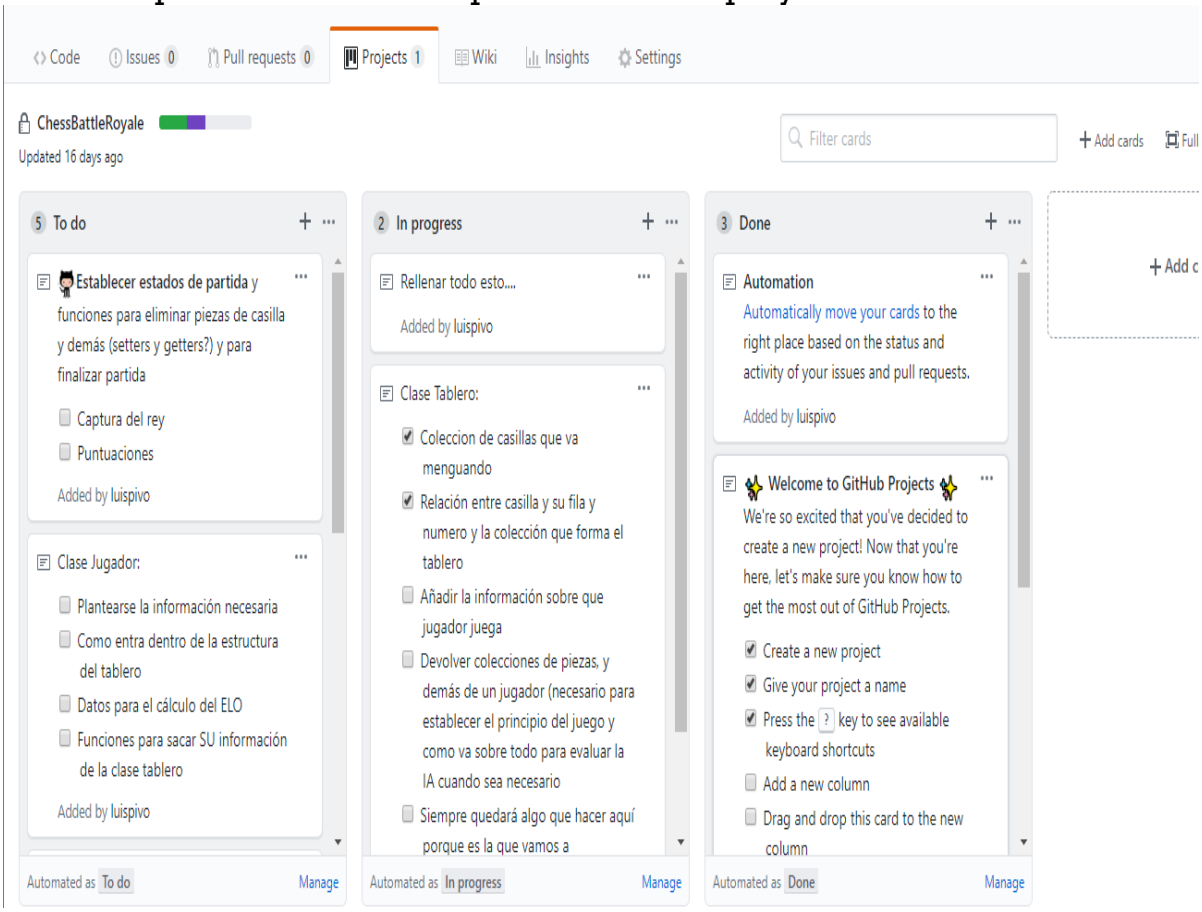
Espero que a fecha de finalización del proyecto "congelaré" ya una versión más o menos estable en la rama principal y ya crearé mis propias ramas para las "expansiones". Pero de momento seguiré trabajando así por

comodidad o pereza y más teniendo en cuenta que en teoría estoy manteniendo el repositorio como privado gracias a ser la cuenta educativa del instituto.

Pero github es algo más que un lugar para tener una copia descargable y un histórico de versiones para poder volver a otros momentos. No he usado (y menos bien) muchas de sus características (algunas de las cuales por la cuestión clara de que está pensado para el desarrollo en equipo y claro cuando el equipo es uno… pocos pull request, colaboradores, etc puedo usar). Pero sí que he explorado algunas de sus funcionalidades.

Tiene la posibilidad de crear wikis (de hecho en algún momento crearé unas pocas páginas para que sirva como wiki del proyecto pero no está en el calendario).

Crear proyectos. Que te permite de una manera sencilla hacer paneles Kanban para ir haciendo una planificación del proyecto dividiendo las cosas



en tareas para hacer, en progreso, realizadas (o los paneles Kanban que desees). Puede parecer sencillo (lo es) pero eso permite tener anotado lo que hay que hacer y comenzar a planificar y organizar el trabajo en equipo. También se puede establecer issues (problemas, bugs, tareas a realizar…) y asignarlas a un colaborador, configurar su prioridad, color, etc. Realizar tags releases (etiquetar las versiones si han de poder bajarse).

No he usado casi nada de esto al ser de momento todo una tarea individual (así que lo he usado brevemente para auto-organizarme pero poco

más). Pero quiero destacar la potencia de la plataforma en todos estos menesteres y recomendaría ahondar e investigar en las distintas posibilidades que se ofrece con github.

La herramienta que sí que he usado más en la realización de este proyecto es GitHub pages. GitHub pages es un servicio libre de hosting estático diseñado para albergar páginas web directamente de un repositorio GitHub tanto para la organización, un proyecto/repositorio o una página personal.

Que sea estático es una desventaja pues no soporta ni permite el uso de código de servidor como PHP, Ruby o Python. Pero a cambio de esto permite publicar rápida y sencillamente contenido web con el repositorio integrado.

Para ir publicando material he usado lenguaje de marcado markdown y html en algún caso. Probando en local gracias a Jekyll (que a su vez requiere la instalación de Ruby y de algunas de sus gemas). Pero vamos así dicho suena mucho más difícil de lo que es; y al final creo que podré enlazar una página sobre el autor (donde prácticamente será un curriculum vitae) con la página del repositorio para poder descargarte el proyecto, leer algunos artículos sobre el Chess Battle Royale, y donde además le adjuntaré la documentación creada con javadoc de las clases que se vayan realizando.

Invito a verlo todo en https://luispivo.github.io/ChessBattleRoyale/. Aunque haya elegido un tema soportado directamente es bastante sencillo modificar los css y hacerlo a tu gusto. Como siempre cuanto más azúcar más dulce. Pero creo que como principio para diseñar y crear un sitio web queda más que aparente.

Con todo ello tengo centralizado y al alcance de cualquiera tanto el código fuente del proyecto, la documentación del proyecto (tanto del código como del proyecto en sí como diagramas que se realicen y todo lo relacionado), páginas web para dar información a los usuarios (tutoriales, guías, enlaces, lo que se nos ocurra…). ¿Se puede pedir algo más?

Es evidente que para un nivel comercial necesitaría otro tipo de herramientas (para p.e. tener código en el servidor para sacar lista de rankings y demás) pero al nivel al que se encuentra actualmente el proyecto…

5 Análisis y diseño

5.1 Diagrama de arquitectura

5.2 diagrama de casos de uso

5.3 diagrama de clases

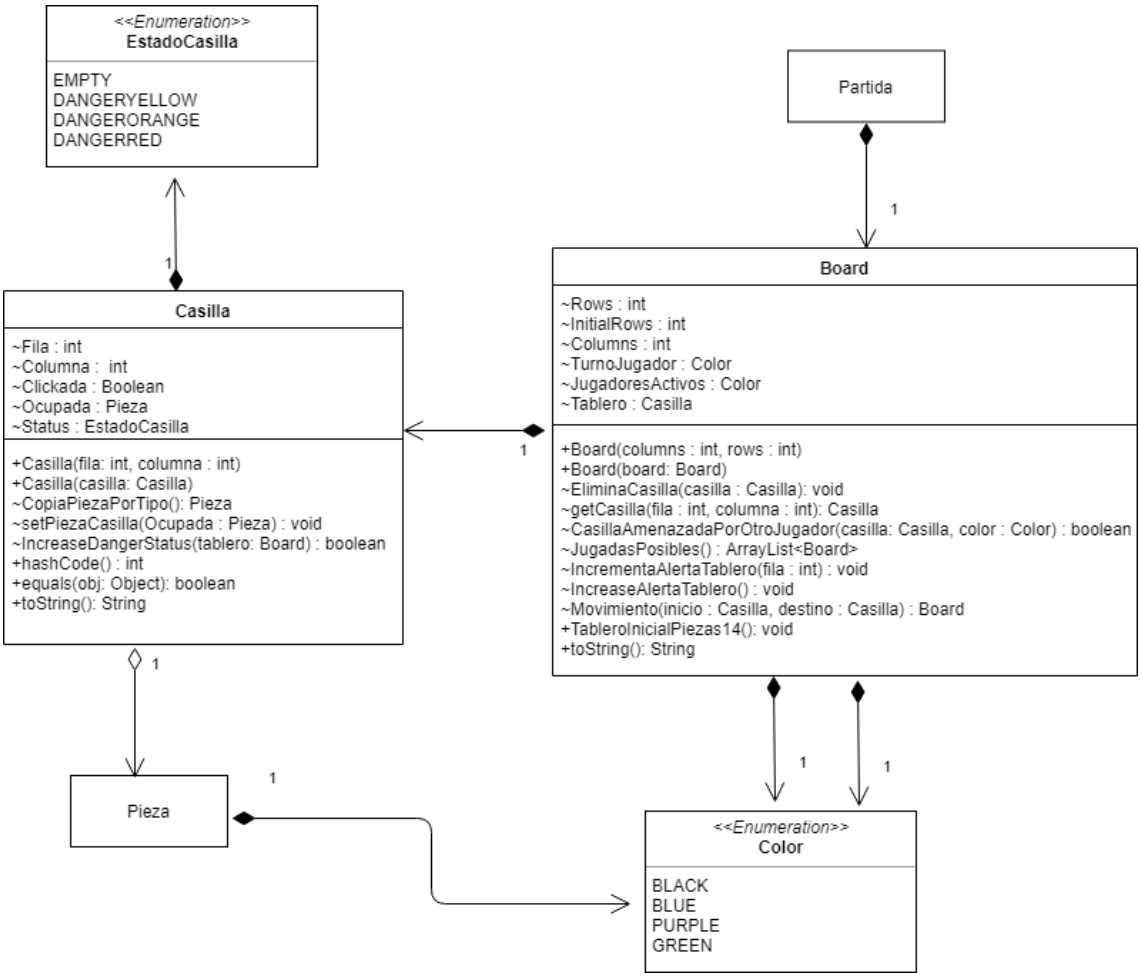
5.3.1 Clases Board y Casilla

La clave de mi implementación será la clase Board.

Aunque el nombre no le hace justicia porque aparte del tablero en sí (como ArrayList de Casillas) contendrá casi toda la información para reflejar un instante del juego en sí (p.e. el tamaño del tablero inicial y actual, el turno del jugador al que le gusta jugar, los jugadores que permanecen activo, el tiempo de cada jugador. De esta manera estará preparado para cuando tenga que hacer la versión multiplayer sea tan sencillo como mandar este objeto a todos los usuarios para actualizar el estado de la partida (evidentemente cada uno de los usuarios ya tendría los datos "estáticos" (por llamarlo de alguna manera ya que no van a variar a lo largo de una partida y han podido "transmitirse" al principio) correspondientes a los jugadores).

Una de las cosas que tengo dudas sobre el esquema es donde pongo el tiempo… En principio tendría que ser 4 valores (1 por jugador) pero también en el ritmo Bronsteiniano lo mismo necesito por jugada… y aparte tienen que tener acceso los 4 jugadores así que parece que lo ideal es tenerlo en Board también pero así junto con casillas y demás … El problema de implementarlo en las primeras versiones es que las IAs primitivas tardan demasiado tiempo así que he preferido no añadirlo y centrarme en probar otros factores que en añadirlo para deshabilitarlo para poder probarlos.

Otra faceta que surgirá en futuras expansiones es que a la hora del juego multiplayer si mando el tablero es más "seguro" porque todos tienen la misma información y estaría toda contenida en él pero lo mismo es más sencillo solo enviar notación y tiempos que ese objeto que a su vez tiene chorrocientas casillas, etc., lo que consumirá menos ancho de banda y sería más sencillo de enviar al resultar simplemente una string y varios timestamp. ¿Qué será mejor?



5.3.2 Pieza

Las piezas en este diseño son un poco peculiares. Para empezar, no tienen en realidad contacto con ningún campo de jugador directamente (sólo indirectamente a través de la enumeración Color) siendo su enlace más fuerte con la propiedad Ocupada de la clase Casilla.

Otra característica es que todas las piezas heredan de una clase abstracta Pieza. De hecho en la implementación de estas clases no tienen atributos especiales sino que es la clase abstracta la que tiene los dos atributos (Color y TipoPieza).

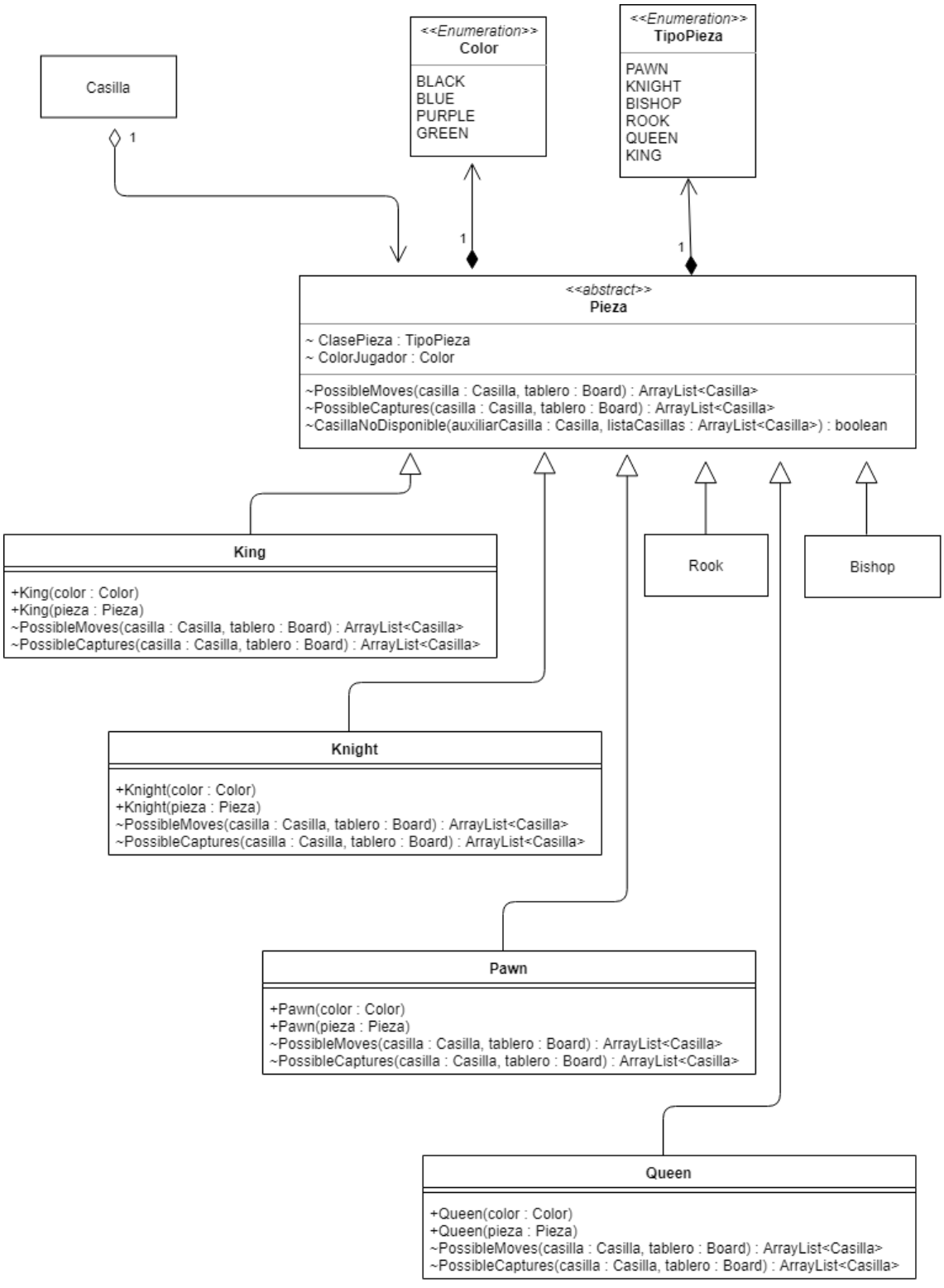
Dudé mucho si hacerlo así o intentar implementar una interface pero creí que la clase abstracta era la mejor opción para poder trabajar de una manera genérica con todas las piezas y servir de patrón para todas las piezas (las actuales del Chess Battle Royale y por si posibles expansiones).

Sin embargo sí que hay una cuestión que no me acaba de gustar de como lo he implementado. Es una cuestión con uno de los métodos que tiene que sobre-escribir cualquier pieza al definirse como abstractos.

Los métodos abstracto de la clase Pieza son PossibleMoves y PossibleCaptures. Para la mayoría de las piezas son dos métodos que serían equivalentes y sería tonto tenerlos pero luego tenemos una pieza como el

peón que no se mueve igual que captura y luego necesitamos solo saber hacia dónde captura para comprobar jaque-mates así que he tenido la necesidad de medio "duplicar" código en algunos casos (quizás haber hecho una interfaz para calcular solamente las posibles capturas hubiera sido más elegante pero es un precio que de momento prefiero pagar para no complicar la "lógica" del programa). También el rey necesita de esta pecularidad al no poder moverse donde se sienta amenazado.

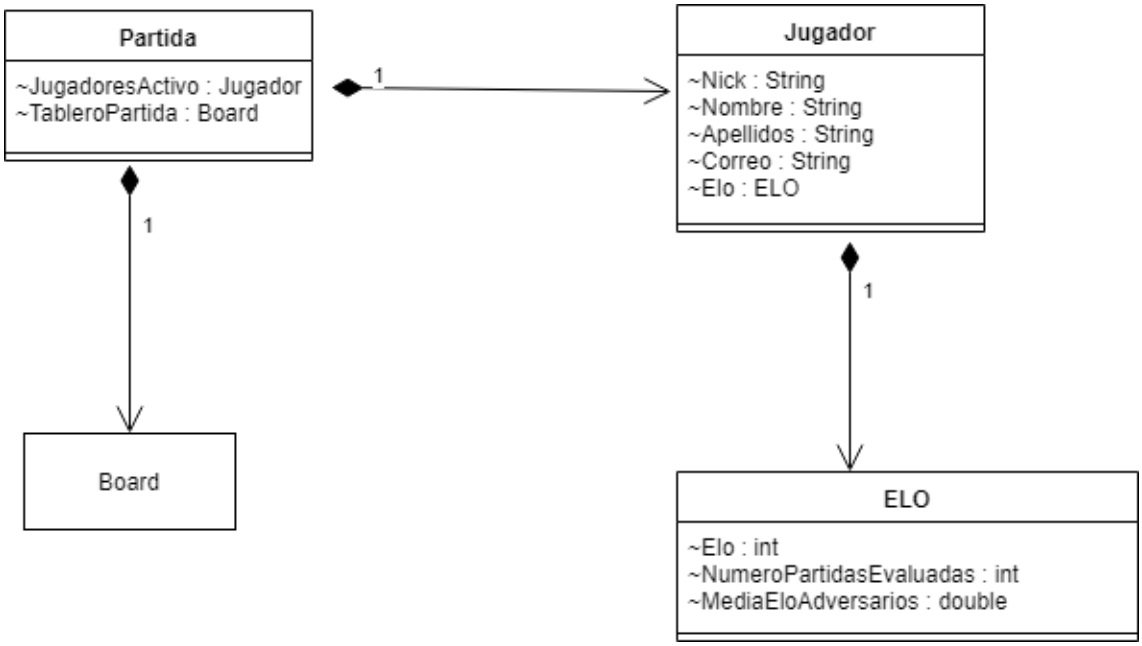
CasillaNoDisponible es un simple metodillo que se usa en todas para ver si la casilla propuesta de movimiento se sale del tablero. Tuve dudas al implementarla ahí o en la clase casilla donde quizás tiene más sentido pero como su única utilidad es con el movimiento de las piezas que no se usa para más cosas creo que este es el lugar.



5.3.3 Partida y jugador

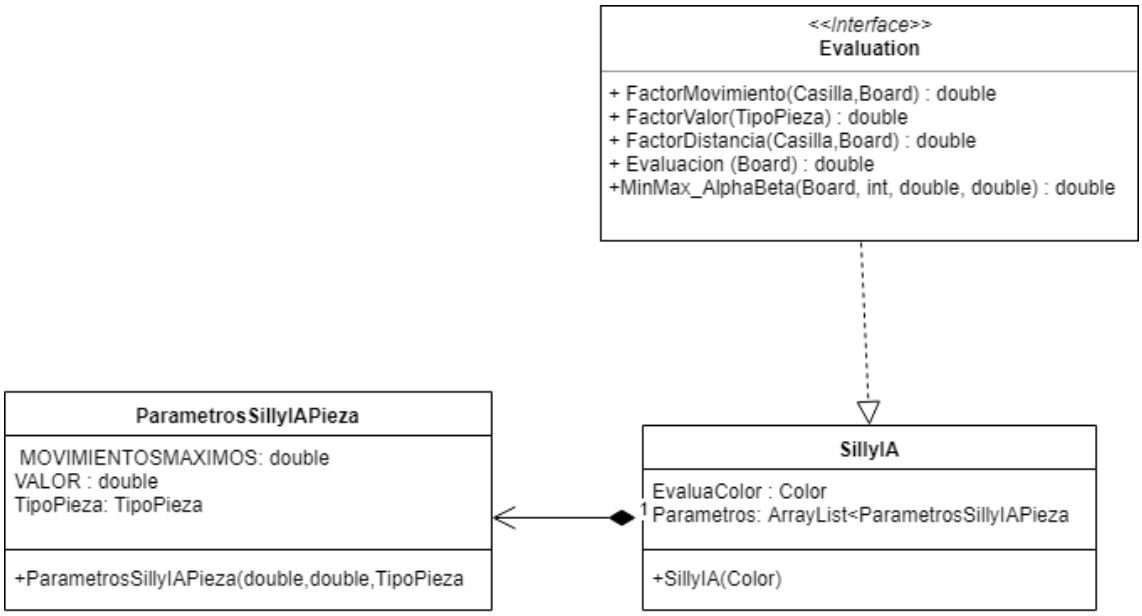
En partida enceraré toda la lógica que controle el flujo de la partida en sí. Tanto la creación de la partida con los jugadores, creación del tablero, flujo de movimientos, funciones para calcular situaciones especiales, puntuaciones, etc.

Es la parte que seguramente me tocará modificar más de la versión α, pero aún así quiero indicar aquí el diagrama de clases inicial para representar como está estructurado y modularizado el programa.



El Jugador tendrá unos campos de tipo jugador para las IAs.

5.3.4 Clases de IAs



Estructura preliminar para las IAs. Debera cambiar para que haya más funciones dedicadas a optimizar los parámetros que aparecen tanto en la clase "POJO" de parámetros ParametrosSillyIAPiezas como los que aparecen propio de las funciones de la interface que se ajustan en su implementación en la clase IA de verdad.

Todo ello en principio irá como una de los campos de jugadores cuando este jugador no sea humano.

5.3.5 Esquemas y clases propias de LibGdx.

Aunque el proyecto esté desarrollado en LibGdx y parte del objetivo es utilizar esta librería y las herramientas que nos brinda no quiero hacer una disertación teórica de esta librería sino dar una serie de pinceladas de como se ha implementado para el proyecto.

Se ha intentado seguir un esquema propio de Scene2D donde en la ventana se crea escenarios (stage) que se encargan de dibujar y ver cómo actúan una serie de actores.

Cuando uno piensa en un juego de ajedrez o de tablero tiende a pensar que el tablero sería como la escena y los actores serían cada una de las piezas. Posiblemente esa estructura en mente sería la más adecuada para realizar animaciones de las piezas, etc. Pero he decidido que la mejor aproximación era considerar a la clase casilla como la principal clase de actor porque hay que recordar que según la lógica que hemos programado es la casilla la que contiene a las piezas y a la que le "ocurren" las cosas (desaparece, cambian de pieza, de peligro, etc). Otro actor será el tablero porque ahí es donde finalmente estarán contenida el resto de información correspondiente a los jugadores y al turno de juego, el tiempo del que disponen, etc.

Todo eso hace necesario modificar esas clases para que extiendan de actor y colocar ahí como se van a ir dibujando y actuando. Además al usar un asset manager para reducir la carga de las imágenes y assets del juego nos llevó al problema de ir introduciendo este asset manager para ser alcanzable por los actores.

Honestamente creo que tal y como lo he codificado deja algunas lagunas y mejoras posibles porque al ser añadido como una especie de parche a las implementaciones originales de la lógica del ajedrez pues…

En dichas clases (Board y Casilla) se han añadido una serie de métodos necesarias para esta implementación gráfica pero que en general han sido relativamente fácil ya teniendo en cuenta las que costaron las del paquete original LogicaInicial.

No comento mucho más sobre el diagrama de clases de esta estructura de LibGdx porque no es algo original del proyecto (Un juego que tiene una serie de screens que a su vez tiene scene que contienen actores que al fin y al cabo actúan y se dibujan; los inputprocesor para interactuar con la escena, etc).

5.4 Diseño de datos (Expansión)

6 Codificación

6.1 Lenguaje, herramientas y entornos de programación

El lenguaje principal usado durante la realización del proyecto es Java. La elección de este lenguaje está basado en la capacidad de ser multiplataforma al ser desarrollado sobre la Java Virtual Machine y disponer de unas buenas librerías para facilitar el desarrollo del entorno gráfico del proyecto. Aparte es uno de los lenguajes básicos usados durante los curos correspondientes al Ciclo. Como librería principal se ha usado LibGdx y sus potentes implementaciones para diseñar juegos. Y tras hacer lo que he hecho y teniendo en cuenta el énfasis en la IA, que se ha de expandir a multijugador, y diversas cosas me pregunto si no hubiera sido mejor empezar con otro lenguaje.

Como todo proyecto LibGdx usa Gradle para la paquetación de las dependencias y he codificado generalmente en NetBeans (tanto la versión 8.2 como la NetBeans Apache 11.0). Usando JavaDoc para la elaboración de la documentación de las clases realizadas.

Como control de versiones he usado git y github como repositorio lejano. Usando los servicios de GitHub-pages (y por consiguiente lenguaje de marcado MarkDown, Jekyll como servidor estático de páginas web teniendo que instalar un entorno de lenguaje Ruby para su instalación. En un futuro quiero personalizar más esta parte cambiando css y htmls aparte de estas capacidades de GitHub-pages.

Para la elaboración de este documento he usado (a mi pesar) Word y sus plantillas (prometo que será la última vez por no atreverme a recordar cuando hacia las cosas bien en LaTeX, por otra parte las pocas gráficas usadas para la visualización de las funciones de la IA con Excell han sido una pesadilla que no quiero recordar), Draw.io para la elaboración de los diagramas, tokentool para la elaboración de las imágenes del juego en sí (parte de un proyecto OpenSource de rptools), las imágenes de este documentos han sido tomadas de fuentes con licencia GNU o Creative Commons).

6.2 Aspectos relevantes de la implementación

6.2.1 Lógica del ChessBattleRoyale en sí

Es la parte que en realidad se ha llevado el peso del trabajo y con la que más a gusto me he llevado. Creo que ha quedado bastante reducida y legible. Los aspectos de diseño de la clase en sí pueden verse en los

diagramas así que intentaré no repetir lo dicho allí y centrarme más en algunas pecularidades de algunos métodos de las susodichas clases.

6.2.1.1 ArrayList de Casillas vs array

En el ajedrez "convencional" el tablero es fijo (no cambian las casillas) con lo cual se puede usar más fácilmente cualquier estructura de los datos (matrices p.e.) que al ser fijas facilitan el acceso en memoria directo a dichas casillas. En cambio en el ChessBatleRoyale el conjunto de casillas del tablero va cambiando con lo cual se requiere que además de las dimensiones iniciales del tablero (InitialColumns e InitialRows) sea necesario saber que casillas en cada momento están disponibles.

Para implementar esto se ha establecido un ArrayList de Casillas que son las que forman el tablero. Esto hace un poco más complejo el acceder a las casillas determinadas (sacarlas y ponerlas en el ArrayList) si las consideramos de una forma "tradicional" de por filas y columnas. La solución para esto es el método getCasilla:

```
    Casilla getCasilla(int fila, int columna){
        if ((fila>=(InitialRows-Rows)/2 && fila
<(InitialRows+Rows)/2)&&(columna>=(InitialColumns-Columns)/2&&
columna<(InitialColumns+Columns)/2)){
            int index=Rows*(fila-(InitialRows-Rows)/2)+columna-
(InitialColumns-Columns)/2;
            return Tablero.get(index);
        }
        else return null;
    }
```

Este método simplemente relaciona el las filas y columnas con el indice que tiene en el ArrayList para todos los tamaños iniciales como los actuales y si nos hemos salido del tablero devuelve un null que nos permitirá fijar los límites del tablero.

La sencillez de la expresión no debe confundir porque para que funcione ha tenido que conjuntarse con la forma de crear el ArrayList de Tablero dentro de la clase Board y no ha sido tan sencilla de sacar a la luz.

Aquí diría unas palabras sobre Java y los nulls y de como esto puede originar dolores de cabeza (lo hizo) luego pero en el momento creí que era la forma elegante de ir borrando casillas innecesarias y de tener el tablero completamente acotado aunque fuésemos quitando casillas y podía dar lugar a otras implementaciones cuando se quisiera cambiar el tablero de otra forma. A posteriori quizás hubiera sido más sencillo tomar unas dimensiones fijas y "marcar" de alguna manera (y no con null) las casillas que se van eliminando dentro de la enumeración que veremos en el siguiente apartado.

6.2.1.2 Eliminación de borde del tablero

Hemos implementado una enumeración que indica cada casilla su estado de "peligro" (enumeración EstadoCasilla). A esta enumeración se pueden añadir cosas pero para la implementación que nos ocupa simplemente hay un incremento del grado de alerta desde Empty(sin

peligro)->Amarillo->Naranja->Rojo->eliminar la casilla. Para hacer este paso en la clase tablero se han incorporado una serie de métodos.

El primero de ellos simplemente calcula el número de filas al que hay que cambiar de estado que depende del color de la casilla en cuestión (si es al inicio solo hay que cambiar 1 fila y ponerla en amarillo, si este perímetro ya está amarillo habrá que cambiar esa fila y ponerla en naranja y la un poco más interna ponerla en amarillo y así sucesivamente).

Esto se implementa simplemente con un switch case para calcular el número de filas y luego llamando al que aumenta el peligro a las filas correspondientes y luego calcula si algún jugador se ha visto eliminado de la partida por la desaparición de parte del tablero (método ColorSeVaDePartida que ya discutiremos posteriormente)

Un detalle importante es el orden que se llaman las filas para ir incrementando su estado de peligro (no es conmutativo) ya que el número de filas y columnas puede cambiar.

```
void IncrementaAlertaTablero(){
        Boolean cambiarTurno=false;
        int numeroFilas=0;
        switch(Tablero.get(0).Status){
            case DANGERYELLOW:
                numeroFilas=1;
                break;
            case DANGERORANGE:
                numeroFilas=2;
                break;
            case DANGERRED:
                numeroFilas=3;
                break;
        }
        for (int i=numeroFilas;i>=0;i--) IncrementaAlertaTablero((InitialRows-
Rows)/2+i);
        for(Color x:JugadoresActivos) if (ColorSeVaDePartida(x)) {
            EliminaPiezasJugadorEliminado(x);
            cambiarTurno=true;
        }
        if (cambiarTurno) EliminaJugadorYPasaAlSiguienteJugador(false);
    }
```

De esta manera IncrementaAlertaTablero() calcula y llama a IncrementaAlertaTablero(fila) para que el bucle de este elimine del ArrayList de tablero las casillas que pasan de rojo a desaparecer y cambian los valores de Rows y Collumns respectivamente si se ha eliminado parte del tablero (se refleja en la booleana eliminarParteTablero).

Para ver más detalles lo mejor es simplemente ir al código fuente…No voy a copiar aquí todo ese código fuente (el if para comprobar si la casilla está dentro del perímetro al que hay que aumentar el peligro es un tanto enrevesado si se lee únicamente en un editor de texto ya que se basa en la geometría del tablero y del ArrayList que lo contiene). Pero creo que el siguiente esquema ayuda a entenderlo:

```
    void IncrementaAlertaTablero(int fila){
        boolean eliminarParteTablero=false;
        for(int i=0;i<Tablero.size();i++){
            if ("Casilla está dentro del perimetro de la fila a incrementar el
peligro")
                if(Tablero.get(i).IncreaseDangerStatus(this)){
```

```
                        eliminarParteTablero=true;
                        i--;
                }
        }
        //Se eliminan 2 filas y columnas
        if (eliminarParteTablero) {
            Rows-=2;
            Columns-=2;
        }
    }
```

Siendo IncreaseDangerStatus un método de la clase Casilla que básicamente aumenta el status de esa casilla (devolviendo false) o devuelve true si la elimina de un ArrayList dado,

6.2.1.3 Eliminación de jugadores y paso de turno

Básicamente todo va alrededor de si el rey de un jugador desaparece (tanto por "jaque mate" como por que el tablero desaparezca bajos sus pies) es game over para el jugador. Eso es un simple bucle que es el método ColorSeVaDePartida:

```
for (Casilla x:Tablero) if( x.Ocupada!=null &&
x.Ocupada.ClasePieza==TipoPieza.KING && x.Ocupada.ColorJugador==color)
```

Es decir si en cualquiera de las casillas del tablero hay una casilla que está ocupada (no es nula) por un rey (TipoPieza.KING) del color que se está comprobando el jugador sigue en partida, si no hay que eliminarlo. Usaremos esa función como he dicho para los jaque mates y las derrotas por caer en el abismo. Para rendirse ya quitaremos la comprobación en su momento.

Si esa función devuelve que hay que quitarlo se eliminara de la colección de colores/jugadores y se eliminaran todas sus piezas aparte de hacer un recorrido para encontrar el siguiente jugador que le toque jugar.

6.2.1.4 Movimiento

Básicamente consistirá en desplazar la pieza de una casilla a otra (una vez comprobado con MovimientoLegal si esto es posible dentro de las reglas del ajedrez gracias a la función que nos da los movimientos posibles de cada pieza en la casilla inicial).

Parece sencillo pero hay que tener cuidado porque básicamente se modifica el tablero y para usarlo en la IA queremos que se pueda no modificar mientras calcula tableros futuros por eso se usa el constructor copia y se devuelve un nuevo tablero.

También hay que hacer un chequeo del jaque mate pero como en este caso es solo de la última jugada es más eficiente ver si la casilla nuevamente ocupada y no recorrer todo el ArrayList.

Cuando comencemos a implementar lo de reproducir partidas y persistencia de estos datos (BD o ficheros) es aquí principalmente donde se añadirá que esta jugada se vaya incluyendo para la "plantilla" de la partida.

Un aspecto importante de la implementación es que como se usa una clase abstracta Pieza esta no se puede instanciar para el típico truco de:

```
Pieza pieza=new (casillaInicial.Pieza);
CasillaDestino.Ocupada=pieza;
casillaInicial.Ocupada=null;
```

Así que se ha tenido que implementar un método CopiaPiezaPorTipo() dentro de la clase Casilla que básicamente copia la pieza que se encuentra en una casilla y así ya instanciada por polimorfismo poderle pasar a la propiedad Ocupada. Este método básicamente consiste en un switch case para que llame al constructor correspondiente según el TipoPieza requerido.

6.2.1.5 PossibleMoves (y PossibleCaptures)

Toda pieza queda definida por su color y su tipo… Ya que con ello podemos calcular todos sus movimientos en un tablero dado (con todas sus casillas y piezas) una casilla de salida determinada. De hecho como ya he dicho tuve dudas de si implementar las piezas como implementaciones de una interfaz con movimientos y capturas o pieza abstracta.

Es en el método PossibleMoves donde se implementa cada uno de los movimientos posibles. No voy a hablar de cada uno de ellos en particular. Además, en principio uno creería que podría hasta habérmelos copiado de cualquier otra implementación de ajedrez pero… ¿recordáis que el tablero cambia? Así que me voy a centrar en las generalidades que son necesarias cambiar por motivos del tablero menguante.

6.2.1.5.1 Piezas Mayores (Menos el Caballo y el Rey)

Al poder recorrer todo el tablero será necesario saber en que momento parar los bucles de los movimientos porque se salgan del tablero. Esto se puede conseguir de diversas maneras pero la que me ha parecido mejor es usar:

- Las casillas esquinas para tener límites de columnas y filas para tener los límites en los que se pueden hacer los bucles necesarios. (Es fácil ver que con esto tenemos las coordenadas máxima de todo el tablero).

```
        Casilla esquinaInferiorIzquierda=tablero.Tablero.get(0);
        Casilla
esquinaSuperiorDerecha=tablero.Tablero.get(tablero.Tablero.size()-1);
        //con esas puedo sacar la fila y columna máxima y mínima (En realidad
solo necesito las filas
        //o las columnas pero dejo comentado la otra parte por si acaso
        int filaMinima=esquinaInferiorIzquierda.Fila;
        int filaMaxima=esquinaSuperiorDerecha.Fila;
        int columnaMinima=esquinaInferiorIzquierda.Columna;
        int columnaMaxima=esquinaSuperiorDerecha.Columna;
```

- Un chequeo por si nos pasamos de los límites del tablero o sobre todo tropezamos con una pieza que si es de nuestro color no podremos saltar o si es de otro color solo podremos alcanzar esa casilla como máximo al comérnosla. Para lo que usamos la función que es igual para todas las piezas mayores llamada CasillaNoDisponible. Esta la hemos colocado en la clase abstracta Pieza para repetir un poco menos código aunque

lógicamente no debería estar ahí por las otras piezas, pero como las otras piezas no lo usan pues…

```
    boolean CasillaNoDisponible(Casilla auxiliarCasilla, ArrayList<Casilla>
listaCasillas) {
        if (auxiliarCasilla==null) return true;
        else if (auxiliarCasilla.Ocupada==null)
listaCasillas.add(auxiliarCasilla);
        else if (auxiliarCasilla.Ocupada.ColorJugador!=ColorJugador) {
            listaCasillas.add(auxiliarCasilla);
            return true;
        } else {
            return true;
        }
        return false;
    }
```

Como se puede ver esta función se encarga tanto de devolver true cuando la casilla no es accesible (o es la última que puede ser accesible porque contiene una pieza enemiga) como ir añadiendo la susodicha casilla al ArrayList de casillas posibles que luego formara el ArrayList de casillas que se devuelva en la función PossibleMoves correspondiente pasándose esta por referencia.

En los casos de las Piezas mayores sin caballo y rey además el método PossibleCaptures será igual a PossibleCaptures así que se limita a devolver una llamada al otro método. Y para simplificar más la cosa como los movimientos de la dama es la suma de los movimientos de una torre y un alfil el método PossibleMoves de la dama en realidad se saca instanciando una torre y un alfil y sacando su lista de casillas.

6.2.1.5.2 Caballo

Curiosamente no presenta ninguna complejidad en comparación. Un pequeño bucle con un "check" de que no se vaya fuera del tablero y ya está. PossibleCaptures igual a PossibleMoves. Pongo aquí el bucle por ser un poco más enrevesado que recorrer una fila, diagonal o columna pero no tiene mayor historia.

```
        for (int i=-2;i<=2;i++){
            if (i!=0){
                aux=3-Math.abs(i);
                auxiliarCasilla=tablero.getCasilla(fila+i,columna+aux);
                if (auxiliarCasilla!=null && (auxiliarCasilla.Ocupada==null
||(auxiliarCasilla.Ocupada!=null &&
auxiliarCasilla.Ocupada.ColorJugador!=ColorJugador)))
listaCasillas.add(auxiliarCasilla);
                auxiliarCasilla=tablero.getCasilla(fila+i,columna-aux);
                if (auxiliarCasilla!=null && (auxiliarCasilla.Ocupada==null
||(auxiliarCasilla.Ocupada!=null &&
auxiliarCasilla.Ocupada.ColorJugador!=ColorJugador)))
listaCasillas.add(auxiliarCasilla);
            }
        }
```

6.2.1.5.3 El peón

La pecularidad del peón es que es una de las piezas en que el método PossibleMoves es distinto al de PossibleCaptures. El peón sólo captura en diagonal asi que el método de PossibleCaptures ha de mirar en las 4 casillas

que puedan haber cosas comestibles. El siguiente bucle anidado hace el trabajo:

```
        for(int i=-1;i<=1;i+=2) for(int j=-1;j<=1;j+=2)
```

En cambio en PossiblesMoves hay que sumar los movimientos en línea recta en general teniendo en cuenta que si hay una pieza en ellos no se la come (da igual su color) y las posible capturas. Es decir,

```
        for(int i=-1;i<=1;i++) for(int j=-1;j<=1;j++){
            auxiliarCasilla=tablero.getCasilla(fila+i, columna+j);
            //Movimientos sin captura (comprobar que no está ocupada la
casilla (no salta)
            if (i==0||j==0 && i!=j){
                if (auxiliarCasilla!=null && auxiliarCasilla.Ocupada==null)
listaCasillas.add(auxiliarCasilla);
            }
            //Diagonales si hay piezas para capturar
            else  if (auxiliarCasilla!=null && auxiliarCasilla.Ocupada!=null
&& auxiliarCasilla.Ocupada.ColorJugador!=ColorJugador)
listaCasillas.add(auxiliarCasilla);
```

6.2.1.5.4 El Rey

Puede parecer un caso más sencillo que el peón… Todas las casillas a distancia de uno y simplemente comprobar que la ficha no es propia para no tener en cuenta las capturas pero… El rey es especial…

A los posibles movimientos hay que quitarle las casillas que estén amenazadas por otro jugador. Algo tan aparentemente "sencillo" como llamar a la función que he llamado CasillaAmenazadaPorOtroJugador:

```
boolean CasillaAmenazadaPorOtroJugador(Casilla casilla, Color color){

    for (Casilla x:Tablero){
      if (x.Ocupada!=null && x.Ocupada.ColorJugador!=color){
        if(x.Ocupada.PossibleCaptures(x, this).contains(casilla)) return true;
      }
    }
    return false;
 }
```

Notesé que no es necesario quitar la casilla x de la lista porque al ser ocupada con una pieza del mismo color…

Pero en PossibleCaptures de la pieza del Rey aunque pueda parecer que el mismo check es necesario no hay que incluirlo. No solamente porque "técnicamente" luego en partidas Blitz se pudiera comer un rey si allí hubiera algo que comer sino porque originaria un problema de recursividad no deseada al buscar calcular todos los movimientos posibles. Para calcular el movimiento de un rey habría que chequear si el movimiento de los otros reyes se lo pueden comer para lo cual hay que calcular todas las casillas que amenaza esos reyes y esto a su vez…

Por eso quitando esa restricción del PossibleCaptures del rey ya no se produce y da los resultados esperados. ¿El inconveniente? Que hay que repetir un poco de código para esas dos funciones del rey.

6.2.1.6 TablerosPosibles

Toda la implementación de los algoritmos IA empleados se basa en la capacidad de calcular todos los movimientos posibles a partir de una posición dada. Así que es necesario poder calcular estos. De eso se encarga la función TablerosPosibles dentro de la clase Board.

Esta función básicamente recorre todas las casillas del tablero para encontrar las ocupadas y que sean del color del turno del jugador y van llenando un ArrayList de Boards con todos los movimientos posibles de esa pieza. Creo que en código se ve más fácilmente.

```
    ArrayList<Board> TablerosPosibles(Boolean noAnalisis){
        ArrayList<Board> tablerosFuturos=new ArrayList();
        for (Casilla x: Tablero){
            if(x.Ocupada!=null && x.Ocupada.ColorJugador==TurnoJugador){
                for (Casilla y:x.Ocupada.PossibleMoves(x, this))
tablerosFuturos.add(Movimiento(x, y, noAnalisis));
            }
        }
        return tablerosFuturos;
    }
```

Evidentemente en futuras implementaciones IA habrá que "ordenar" estos tableros para que la poda alpha-beta vaya mejor pero de momento esta es la clave de poder comenzar a realizar los árboles del juego.

6.2.1.7 Distancia al final del tablero

Aunque en el ajedrez tradicional la importancia de controlar el centro es importantísima en el ChessBattleRoyale esto se acrecienta al disminuir el tablero. Eso se va a imponer en la IA mediante factores que sopesen la distancia a este centro. Pero si bien en el ajedrez normal calcular esa distancia no tiene mayor dificultad (al menos para el ojo humano) y se puede estimar hasta como una constante dependiendo de la casilla en un tablero que va menguando… Por eso he implementado una función que se encarga de este menester DistanciaFinalTablero.

Básicamente consta de un switch case que incrementa esta distancia si el tablero aún no presenta muestra de peligro (no está rojo o de otro color) y el siguiente código:

```
        for(int i=0;i<=Rows/2;i++){
            if((fila==i+(InitialRows-Rows)/2||fila==(InitialRows+Rows-2*i-
2)/2)||(columna==i+(InitialColumns-
Columns)/2||columna==(InitialColumns+Columns-2*i-2)/2)){
                contador=i;
                break;
            }
        }
```

Que en lenguaje humano lo que hace es calcular el mínimo valor de lejanía de la fila o la columna con respecto al máximo valor de las columnas y filas máximas actuales. La geometría que fundamenta el bucle es similar a la que empleaba cuando calculábamos las casillas que estaban en el perímetro de peligro para el IncrementaAlertaTablero pero a la inversa.

No me gusta porque no parece demasiado intuitivo y me costó estar pintando casillas y tablero… Pero es otro de los problemas por no tener una geometría fija del tablero.

6.2.1.8 SillyIA

Ya hemos mencionado que vamos a usar la búsqueda minimax con poda alpha-beta como algoritmo para la implementación de la IA y hemos presentado el algoritmo en pseudocódigo. En java y con la implementación del ChessBattleRoyale que estamos haciendo el algoritmo es casi completamente directa así que no la reproduzco aquí. Solo queda decir que por eso implementamos las funciones de TablerosPosibles.

Sin embargo no hemos hablado de cuál iba a ser nuestra función de evaluación estática de cada posición así que vamos a ver cómo hemos codificado los factores que queremos que tenga en cuenta nuestra IA.

En principio la fórmula que vamos a emplear es:

$$Evaluación = \frac{\sum_{piezas} V_{jugador} - \sum_{jugadores \neq jugador} \sum_{piezas} V^{i}_{jugador}}{numero\ jugadores}$$

Donde cada término V es igual a:

$$V_{jugador} = F_{valor} \cdot F_{distancia} \cdot (1 + F_{movimientos})$$

Estas expresiones se pueden complicar más o menos y sólo estoy tratando de la implementación actual que he realizado yo con unos parámetros fijos que en posibles expansiones podremos ajustar. Cada una de estos valores F presentan la siguiente forma dependiendo de la pieza sobre la que se estén evaluando.

6.2.1.8.1 El F_{valor}

El F_{valor} no es más que intentar pesar el tipo de pieza que es. Ya sabéis lo que suelen enseñar de un peón vale uno, un alfil 3…). Hay que tener en cuenta en la implementación dos cuestiones:

- Aunque se han puesto valores semejantes a los ajedrecísticos con los cambios de reglas… Seguiremos trabajando en ello.
- El valor del rey aquí es como pieza de ataque. Todos sabemos que como si se pierde se pierde la partida esto se tratara como extra para la hora de evaluar.

6.2.1.8.2 El $F_{movimiento}$

He considerado que como primera aproximación poner un factor que sopese si una pieza tiene más liberta de moverse sería una buena idea (más movimiento más útil la pieza). Este factor he pensado que irá de 1 (máximo movimientos posibles versus los teóricos en el centro del tablero) a 0 (pieza inmovilizada). Aún así una pieza sin movimientos tiene cierto potencial latente

por eso hay que introducir un BIAS y ajustar. En primer término he cogido la función $1+F_{movimiento}$ porque asegura un cierto compromiso en ello.

6.2.1.8.3 El $F_{distancia}$

Los anteriores factores lo único especial que tenía la codificación es que he preferido definir una clase "pojo" con los parámetros de cada una de las piezas y luego comparar si la pieza que se está evaluando el tipo aprovechándome de que he definido el operador equals para solo comparar el tipo de pieza y que me devuelva los valores de los parámetros. En cambio en el factor distancia es donde hay que contabilizar que el tablero está menguando y cuanto más lejanos estemos al peligro pues la pieza tendrá un mayor valor.

Existen muchas posibilidades para tomar esto en cuenta (de hecho cuando el tablero es fijo suelen tomar parte de la geometría del tablero construyendo matrices de valores de posición según pieza). Una de las cosas que tiene que cumplir es que sea una función que valga alrededor de uno en las zonas sin peligro (el centro) y que disminuya conforme se acerca al borde del tablero (y más en los casos que este esté en zona de peligro inminente) y que vaya cambiando conforme el tablero va menguando.

Mi candidata tras muchas intentonas es la siguiente función:

$$F_{distancia} = \sqrt[2]{\frac{distancia}{\theta_{filas}}} + \frac{\left(1 - \frac{distancia}{\theta_{filas}}\right)}{(distancia_{maxima} + 1) - \theta_{rows}}$$

Las imágenes de cómo se comporta esta función con la distancia y conforme el tablero va disminuyendo el tablero se pueden ver en el apéndice D.

6.2.2 La implementación visual con LibGdx

Siento que esta parte ha de ser mejorada en futuras versiones porque la implementación que he hecho ha emburullado bastante la lógica del ChessBattleRoyale que tenía implementada para que funcionara.

Como ya he dicho, no quiero centrarme en demasía en describir los fundamentos de la herramienta sino las características distintas. No obstante, debo decir que la idea era usar la funcionalidad de Scene2D para ir estableciendo en cada ventana el stage correspondiente a la que podían unirse las clases anteriormente creadas y descritas como actores. Así sólo habría que sobrescribir el método draw de cada uno de ellos para ir dibujando.

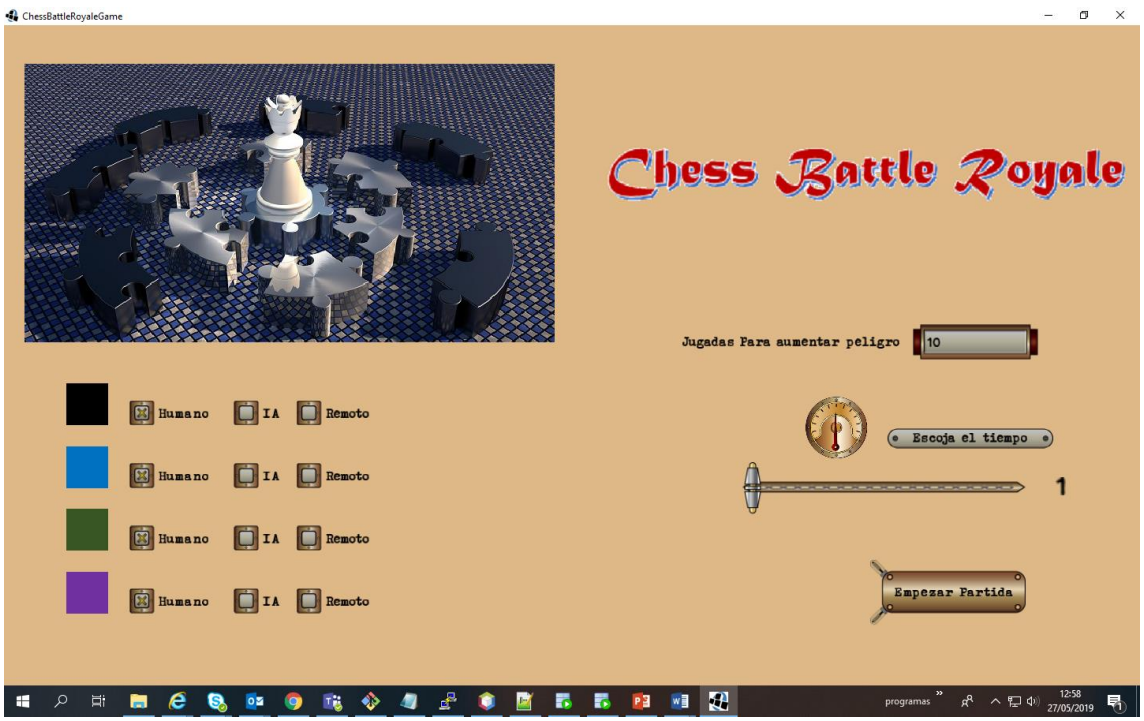
Así en principio podría parecer suficiente con hacer actores a cada una de las casillas y al tablero (para pintar otra información de la partida que se guarda ahí y no en cada casilla). Así lo he hecho pero… al hacerlo me he visto obligado a usar otro método para movimiento (porque quitar actores al stage no presenta problemas pero unirlos sí y hacía que no se reflejaran los estados

del nuevo Tablero copia). Eso no sé que habrá hecho con mi implementación de la IA pero veremos.

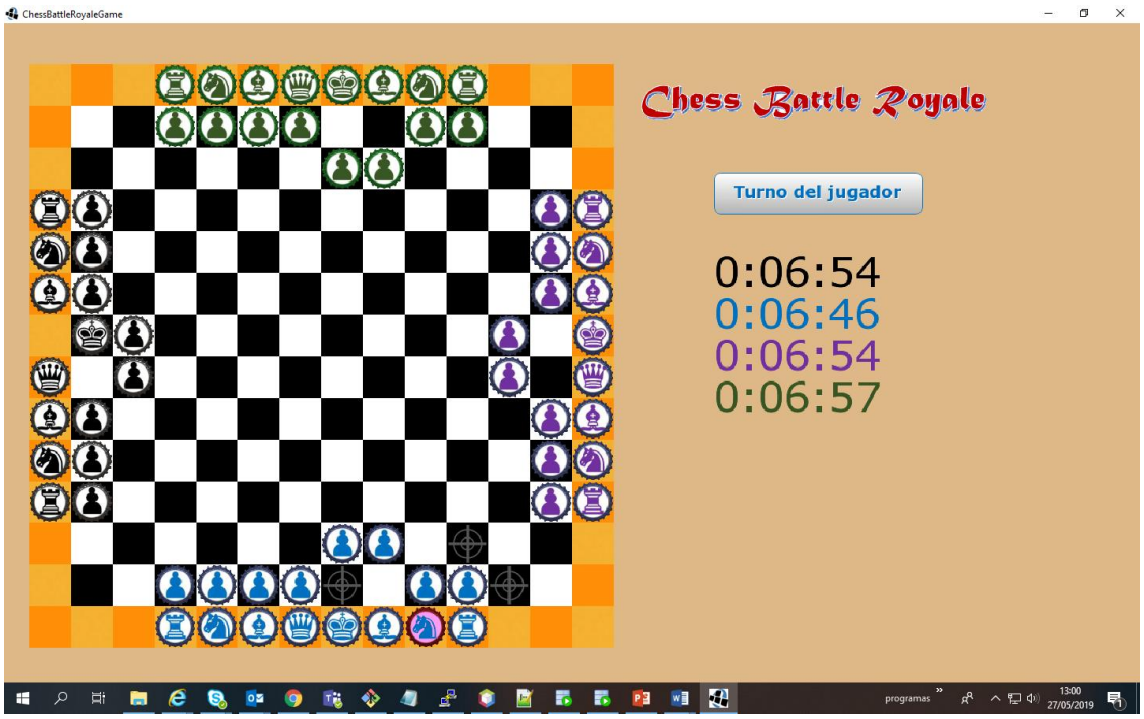
He usado el Texture Packager junto con el Asset Manager para colocar las propias texturas necesarias para el tablero y la pantalla en general del juego. Pero para la ventana de configuración quise usar Scene2Dui junto con skin para poder cambiar el aspecto gráfico sencillamente. Los dos sistemas de tratar con las textures no se llevan demasiado bien.

7 Manual de usuario

Bueno, este es el manual de la versión v0.2alpha1. Básicamente es ejecutar el programa. Configurar el slider para los minutos por jugador. Introducir el número de movimientos a partir del cual se aumenta el peligro (posiblemente desaparecerá esta opción en futuras versiones pero ahora se pone por facilitar las pruebas). Y darle a comenzar partida.



Una vez en el tablero simplemente si esta en tu turno (reflejado por el color del cuadrado correspondiente y porque es el reloj que se está moviendo) hay que clickar la ficha que se quiere mover (se señalará para que se sepa que es esa ficha y solo puedes mover tus propias fichas de tu color) y mover a una casilla que pueda moverse (se señala con una diana gris). En ese momento se moverá la pieza y se parará el reloj pasando a correr el turno del jugador siguiente. Si se quiere mover otra pieza se deselecciona la ficha clickando de nuevo y se puede volver a escoger pieza.



Cuando todos los jugadores hayan movido se considera 1 ciclo de movimiento y si han pasado los ciclos configurados se incrementa el status de peligrosidad correspondiente. Si este sobrepasa el rojo desaparece esa parte del tablero con todas las piezas correspondientemente. Si una de esas piezas es el rey desaparecen todas las piezas de ese jugador que habrá perdido la partida.

Simple, ¿no? Para más información y demás visitar la página web del proyecto y repositorio https://luispivo.github.io/ChessBattleRoyale/ donde ahora mismo puedes encontrar un mini-tutorial de como se mueven las piezas y en un futuro habrá tutoriales y documentos de estrategias más elaborados.

8 Requisitos e instalación

8.1 Instalación y Requisitos

El programa al final corre en la JVM con lo cual tiene como base los requerimientos que tenga la JVM hasta la versión que se haya configurado el java. (Y por supuesto en principio que Java esté instalado en el sistema). De la página de Oracle:

- **Windows**
- Windows 10 (8u51 y superiores)
- Windows 8.x (escritorio)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64 bits)
- Windows Server 2012 y 2012 R2 (64 bits)

- RAM: 128 MB
- Espacio en disco: 124 MB para JRE; 2 MB para Java Update
- Procesador: Mínimo Pentium 2 a 266 MHz
- Exploradores: Internet Explorer 9 y superior, Firefox
- **Mac OS X**

- Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+
- Privilegios de administrador para la instalación
- Explorador de 64 bits

Se requiere un explorador de 64 bits (Safari, por ejemplo) para ejecutar Oracle Java en Mac.

- **Linux**

- Oracle Linux 5.5+[1]
- Oracle Linux 6.x (32 bits), 6.x (64 bits)[2]
- Oracle Linux 7.x (64 bits)[2] (8u20 y superiores)
- Red Hat Enterprise Linux 5.5+[1], 6.x (32 bits), 6.x (64 bits)[2]
- Red Hat Enterprise Linux 7.x (64 bits)[2] (8u20 y superiores)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64 bits)[2] (8u31 y superiores)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 y superiores)
- Ubuntu Linux 15.04 (8u45 y superiores)
- Ubuntu Linux 15.10 (8u65 y superiores)
- Exploradores: Firefox

No hacemos mucho uso de texturas ni de cosas raras así que debería funcionar con esas especificaciones siempre que el ordenador tuviera instalada la correspondiente versión de java. También habría que ver la gráfica pero funcionará siembre que funcione OpenGL (2.0 por seguridad) en ella.

La IA sí que hace un uso extensivo del CPU y de crear y destruir objetos así que es probable que si se juega contra la IA estos requisitos mínimos hayan de ser expandidos para un correcto funcionamiento. Pero faltarían pruebas más extensivas.

Una vez generado el fichero jar correspondiente mediante la task del gradle desktop:dist simplemente queda colocar este jar junto con los ficheros de los assets necesarios (los atlas correspondientes, los ficheros de fuentes y el json de la skin) y ejecutar java –jar el fichero. Pero para simplificar aún más la instalación del programa he bundleado este fichero (usando el programa Launch4j) para que el proceso de instalación simplemente en descomprimir el fichero ChessBattleRoyale.zip copiar una carpeta y hacerle doble click a ChessBattleRoyale.exe.

Aunque no he comprobado que esto sirva igual para Linux existen formas similares.

8.2 Descripción del entregable

El entregable básicamente es una foto estática de como está el proyecto en github a fecha de la entrega de 29-05-2019. Este es un proyecto que sigue elaborándose y no puede ser otra cosa.

Como hacer un clone del repositorio en un cd sin explicación de la estructura queda así de raro

9 Conclusiones

9.1 Conclusiones sobre el trabajo realizado

Una y no más, Santo Tomás

9.2 Posibles ampliaciones y mejoras

Durante la elaboración de esta memoria se han presentado muchas notas sobre expansiones y posibilidades en un futuro. Muchas no se han llevado a cabo dada las limitaciones del tiempo y mis posibilidades…

Creo que es necesario este apartado para puntualizar en el plan de trabajo que cosas son prioritarias, que quedan como expansión y que sería cuestión de un Chess Battle Royale 2.0. En principio lo principal es lo que se señala en los apartados 1 y 2 de los Objetivos. Pero creo que no es malo que lo repita y acote si cabe un poco más los primeros pasos:

Lo que definiremos como "juego base": Tablero y piezas con gráficos 2D. Sus movimientos legales.

El camino a recorrer a partir de ahora es ahondar en las posibilidades que abre la escritura de las partidas para poder realizar una especie de "Visor" de partidas que pueda usarse para hacer repeticiones de partidas y pruebas sobre reglas y partidas en el mismo dispositivo.

También con la escritura se podrá realizar una recolección de esas partidas (movimientos, jugadores y resultados) en una base de datos. (Ver Apéndice A para algunas cosas pensadas en cómo anotar estos movimientos). Las partidas guardadas deben poder reproducirse en el juego base y viceversa. Y con los datos de los jugadores y sus resultados se elaborará un Ranking de jugadores (Clasificación). (Ver Apéndice B para algunas consideraciones preliminares sobre esta clasificación)

Todo lo demás. (Aunque sí, hay un orden pero sin tener los puntos I y II es como el cuento de la lechera).

El usar NodeJs para que el juego se pueda jugar en multiplayer remoto es otro de los aspectos más importantes que quedan por realizar. Y que es necesario para el aspecto económico de la aplicación.

También quiero en un futuro hacer una pantalla para configurar diagramas y así configurar la pantalla inicial que sería un paso para usar esta.

Al final con tantas expansiones posibles y más o menos factibles queda un enorme trabajo por el que como he dicho necesitaría de un equipo más amplio.

10 Referencias

[1] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Chaturaji. [Último acceso: 31 3 2019].

[2] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Variante_del_ajedrez. [Último acceso: 31 3 2019].

[3] D. Pritchard, La Enciclopedia de las Variantes del Ajedrez., Games & Puzzles Publications., 1994.

[4] «Lichess,» [En línea]. Available: https://lichess.org.

[5] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/El_Turco. [Último acceso: 31 3 2019].

[6] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Deep_Blue_(computadora). [Último acceso: 31 3 2019].

[7] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Fritz_(ajedrez). [Último acceso: 31 3 2019].

[8] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/ChessBase. [Último acceso: 31 3 2019].

[9] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/AlphaZero. [Último acceso: 31 3 2019].

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan y D. Hassabis, «A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,» *Science,* vol. 362, nº 6419, pp. 1140-1144, 2018.

[11] S. Hudson, «Scid,» [En línea]. Available: http://scid.sourceforge.net.

[12] «PlayChess,» [En línea]. Available: http://en.playchess.com.

[13] «Internet Chess Club,» [En línea]. Available: https://www.chessclub.com.

[14] «Free International Chess Server,» [En línea]. Available: https://www.freechess.org.

[15] «Chess.com,» [En línea]. Available: chess.com.

[16] «Chess24,» [En línea]. Available: www.chess24.com.

[17] P. d. R. Laborales, «Ley 31/1995 del 8 de Noviembre».

[18] «Real decreto 39/1997 del 17 de enero sobre Reglamento de Servicios de Prevención,» *BOE,* nº 23 de abril, 1997.

[19] «Real Decreto 488/1997 del 14 de abril sobre disposiciones mínimas de seguridad y salud relativas al trabajo con equipos que incluyen pantallas de visualización,» *BOE.*

[20] «Real Decreto 486/1997 del 14 de abril sobre disposiciones mínimas de seguridad y salud relativas a los lugares de Trabajo,» *BOE.*

[21] Fraternidad-Muprespa, «Manual de Prevención de riesgos personal administrativo».

[22] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Minimax.

[23] «github,» [En línea]. Available: http://www.github.com.

[24] «git,» [En línea]. Available: https://git-scm.com/.

11 Apéndice A: Escritura

En principio la idea era usar una especie de escritura Algebraica a-lo ajedrez pero modificada por las necesidades de ser 4 jugadores y con el tablero cambiante. En principio el trabajo ya está hecho porque lo único que hay que hacer es ir recolectando las filas y columnas de las casillas de cada uno de los jugadores.

Una forma sencilla y que surge espontáneamente cuando sabes que es una partida de ajedrez es un fichero de tipo json que recolecte primero una cabecera con información de los jugadores (y como extra lo mismo información de la partida) y luego parejitas:

```
{
	"Partida":[
		{"Movimiento": 1,[
			"Black":{
				"Inicio":[13,13],
				"Final":[12,13] },
			"Blue":{
				"Inicio":[5,5],
				"Final":[3,2] },
			"Green":{
				"Inicio":[0,5],
				"Final":[1,3] }
			]
		}
	]
}
```

Las clases empleadas en el juego ya generan esa información con lo cual pasarla a escribir en un fichero tanto en forma json como simplemente casilla inicial, casilla final es casi automática. Y serían fácil luego de manejar mediante una base de datos que usara ficheros json como mongoDB (añadiendo un _id a cada fichero).

De hecho aunque para la visión del juego me he centrado en el juego en las pruebas anteriores siempre pintaba los tableros en forma de string e iba transformando la notación de la partida para ir moviéndolas. Así que en realidad la base de lectura está ya realizada y comprobada pero no ha sido puesta por querer también incluir el juego.

12 Apéndice B: Ranking (Clasificación)

En el ajedrez se suele usar, (por diversas federaciones la más importante la FIDE), un sistema de ranking denominado ELO en honor al físico americano de origen húngaro que desarrolló el método matemático, basado en cálculo estadístico, para calcular la habilidad relativa de los jugadores. Aunque en algún portal de chess online se usa el conocido como sistema de puntuación Glicko (que se encuentran en dominio público y que se supone que mejora a los ELOS por tener en cuenta el tiempo entre mediciones de fuerza). Son estos sistemas los que tengo en mente para establecer el ranking de jugadores y con ello evaluar la fuerza de los jugadores.

Aunque la base matemática y sus fórmulas difieran ambos sistemas se basan en una estimación de cuántas partidas deberían haber ganado o perdido un jugador frente a una oposición dada. Por ello nuestro programa debe recopilar los resultados de las partidas y los oponentes (y su fuerza). Y aplicando el algoritmo correspondiente calcular la puntuación estimada del jugador o su variación por jugar x partidas con sus respectivos resultados frente a unos rivales.

Pero el ajedrez es de dos y el juego de la propuesta es de 4 jugadores. Creo que un primer acercamiento consistirá en tomar el orden de eliminación y dar la siguiente puntuación de partida:

0 Puntos: Ser eliminado el primero de la partida.

¼ Puntos: Ser eliminado el segundo.

¾ Puntos: Ser eliminado el tercero.

1 Punto: Ser el ganador.

Con esto y algún retoque estableceremos una puntuación estimada que nos ayudará tanto para toques competitivos como para el matchmaking futuro cuando finalmente saquemos la versión multijugador. Los campos en la clase jugador están preparados para eso.

13 Apéndice C: Sobre IAs

13.1 Definición

Hay veces que las propias definiciones son un tanto oscuras (o poco claras) y muchas veces encierran más una forma de enmascarar con una simbología mística o mágica cosas que no son más que pura implementación técnica o científica. La inteligencia en sí es una de esas cosas cuya definición no ha estado completamente clara o fuera de "disputas"; pues imaginaros la inteligencia artificial.

Andreas Kaplan y Michael Haenlein definen la inteligencia artificial como "la capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas a través de la adaptación flexible". Quizás sea una definición tan buena como otra pero cuando la analizamos con ojos un poco más fríos es fácil darse cuenta que aunque en general puede ser acertada la implementación en sí no deja de ser ‘‘poco’’ inteligente.

Tradicionalmente la inteligencia artificial (que según la wiki ahora se conoce también como IA simbólico-deductiva estaba basada en el análisis formal y estadístico del comportamiento humano ante diferentes problemas. De esta forma se usaban razonamiento basados en casos bajo un árbol de decisiones, usando las reglas matemáticas necesarias o incluso redes bayesianas para inferir probabilísticamente las mejores decisiones a tomar.

De esta manera se intentaba establecer un modelo que aplicase a la situación que se quisiese resolver, (muchas veces mediante un conocimiento más que experto del problema en sí), y a partir de ese modelo calcular cual era la mejor respuesta (muchas veces a base de fuerza bruta del ordenador usado).

El ajedrez ha sido un juego que clásicamente ha sido un campo de batalla para estas aproximaciones a la inteligencia con bastantes buenos resultados, básicamente basándose en evaluaciones de posiciones y un algoritmo minMax (y variantes mejoradas como la poda Alpha-Beta) que simplemente van cogiendo la variante que mejor puntuación tiene para el jugador teniendo en cuenta que el otro jugador escogerá la que dé peor puntuación para el contrario.

Hoy en día se han puesto en relevancia la IA subsimbólica-inductiva (o dicho de otra manera parecen más brillantes). El aprendizaje se realiza basándose en datos empíricos y en un proceso de ‘‘auto’’-aprendizaje, en los que no se conoce el modelo o el algoritmo pero si una cantidad ingentes de inputs y outputs a través de los cuales con un modelo matemático de variable complejidad (al que llaman redes neuronales) pues pueden ajustarlo y sacar predicciones.

Contado como lo cuentan pues parece hasta mágico (y no estoy tratando de minimizar sus resultados) pero en realidad tanto la IA ‘‘clásica’’ como esta moderna que ‘‘aprende’’ no es tan ‘‘lista’’ (y con esto quiero decir que no están mágica).

En el siguiente mini-capítulo voy a intentar resumir un ejemplo inicial de un curso de Google sobre Deep Learning que creo que explica perfectamente el enfoque de esta técnica computacional. Y sí, creo que el Chess Battle Royale necesitará de este tipo de técnicas para futuras IAs más decentes pero… eso futuro.

13.2 MACHINE LEARNING

> *¿Cómo sabía yo que al jugar 18.d5, 21.a5, 25.h5 no se reducirían mis posibilidades de victoria? Porque he visto muchas partidas similares en los libros antiguos de ajedrez.*
>
> *Un año más tarde, durante el torneo de IBM de Amsterdam en 1968, contemplé divertido cuán rápido Lubomir Kavalek había llevado a cabo el mismo plan en su partida con Lengyel. Le pregunté más tarde a Kavalek la razón de su veloz juego, y me explicó que Vlastimil Hort le había enseñado mi partida con Winiwarter unos pocos meses antes. He aquí porque la generación más joven es siempre más inteligente.*
>
> ***D.Bronstein "Aprendiz de brujo"***

Recordar la disyuntiva del apartado anterior.

En muchos casos de programación lo que se pretende es implementar un algoritmo para que a través de unos datos de entrada (inputs) se obtengan unos resultados (outputs). El algoritmo, la ecuación, el problema puede ser más o menos complicado y requerir unas u otras técnicas pero… ¿Y si el problema no es que se conozca el algoritmo y se quieran calcular los resultados sino que se conocen inputs y outputs pero no se conoce el algoritmo?

El curso propone como encontrarías que una serie de datos de inputs y outputs tal que:

INPUT	**0**	**4**	**10**	**13**	**17**	**23**	**40**
OUTPUT	32	39.2	50	55.4	62.6	66.5	?

¿Qué valor tomaría el output a 40?

La solución es mucho más sencilla de lo que aparenta si se piensa durante más de un segundo. Tal vez te fijas que es una conversión o quizás decidas hacer una gráfica y como ves que parece una línea recta la ajustes simplemente con un coeficiente de regresión (o si fuera otro tipo de modelo con otro tipo de ajuste). En cualquier caso llegarías a la conclusión que OUTPUT = 1.8 * INPUT + 32 (conversión de grados Celsius a Farenheit).

Si este modelo de inputs y outputs fuera más complejo… no se podría usar una regresión linear o si fueran datos reales el coeficiente de regresión no sería perfectamente 1, y…

El modelo de redes neuronales lo que hace es establecer un mecanismo que a base de distintas capas con diferentes unidades de ajuste ("neuronas") pueden matemáticamente ir ajustando los parámetros para disminuir la distancia (o pérdida) de lo que predice ese modelo con los inputs y outputs

de los datos de verdad y llegar a tener el mejor algoritmo posible dado los datos, el paso para llegar a estas iteraciones del modelo, etc.

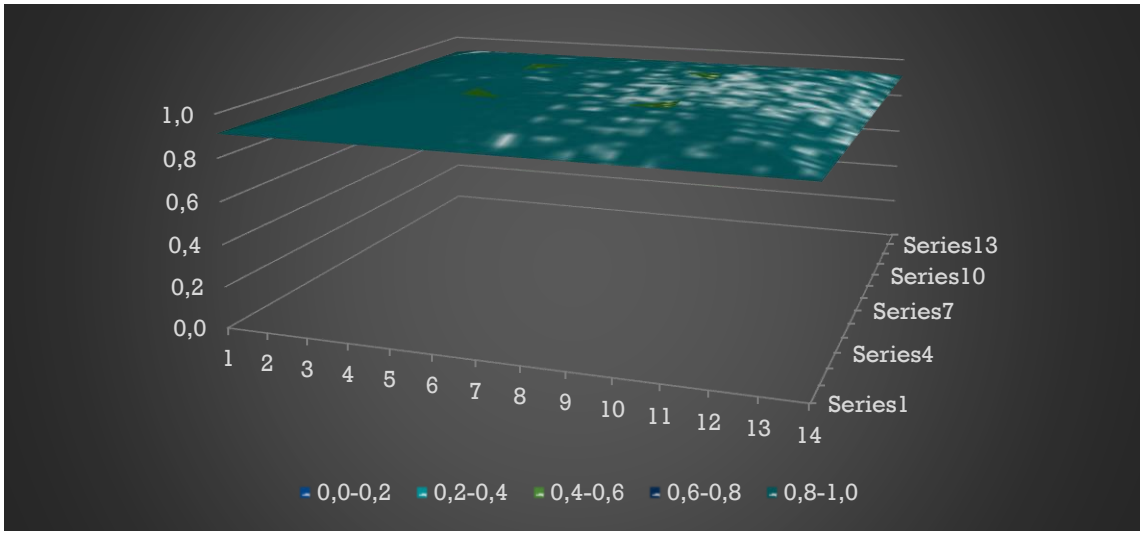
¿A que dicho así suena menos mágico que dicho de forma de entrenar al modelo con los datos para que aprenda cual…? De hecho eso son cosas que "tradicionalmente" siempre se ha hecho en el campo de la ciencia y de hecho las bases matemáticas son más que conocidas. Sigue siendo maravilloso las aplicaciones que están saliendo y demás; pero fundamentalmente (y si me permiten un chiste de mi época cuando buscaba funcionales de energía de correlación) si tienes tantos datos como variables a optimizar no necesito ni algoritmo puedo hacer una tabla.

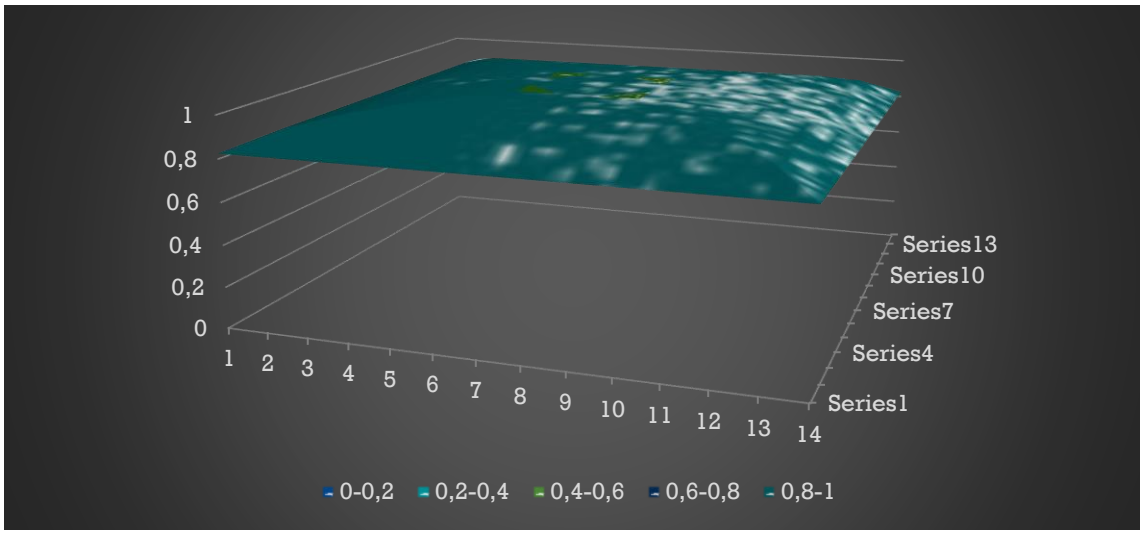
¿Cómo esto se puede aplicar al ajedrez? ¿O mejor dicho como lo han aplicado Google? Pues aparte de con mucho ingenio y muy bien, pues "técnicamente" haciendo que aprenda la máquina a base de muchas partidas y demás y luego ya el algoritmo lo ponen en máquinas más normales.

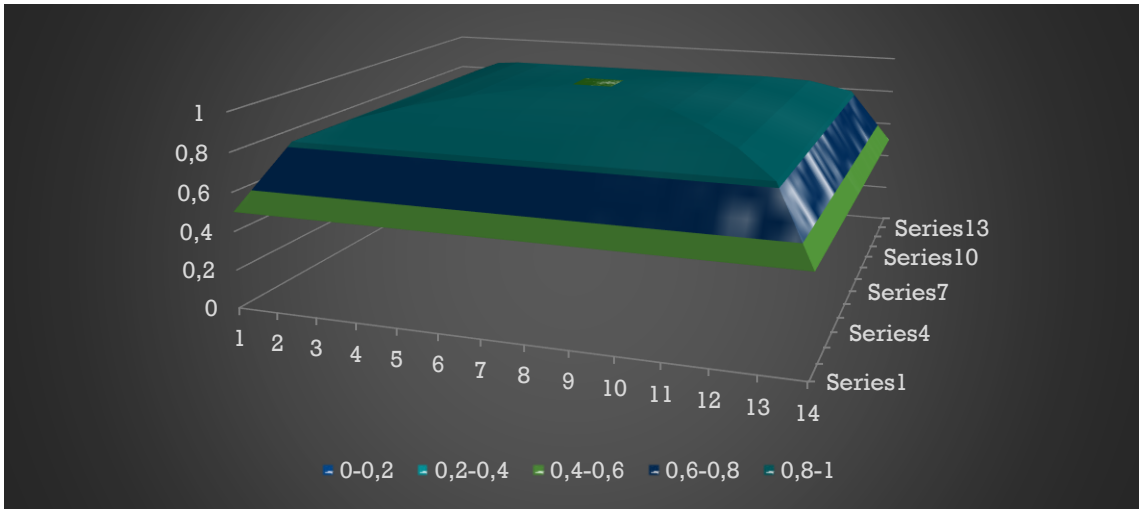
El caso es que aunque es plausible que el aprendizaje en el cerebro humano actúe en parte así en base a una memoria y un "fiteo" de variables por la complejidad neuronal, no me cabe duda que si estos ajustes se hacen con cabeza evitan una complejidad innecesaria.

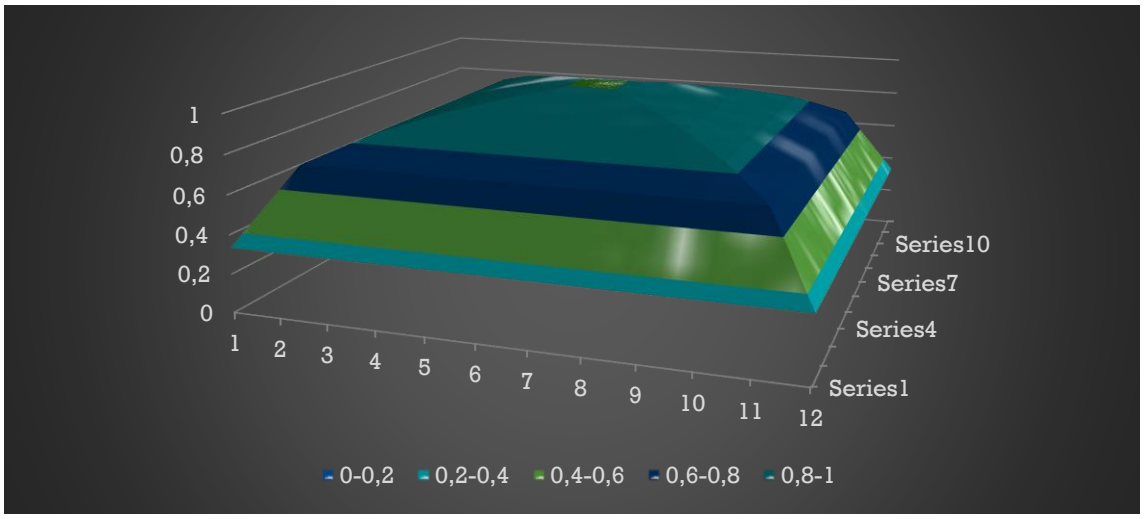
Por ello la primera aproximación a hacer una IA en el ChessBattleRoyale va a intentar ser una más clásica usando el conocido algoritmo MinMax junto con usar funciones que puedan ser más interpretables. No porque vayan a obtenerse unos mejores resultados que con el machine learning sino porque quiero ver si se puede sacar algo de "sabiduría· interpretando los valores que saquemos.

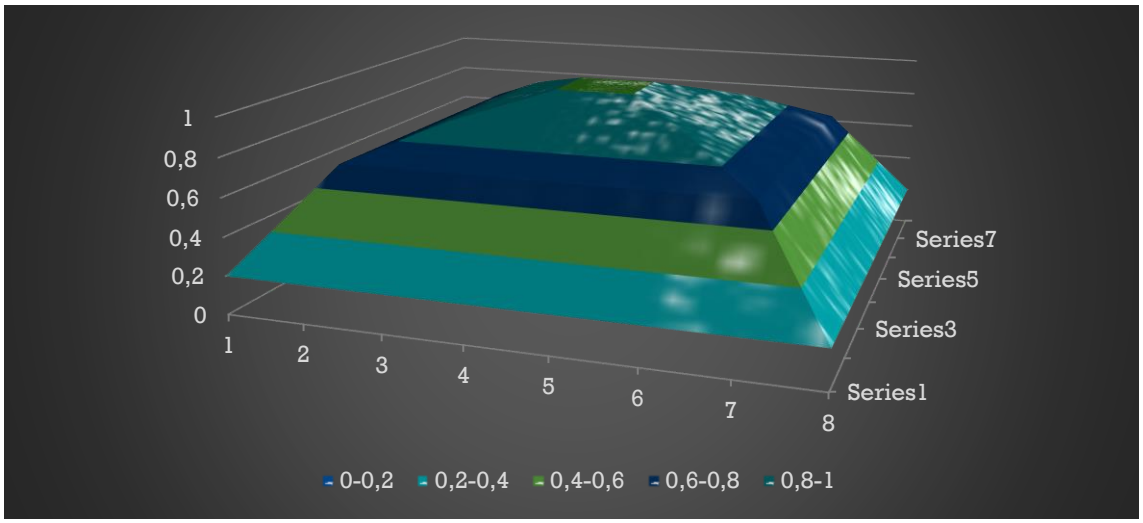
14 Apéndice D: Gráficas del Factor Distancia

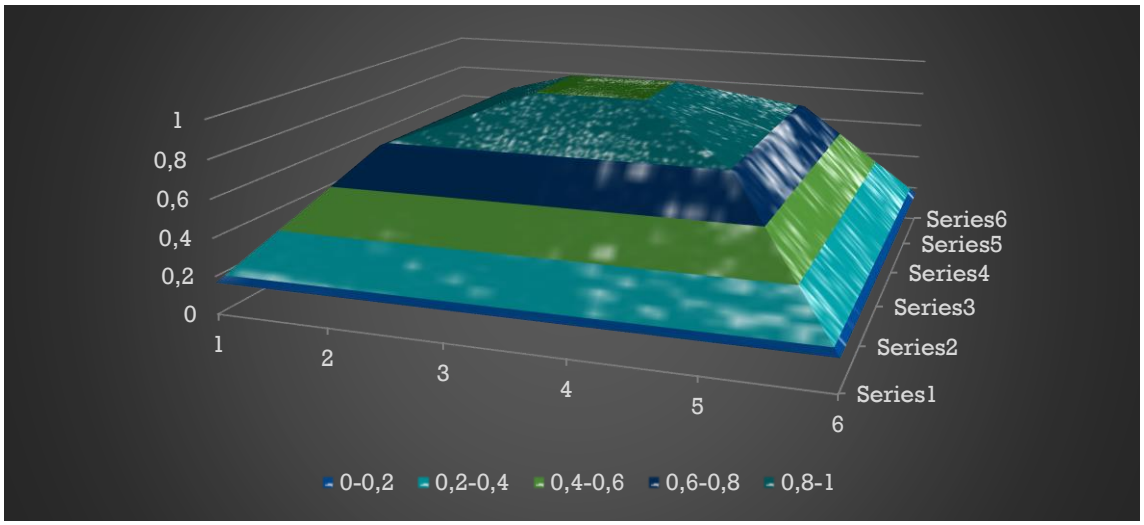


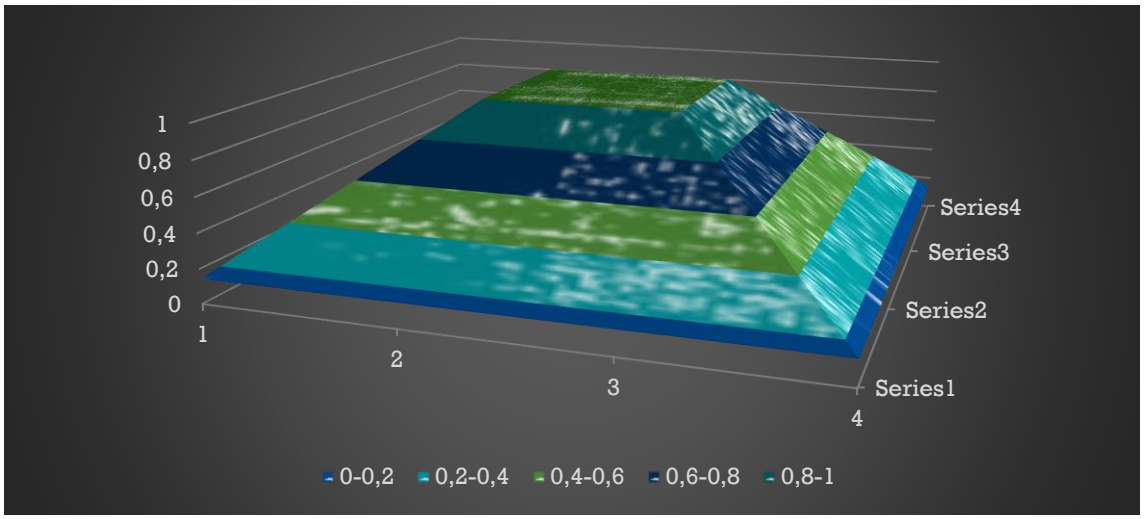

1
0,8
0,6
0,4
0,2
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Series13
Series10
Series7
Series4
Series1
0-0,2
0,2-0,4
0,4-0,6
0,6-0,8
0,8-1


1
0,8
0,6
0,4
0,2
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Series13
Series10
Series7
Series4
Series1
0-0,2
0,2-0,4
0,4-0,6
0,6-0,8
0,8-1


1
0,8
0,6
0,4
0,2
0
1 2 3 4 5 6 7 8 9 10 11 12
Series10
Series7
Series4
Series1
0-0,2
0,2-0,4
0,4-0,6
0,6-0,8
0,8-1


1
0,8
0,6
0,4
0,2
0
1
2
3
4
5
6
7
8
Series7
Series5
Series3
Series1
0-0,2
0,2-0,4
0,4-0,6
0,6-0,8
0,8-1


1
0,8
0,6
0,4
0,2
0
1
2
3
4
5
6
Series6
Series5
Series4
Series3
Series2
Series1
0-0,2
0,2-0,4
0,4-0,6
0,6-0,8
0,8-1


1
0,8
0,6
0,4
0,2
0
1
2
3
4
Series4
Series3
Series2
Series1
0-0,2
0,2-0,4
0,4-0,6
0,6-0,8
0,8-1