INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

JUEGO PC: STRATEGO

Realizado por:
Jaime Benavent Alba

Dirigido por
Jose Ramón Portillo Fernández

Departamento
Matemática Aplicada I

Sevilla, DICIEMBRE de 2014

Agradecimientos

Es difícil acordarme y más nombrar a todas aquellas personas las cuales han puesto su granito de arena para animarme y ayudarme para poder acabar este trabajo, así que quiero dar las gracias sobre todo a:

Mi familia por ese apoyo incondicional en los momentos buenos y no tan buenos que he pasado frente al ordenador para poder finalizar este juego, a mi jefe de proyecto José Ramón Portillo Fernández, por su guía y camino mostrado para realizar un buen trabajo, mis amigos que me aconsejaban en las posibles mejoras del juego y por último a toda persona anónima en la red que solucionan aquellos problemas que no consigues resolver a cambio de nada. Como también a la página de sourceforge por disponer de tantos proyectos de código libre que me han servido de muchísima ayuda para la elaboración de muchas clases.

En definitiva, muchísimas gracias a todas las personas que han colaborado conmigo para que pueda desarrollar este juego para un pc.

Resumen

El proyecto se basa en la creación del conocido juego de mesa Stratego adaptado a PC.

Para ello hemos representado todas las operaciones que se pueden realizar, como son los movimientos tanto de atacar como defender, los turnos de juego, las estrategias para jugar contra la computadora y demás acciones que se ven en el juego. Es posible jugar una partida contra un jugador o la computadora, la cual tiene una inteligencia artificial basada en una serie de estrategias, las cuales se deciden al principio del juego.

La finalidad del juego, se basa en que uno de los jugadores de la partida se haga con la bandera del oponente, atacando y defendiendo con un ejército diferenciado por un rango para saber quién vence en cada enfrentamiento. El que lo consiga gana la partida.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

Un videojuego es una aplicación realizada por un ordenador, a través de los programadores y diseñadores gráficos, al cual pueden jugar un número limitado de personas, el primer objetivo es el entretenimiento. Un videojuego se muestra en un aparato electrónico (ordenador, videoconsola, smartPhones) donde los jugadores envían las funciones a realizar por la máquina, ésta le muestra al usuario una parte gráfica al instante.

Los primeros videojuegos modernos aparecieron en Estados Unidos en la década de los años 60 y desde entonces no han parado de avanzar y desarrollarse tecnológicamente. Hoy día existe muchísimos tipos de juegos, que van desde los juegos de mesas de toda la vida, aventuras gráficas, deportes, agilidad mental, plataformas, etc. Todos estos tipos de juego se han ido adaptando a los avances hardware qué tenía la máquina, mejorando tanto gráficamente como en jugabilidad.

DEFINICIÓN DE OBJETIVOS

El principal objetivo de este proyecto es la implementación del juego de mesa llamado **Stratego**.

La compilación del juego se puede dividir en varias partes:

- Implementar el juego en un lenguaje de programación, de tal manera que podamos obtener un ejecutable sencillo de usar.
- Hacer que el programa cumpla todas las reglas del juego, explicadas en **reglas del juego**, se puede ver en el índice.
- Conseguir que el juego tenga una inteligencia artificial de manera que podamos jugar una partida contra la computadora y que esta realice movimientos con sentido.
- Establecer unas estrategias de inicio y una dificultad, y así tener más rivalidad contra la computadora.
- Conseguir que se pueda jugar de dos jugadores por red, conectándose los dos equipos a través de sus direcciones IPs y abriendo el mismo puerto.
- Usar varias interfaces del juego para poder cambiar la apariencia del juego y hacerlo más ameno.

Para la implementación de estas partes usaremos el lenguaje de **Java**, ya que cuenta con una gran variedad de librerías que nos ayudaran con la interfaz del juego, de manera que sea intuitiva y fácil de usar.

La parte más importante de la aplicación será el desarrollo de la inteligencia artificial de la computadora, la cual se hace a través de un algoritmo de búsqueda basado en comprobar todos los movimientos posibles y elegir el mejor dentro de un tiempo que estará establecido con la dificultad del juego.

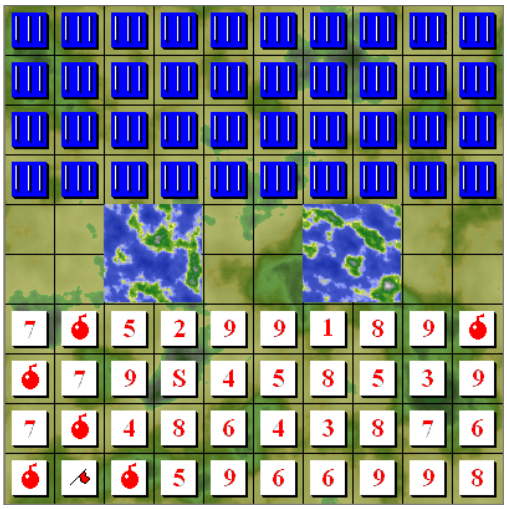
Para finalizar la aplicación se creara un archivo que será compatible para ejecutarse en cualquier sistema operativo (Windows, Linux o Mac).

ANÁLISIS DE ANTECEDENTES

Historia

Los orígenes de Stratego se remontan al tradicional juego de mesa chino "Selva" ya que tienen un tablero y algunas reglas similares.

La aplicación es una adaptación al juego de mesa Stratego, que apareció en Europa antes de la Primera Guerra Mundial, diseñado por la señorita Hermance Edan que presento la patente en 1908 de juegos de combates con piezas móviles sobre un tablero. Aunque hasta 1910 el señor Depaulis no sacó la primera versión del juego dividiendo los ejércitos en los colores rojo y azul.



Primera versión de Stratego

Las instrucciones y las explicaciones del juego se encuentran reflejadas en el **manual de usuario**, se puede mirar en el índice.

El juego es muy popular en los Países Bajos, Alemania y Bélgica, donde se celebran regularmente campeonatos nacionales y mundiales. La escena internacional "Stratego", en los últimos años, ha sido dominada por los jugadores de los Países Bajos.

La versión electrónica de "Stratego" fue presentada por Milton Bradley en 1982. Tiene características que hacen muchos aspectos del juego notablemente diferentes a los del Stratego clásico.

Tablero

El juego se desarrolla en un tablero cuadrado de 10x10.

En el medio del tablero hay dos lagos de 2x2 por los que no se puede pasar y el resto del tablero se puede recorrer en cualquier dirección, excepto en diagonal.

Tablero de mesa

Número de jugadores

El juego está basado en dos jugadores que coloca cada una 40 piezas dejando los lagos en medio.

En esta aplicación se ha desarrollado la manera de jugar contra la computadora dotándola de una inteligencia artificial.

Piezas

Todas las piezas tienen un rango o valor asociado. En la mayoría de los casos ese rango sólo sirve para determinar el resultado de los enfrentamientos con el rival, pero hay excepciones.

Una de las excepciones es la bomba, que no se puede mover y sólo puede ser eliminada por el minero. Sin embargo, la bomba elimina a cualquier otra pieza que la ataque.

Otra excepción es el espía, capaz de eliminar a la pieza de mayor rango, el mariscal, pero sólo si el espía ataca primero, no si es atacado por el mariscal. Sin embargo, el espía pierde en cualquier enfrentamiento con otra pieza.

Por último, el explorador es la única pieza que puede avanzar más de una casilla por turno, pero sin poder saltar sobre otra pieza.

Las piezas tienen distintos rangos, a continuación se muestra su rango de mayor a menor y el número de piezas que hay de cada color.

- 1 - Mariscal → 1
- 2 - General → 1
- 3 - Coronel → 2
- 4 - Comandante → 3
- 5 - Capitán → 4
- 6 - Teniente → 4
- 7 - Sargento → 4
- 8 - Minero → 5 (Desactiva bombas)
- 9 - Explorador → 8 (Puede recorrer cualquier distancia en línea recta)
- S - Espía → 1 (Puede eliminar al Mariscal si ataca primero)
- B - Bomba → 6 (Destruye a cualquier atacante, excepto mineros)
- F - Bandera → 1 (Cuando se ataca esta pieza se gana la partida)

Piezas

Reglas del juego

Turnos: El juego se desarrolla por turnos. Siempre comienza el color rojo, pudiendo realizar solo un movimiento y dará paso al siguiente jugador.

Posiciones validas: Se puede mover por todo el tablero, excepto si la casilla está ocupada por otra pieza o por dos grupos de casillas de 2x2 en las que no puede haber piezas. Suelen estar marcadas en el tablero como lagos (mientras el resto del tablero es un campo de batalla). Su función es evitar ataques frontales masivos y permitir desarrollar estrategias acumulando piezas a sus alrededores.

Movimientos permitidos: No se permiten movimientos en diagonal ni moverse entre dos casillas en tres turnos consecutivos. Cada jugador mueve una única pieza por turno. Los movimientos posibles son hacia delante y atrás, izquierda o derecha. La única excepción es el explorador, que puede moverse por todas las casillas que desee en línea recta. Las piezas no se pueden mover fuera del tablero ni a una casilla ocupada por una pieza del mismo ejército.

Ataques: Si una pieza se mueve a una casilla ocupada por una unidad del otro ejército se considera un ataque y ambas descubren su identidad (girándose). En este caso la pieza de menor rango es eliminada y la de mayor rango ocupa la posición de la anterior. Si las dos piezas tienen el mismo rango se eliminan ambas.

Reglas opcionales: Hay reglas concretas que se pueden acordar antes de comenzar la partida y que dan más variedad al juego. Solo son válidas para jugar contra la computadora.

- Mostrar todas las piezas.
    - Revelar piezas al ser eliminada.
    - No revelar al defensor (solo puede el explorador).
- Defender y no atacar.
- Aventajar al defensor en caso de empate.
- Eliminar la bomba después de un ataque.

Estrategias

El Stratego es un juego con conocimiento parcial del entorno. La planificación y el pensamiento estratégico juegan un papel destacado en este juego. También son muy importantes los aspectos psicológicos.

Estrategias básicas:

- Colocar inicialmente las piezas de modo que la bandera quede protegida, a la vez que se intenta despistar al contrario de su ubicación en el tablero.
- Disponer de las piezas de más rango para atacar
- Identificar la forma de movimientos de las piezas enemigas durante la partida, de manera que obtengamos una idea de su distribución en el tablero.
- Colocar las piezas más fuertes y/o bombas lejos de la bandera (aunque esto es arriesgado), con el fin de engañar a un oponente para intentar que ataque el lado equivocado.

La colocación de la bandera: Dado que el objetivo para ganar es capturar la bandera, su colocación es vital. Comúnmente se coloca en la fila de atrás, rodeado por dos o tres bombas de protección.

La exploración efectiva: Los exploradores son muy útiles para el final del juego, una vez que el tablero está más despejado. Pueden ser utilizados para identificar las bombas en la última fila, matar al espía si se delata o incluso capturar la bandera.

Las estrategias del espía: En la mayoría de las partidas, es recomendable tener un espía a la sombra de un general o coronel, puesto que estas piezas suelen ser susceptibles a ser atacadas por el mariscal. Manteniendo un general o coronel en la misma zona que el espía permite una retirada efectiva mientras el mariscal del oponente puede ser emboscado por el espía.

La estrategia de los mineros: Al descubrir las bombas del oponente, pueden optar por dejarlas en su lugar en vez de desactivarlas con un minero, con la intención de disminuir la capacidad de movimiento del enemigo. Al mantener a los mineros inmóviles en su propio territorio durante el inicio de la partida, un jugador puede confundir al oponente haciéndole creer que son bombas.

La protección de las piezas: Uno de los conceptos más importantes de Stratego es el desconocimiento y la desorientación, por lo que el manual recomienda llevar una pieza junto a otra que no sea mucho más fuerte que la otra.

Aprovechar una ventaja: Si un jugador tiene la suerte de haber ganado una ventaja sobre su oponente, vale la pena aprovechar esa ventaja atacando al oponente sus piezas de mayor rango para eliminarlas del tablero.

Atacar piezas desconocidas: Una estrategia arriesgada, que podría ser necesario cuando se va perdiendo, es atacar a una pieza desconocida e inmóvil con una pieza fuerte. Esta estrategia se basa en las probabilidades. Matemáticamente, la probabilidad es de 7 sobre 40, pero en realidad esta probabilidad puede ser mejorada si no se ataca a las piezas que pudieran ser bombas, o manteniendo un seguimiento de las piezas ya identificadas.

Finalidad del juego

El juego termina cuando uno de los dos oponentes encuentra la bandera del otro jugador y la ataca con cualquier pieza o cuando uno de los dos jugadores se queda sin piezas de movimiento (solo bombas y bandera). Cualquiera de las dos maneras es válida para ganar la partida.

También queda la posibilidad de pactar un empate si la partida se alarga mucho y siempre que los dos oponentes estén de acuerdo.

APORTACIÓN REALIZADA

La base de estudio de este proyecto es obtener un juego que sea fácil de interpretar y tenga entre otras opciones una modalidad de juego contra la computadora gracias a un algoritmo que permita una buena inteligencia artificial.

Por suerte hemos reutilizado gran parte del código de varios proyectos ya creados en sourceforge (en la **Bibliografía** están las direcciones web) y modificado de tal manera que no tenga ningún error en el código para que funcione y tenga las prestaciones que me interesaban para la realización del juego.

Aunque el mayor tiempo del desarrollo ha estado basado en el estudio de varios proyectos para reutilizar su código, me ha servido para desarrollar otras funciones en el proyecto como añadir la carga de estrategias para el usuario al igual que la computadora. También hemos realizado varios cambios en la interfaz del juego para que tenga una interfaz fácil de entender para una posterior ejecución de cualquier usuario. Dentro de varias clases hemos tenido que añadir métodos y modificar varias líneas de código para que todo funcione correctamente. Y para el modo de juego de **Dos Jugadores** después de comprobar el código y hacerle varias pruebas, conseguimos hacerlo funcionar perfectamente.

También tenemos la posibilidad de volver a reutilizar el código para futuras mejoras, lo que podría ayudar a un posterior desarrollo o estudio de la aplicación a diseñar.

Ahora nos planteamos una serie de cuestiones para entender el juego.

Partimos del juego de mesa compuesto de un tablero, piezas como objetos materiales, y el humano es el que se encarga de darle inteligencia para poder competir y ganar a su adversario.

Pero ¿cómo sería si lo quisiéramos llevar a un modo virtual, del que no se necesite nada material, y se pueda abrir y cerrar en el momento que deseemos?

Es más, ya no solo eso, uno de los problemas que podemos encontrarnos jugando en la vida real, es la infracción de alguna regla de dicho juego de mesa. No puedo siempre estar seguro de que he realizado un movimiento permitido, o como es más lógico, que el adversario realice un movimiento que no sea válido según las normas.

Una de las aportaciones de este proyecto, es poder jugar de la misma manera o incluso más cómodamente que se jugaría si dispusiéramos del material físico necesario, pero en este caso, con un solo clic de ratón.

Además, en este caso, no cabe duda de que un jugador, ya sea humano o artificial, realiza un movimiento permitido, y no se ha equivocado, ya sea intencionadamente o no. Para ello se utilizan técnicas que calculen antes de poder posicionar la pieza, si ese movimiento está permitido o no, si no lo está, obliga incondicionalmente a que el jugador rectifique. Por tanto, esto hace que el juego sea más seguro y no esté dado a errores, para que se pueda jugar con tranquilidad.

Hasta ahí bien, pero ¿y si nos planteamos si hay alguna forma mejor de jugar? O en un determinado momento de la partida, ¿Cuál será el movimiento más correcto? En una partida normal, un humano puede adelantarse un número de jugadas limitadas, dependiendo del punto en el que se encuentre la partida y la habilidad del jugador. Pero ¿son suficientes? Para ello, gracias al código reutilizado, después de entenderlo bien comprobamos que hay varias formas de que una máquina se encargue de realizar grandes cálculos, y nos permita conocer cuál es el movimiento correcto. Solo tendremos que hacer unos cambios para adaptarlo a nuestras necesidades.

Pero para que la máquina juegue bien, hay que darle unas pautas básicas de las normas del juego, “enseñarle a jugar”, y además, enseñarle como elegir el mejor movimiento. Lo cual nos lleva a otro problema a plantearse, ¿cómo le hago saber a la computadora si un movimiento es bueno o malo?

No es fácil elegir una opción que nos permita saber en cada momento cuál es la mejor jugada, pero se intenta acercarse lo máximo posible a dicho movimiento. Para juegos simples, (como por ejemplo 3 en raya), es fácil llegar a una solución final, ya que no se compone de una gran complejidad y el movimiento correcto es bastante sencillo de calcular. La cosa se complica cuando no hablamos de 7 u 8 movimientos posibles, sino cuando podríamos hablar de millones de posibles jugadas.

Sabemos que las máquinas son capaces de calcular miles de operaciones por segundo mucho más rápido que cualquier humano, pero aún así, tardarían muchísimo en calcular todas las opciones posibles para este tipo de juegos más complejos.

En todo esto se basa lo más importante que se intenta aportar en este proyecto. Se estudian diferentes formas de afrontar los problemas y determinar la mejor opción, y el coste que puedan conllevar cada uno de ellos.

ANLISIS TEMPORAL

Las horas estimadas para un proyecto de **Ingeniería Informática Técnica** son **270 horas.**

Esta estimación, es un cálculo aproximado, ya que puede variar mucho las horas empleadas en un proyecto que en otro, aunque el trabajo realizado deba coincidir cercanamente con esa estimación.

Para este proyecto, se ha invertido un total de **282 horas**, lo que implica 12 horas más invertidas en dicho proyecto. Eso implica un **4,4%** más, empleado para finalizar este desarrollo. Este porcentaje es muy razonable, ya que es muy común que se comentan errores en el cálculo estimado de tiempo necesitado. Y en este caso, se puede apreciar que no es demasiado elevado el error cometido y por tanto el tiempo de exceso empleado.

Este tiempo utilizado, puede diferenciarse en varios grandes grupos para realizar una separación y poder estudiarlo. Estos grupos podrían ser:

Estudio previo del entorno de desarrollo. Antes de comenzar a programar, es necesario conocer el entorno en el que se va a trabajar. Previamente se debe conocer que programas se van a utilizar, el lenguaje de programación, etc. En este caso, Usaremos el programa **Eclipse** y **Java** como lenguaje de programación, ya que han sido impartidos en clase y tengo bastante conocimiento sobre ambos. El tiempo empleado en este punto es de **1 hora.**

Estudio previo del desarrollo del software. Este estudio se basa en conocer cómo poder implementar y que requisitos y problemas nos puede dar el desarrollo del proyecto para que, uniéndolo con el anterior estudio, podamos comenzar a desarrollar. Este estudio en este caso se basa en cómo conseguir que se cumplan las normas del juego, y como conseguir una eficiencia por la computadora. El tiempo empleado en este punto es de **25 horas** aproximadamente.

Diseño e implementación del software. Al usar un proyecto reutilizado lo más complicado fue entenderlo entero para poder corregir los errores que me iban surgiendo durante el estudio del código para poder modificarlo a mis necesidades y dar lugar a un programa ejecutable, ésta es la parte que más tiempo nos ha consumido. El tiempo empleado en este punto es de 1**25 horas** aproximadamente.

Pruebas y corrección de errores. Este apartado no se trata simplemente de corregir los errores que han surgido después de la implementación del proyecto. Como se trata de un proyecto de investigación, mientras se va diseñando e implementando, surgen problemas y nuevas ideas, lo que implica supuestos errores y correcciones para posibles mejoras. Por tanto este apartado se podría considerar también de implementación, pero de una manera un poco diferente. Además el tiempo empleado para las pruebas realizadas para el correcto funcionamiento final también han sido recogidas en este apartado. El tiempo empleado en este punto es de **50 horas** aproximadamente.

Redacción de la memoria. A la hora de realizar un proyecto, es necesaria la construcción de una memoria donde se notifican los pasos y las determinaciones que se han ido tomando, dando razones de las mismas. Sin una memoria el proyecto podría quedar indocumentado y por tanto incompleto. El tiempo empleado en este punto es de 5**0 horas** aproximadamente.

Entrevistas con el tutor. En las entrevistas se le plantea al tutor las dudas que van surgiendo y sirve para que nos pueda guiar y corregir los errores que se puedan estar cometiendo. Se emplea menos horas ya que las entrevistas suelen ser cortas, pero aun así, son de gran utilidad ya que sin ellas puede que el proyecto se saliera del camino que debería de tomar. El tiempo empleado en este punto es de **6 horas** en 6 entrevistas.

Diseño y redacción de la presentación. Finalmente, será necesaria la creación de una presentación para exponer todas las ideas que se han ido desarrollando a lo largo del proyecto. Una especie de resumen que recoja lo más importante y que a la vez, quede bien detallado. El tiempo empleado en este punto es de **25 horas** aproximadamente**.**

COSTES DE DESARROLLO

Para que un proyecto sea satisfactorio, los costes de desarrollo deben estar dentro de unos límites establecidos según el tipo del mismo. En este caso, al tratarse de un desarrollo enteramente de programación, los únicos costes que se pueden tener en cuenta son:

- **Utilización del Entorno de trabajo (Portátil).**
- **Gasto de luz invertido.**
- **Coste de los programas utilizados para el desarrollo.**

El entorno de trabajo que se ha utilizado ha sido un portátil para avanzar en todo lo referente a la ejecución del proyecto (programación, diseño, memoria, entrevistas, etcétera).

El portátil ya disponíamos de él, por tanto no supone coste alguno. Simplemente el mínimo desgaste que se le pueda producir.

Otro de los costes posibles a incluir que se han comentado es el gasto de luz consumida. Quizás este punto puede ser el que más gasto haya podido producir, pero el gasto de luz utilizado para programar es mínimo, se podría llevar a cabo una estimación y obtener un coste aproximado, pero ese coste no tendría sentido ya que la luz gastada expresamente para el proyecto es bastante bajo y aparte gran parte del proyecto se ha desarrollado en las bibliotecas de la universidad de Sevilla.

El software utilizado para desarrollar esta aplicación ha sido descargado mediante la web de **Eclipse** gratuitamente y el **Office 2010** por el repositorio también gratuito que ofrecía la universidad para los alumnos de informática en el MSDN.

Por tanto, se podrían hacer varias estimaciones de costes, sobre el gasto por la utilización del portátil, o el gasto de luz consumido… pero son gastos que son necesarios y tan pobres que son despreciables, por lo que el coste producido es mínimo.

ANÁLISIS DE REQUISITOS

En este proyecto se pretende cumplir los siguientes requisitos:

- Ejecutarse correctamente en un PC con un sistema operativo actual instalado.
- Tener opción de seleccionar el número de jugadores posibles, dando opción a poder participar entre 1 o 2 jugadores.
- Debe existir la opción de jugar una partida participando la computadora.
- Cumplir correctamente las normas del juego establecidas en el apartado de **reglas del juego**.
- Dicha computadora debe ser capaz de completar una partida de manera competente y en un tiempo razonable.

Para este tipo de proyectos, los requisitos no son excesivos. Pero aunque no sean demasiados, resulta bastante complicado llegar a obtener una solución válida para cumplir cada uno de estos objetivos. Ya que, algunos de ellos, abarca una complicación interna bastante elevada.

Estos requisitos, dan libertad a la hora de elegir lenguaje de programación y sistema operativo, pero a su vez, hay que tener cuidado a la hora de intentar cumplir el resto de requisitos necesarios.

DISEÑO

En este apartado se intenta razonar las determinaciones informáticas que se han tomado para la realización del proyecto, como la elección del sistema operativo, el lenguaje de programación, la presentación y la ejecución de la aplicación.

Uno de los objetivos del proyecto es ejecutar la aplicación en cualquier PC actual y aunque se pueda desarrollar en los tres sistemas operativos más importantes, como Windows, Mac o Linux, nos quedamos con Windows, ya que después se puede adaptar a los demás sistemas operativos.

El lenguaje utilizado en este proyecto es **Java**, y gracias al programa **Eclipse** hemos conseguido desarrollar el juego.

La presentación del juego se basa en varias pantallas y un menú para elegir la opción que quieras antes de empezar a jugar.

Pantalla de presentación: Se trata de una imagen que representa al juego y una vez que empiezas a jugar ya no la vuelves a ver.

Pantalla de menú: Esta pantalla nos permitirá elegir una serie de opciones antes de empezar la partida, como el número de jugadores, la dificultad, la estrategia que seguirá la computadora o alguna reglas opcionales que si estas en modo **2 Jugadores** no son accesibles.

Pantalla de juego: Esta pantalla nos permite ver el tablero del juego y nuestras fichas para colocarlas o cargarlas automáticamente con estrategias previamente diseñadas y posteriormente comenzar la partida.

La ejecución de la aplicación: Una vez terminado todo el código elaboramos un archivo con extensión **jar** desde el **Eclipse** y ya solo tenemos que hacer doble clic sobre el archivo.

Diseño gráfico: La interfaz gráfica está basada en una búsqueda en internet de varias imágenes de varias versiones del juego. Y con ayuda del **paint** retocarlas y adaptarlas a nuestra aplicación.

IMPLEMENTACIÓN

Para la implementación de la aplicación hemos reutilizado varios proyectos de sourceforge (en la **Bibliografía** están las direcciones web) para obtener una serie de clases dentro de varios paquetes, explicados a continuación, y así poder tener bien organizado el código. También hemos usado varias librerías de Java previamente estudiadas en la API de Oracle para saber sus funciones.

A continuación explicamos las clases que han sido modificadas y las que más importancia tienen.

Clases:

- **Juego:** En este paquete se encuentran las clases necesarias para comenzar el juego en cualquier modalidad. Las tenemos explicadas a continuación:

  Apariencia: En esta clase se declaran las distintas apariencias del juego, desde la pantalla de inicio, el fondo, las piezas y los iconos de la barra de herramientas. De las cuales, en la parte de la imagen de inicio tuvimos que realizar varios cambios para que cargase bien la imagen ya que tenía fallos de compilación, y también tuvimos que eliminar un botón y crear uno nuevo cambiando varias líneas de código hasta que funcionase correctamente.

  Ayuda: En esta clase se explica las instrucciones del juego. Que fueron traducidas al español ya que todo el juego ha sido traducido., cambiándole todos los *String* al español.

  BotonesPiezas: En esta clase se desarrolla los distintos tipos de fichas que hay para cada equipo. La cual no he tenido que modificar ya que venía bien creada para mis necesidades.

  CompControles: Es una interfaz que la implementa la clase IAMotor. A la que no le realice ningún cambio.

Cargar: En esta clase se desarrolla el menú que hay en los ajustes para elegir el tipo de táctica para la computadora. La cual carga un fichero con una estrategia ya creada y se lo asigna al usuario o la computadora. En esta clase hice varios cambios para comprender la manera de carga de ficheros, aunque al final no hacían falta cambios.

Cliente: En esta clase se desarrolla el cliente, que será el jugador rival con el que jugaremos en red contra él. Gracias a esta clase aprendí a usar la clase Socket que sirve para conectar ambos jugadores media una dirección IP y un puerto para establecer la conexión.

ControlesUsuario: Es una interfaz que la implementa la clase IAMotor. A la que no le realice ningún cambio.

IA: En esta clase se desarrolla la colocación de las piezas, el inicio del juego y los movimientos posibles de la computadora comprobando distintas posibilidades. Después de estudiar bien la colocación de las piezas y ver que solo se cargaba el fichero para la computadora, desarrolle en el método de carga de piezas en el tablero un nuevo bucle que también cambiara la estrategia del usuario y así poder empezar los dos jugadores con las piezas sobre el tablero y con estrategias previamente desarrolladas por la clase **Editar.**

IAMotor: En esta clase se desarrolla las distintas vistas del juego desde que se inicia hasta el fin de la partida. Ya que en su interior tenemos los métodos que dan vida a los botones de la barra de herramientas. En esta clase nos encontramos con un gran error al principio del proyecto, no me cargaban las piezas rojas o azules, dependiendo de cual elijas al principio, hasta que al final añadiendo un par de líneas de código en el método **JuegoNuevo** resolvió el problema y pude ver por primera vez todas las piezas colocadas sobre el tablero.

MovimientoOponente: Es una interfaz que la implementa la clase Vista. A la que no le realice ningún cambio.

Opciones: En esta clase se desarrolla toda la configuración del juego, varias opciones de juego y la dificultad o las tácticas que empleara la computadora para jugar. En esta clase no realice ningún cambio ya que después de estudiar bien su funcionamiento descubrí los distintos algoritmos que ha establecido en la barra de dificultad para complicar el juego y hacer pensar más a la computadora contra más dificultad.

StrategoMain: En esta clase se ejecuta el juego Stratego. En la cual no realice ningún cambio ya que solo se basa en ejecutar la clase **Vista**, la cual es el hilo final del funcionamiento del proyecto.

TestingTablero: En esta clase se testea que los movimientos y ataques sean válidos. Tampoco le tuve que realizar cambios ya que es una clase muy simple compuesta por un par de métodos fáciles de entender.

Vista: En esta clase se muestran todas las vistas del juego, las actualizaciones de los movimientos, los iconos de la barra de herramientas y varias condiciones para jugar en red. En esta clase tuve los mismos problemas que en la clase **Apariencia** ya que no me cargaba la imagen de bienvenido y después de varias pruebas entre las dos clases comprobé que el error se encontraba en la otra clase y esta comenzó a funcionar correctamente.

- **Juego.Editar:** En este paquete se encuentran las clases necesarias para elaborar las estrategias de inicio y poder cargarlas en una partida.

  Editar: En esta clase se desarrolla la manera de elaborar las estrategias de inicio que tendrán los jugadores. La cual se encarga de guardar en un fichero la estrategia para cargarla posteriormente

EditarControles: Es una interfaz que la implementa la clase EditarTablero. A la que no le realice ningún cambio.

EditarMain: En esta clase se ejecuta el Editar y así empezar a diseñar las estrategias. No le tuve que realizar cambios.

EditarTablero: En esta clase se desarrolla la manera de añadir o eliminar piezas para que funcione la clase Editar. A la que no le realice ningún cambio.

- **Servidor:** En este paquete se encuentran las clases para crear el servidor y poder jugar en red. El cual no tuve que modificar ya que documentándome bien sobre la clase Socket entendí su funcionamiento.

  Controlador: En esta clase se controlan todos los casos que van surgiendo a la hora de conectarse en red. En esta clase elimine varias líneas de código que lo único que hacían era volver a preguntarse lo mismo varias veces o simplemente no llegaban a ejecutarse, y comprobando siempre que funcione bien.

  Hash: Toda clase debe proveer de un método hashCode que permite recuperar el Hash Code asignado, por defecto, por la clase Object. Es como una clase por defecto que siempre hace lo mismo, por lo que no la modifique nada.

  Juego: En esta clase están desarrollados los casos para comenzar el juego en red. En la cual la clase **Socket** crea los dos jugadores asignados al servidor o el cliente y así poder empezar la partida.

  Mensaje: En esta clase se declaran las variables de estados en red que aparecerán posteriormente convertidas en mensajes durante el juego. No le realice cambios ya que es como una interfaz.

Servidor: En esta clase se crea el servidor, que será necesario para que el cliente se conecte y se pueda empezar a jugar. La cual es el servidor de la clase **Socket** y así tener la conexión Cliente-Servidor deseada. Y poder comprender del todo como funciona esa clase gracias al **Cliente** desarrollado en el paquete **Juego.** Ya que gracias a estas dos clases y la documentación leída en internet no tuve que realizar ningún cambio ya que la entendí perfectamente.

ServidorControlador: En esta clase se desarrollan todas las conexiones para poder jugar. Ya que se instalan los controladores necesarios en el servidor para así poder llamar al **Cliente** y que este se conecte para empezar la partida.

ServidorMain: Esta clase se encarga de asignar un puerto que tendrá que insertar cada jugador a la hora de comenzar. Después llama a la clase **ServidorControlador** para poder empezar a jugar. Esta clase imprime tu IP y tu puerto para su posterior conexión.

ServidorMotor: En esta clase se crean los estados comentados en la clase **Mensaje** para su posterior uso durante la conexión del juego entre usuarios.

ServidorRemoto: En esta clase se desarrollan los métodos para para que la clase **Servidor** funcione correctamente.

- **Servidor.Remoto:** En este paquete se encuentran las clases para comprobar la conexión Cliente-Servidor.

  ControladorRemoto: En esta clase tenemos los métodos para hacer pruebas de conexión entre Cliente-Servidor, y comprobar que funcionan gracias a la clase **RemotoMain**.

  RemotoMain: En esta clase se ejecuta lo necesario para probar los métodos de la clase **ControladorRemoto**. Esta clase comprueba que las conexiones Cliente-Servidor funcionan correctamente.

- **Stratego:** Este paquete se basa en crear las clases básicas con todos los tipos de objetos que existen y que serán usadas por el paquete **Juego**. En este paquete se han modificado pocas clases ya que estaban bien implementadas para mis necesidades.

  Ajustes: En esta clase se declaran las opciones de juego que habrá en la pestaña de Ajustes, dándole unos valores iniciales. Los cuales se pueden cambiar durante la partida. No le realice cambios ya que era una clase bastante simple que se basa en iniciar variables que acabaran siendo botones.

  ***Estado*:** En esta clase se declaran los estados del juego (iniciando, jugando o parado). Que sirven para saber en que momento del juego te encuentras cuando tienes el código por delante. Y tampoco le realice cambios.

  Motor: En esta clase se desarrolla la creación de la partida.

  Movimiento: En esta clase se desarrollan los posibles movimientos que tiene una pieza gracias a la clase **Puntero** que se encarga de marcar su posición para su posterior avance.

  Pieza: En esta clase se desarrollan las piezas. Dándoles un color (rojo o azul) y un rango. Es una clase muy simple que se basa en métodos getter y setter por lo que no fue modificada.

  Puntero: En esta clase se declaran las variables 'x' e 'y' para la creación de las posiciones del tablero y poder desarrollar el movimiento de las piezas y saber donde se encuentran. Se basa en otra clase bastante simple que solo declara dos variables y les asigna un valor. Por lo que tampoco fue modificada.

  Rango: En esta clase se desarrollan los rangos de cada pieza. Simplemente a través de un switch se estudia cada rango y se le asigna un valor para su posterior uso en el juego. Al ser tan simple tampoco se modificó.

Tablero: En esta clase se desarrolla el tablero del juego, creándose los lagos centrales por los que no se puede pasar, el resto del tablero, la colocación de las piezas y los posibles movimientos de cada pieza y sus ataques. Esta clase fue de las primeras en intentar desarrollar gracias a un ejemplo visto en internet y tras finalizar su construcción la intente implementar en esta clase y tras comprobar como había elaborado el tablero en algún proyecto de sourceforge. Solo tuve que juntarlos corrigiendo sus errores y adjudicarle los demás métodos que venían en esta clase. Con lo que obtuvimos un nuevo tablero para empezar a desarrollar sobre él todas las jugadas posibles.

Librerías:

- **Java.awt:** Esta clase contiene todas las clases para crear interfaces de usuario y para la pintura de gráficos e imágenes.

  BorderLayout: La clase BorderLayout establece un contenedor para organizar y cambiar el tamaño de sus componentes para encajar en cinco regiones: norte, sur, este, oeste y centro.

  Rectangle: La clase Rectangle especifica un área en un espacio de coordenadas que está encerrado por el punto superior izquierda del objeto Rectangle (x, y) en el espacio de coordenadas, su anchura, y su altura.

  Graphics: La clase Graphics es la base abstracta para todos los contextos gráficos que permiten una aplicación para dibujar sobre los componentes que se realizan en diversos dispositivos, así como en las imágenes fuera de la pantalla.

  Graphics2D: La clase Graphics2D extiende de la clase Graphics para proporcionar un control más sofisticado sobre geometría, transformaciones de coordenadas, la gestión del color y la disposición del texto.

  Image: La clase Image es la superclase de todas las clases que representan imágenes gráficas.

image.BufferedImage: La subclase BufferedImage describe una imagen con un buffer accesible de datos de imagen.

image.AffineTransformOp: Esta clase utiliza una afín para realizar una aplicación lineal de coordenadas 2D en la imagen de origen o de la trama de coordenadas 2D de la imagen de destino o en la trama.

image.BufferedImage: La subclase BufferedImage describe una imagen con un buffer accesible de datos de imagen.

Event: Proporciona las interfaces y clases para hacer frente a diferentes tipos de eventos disparados por componentes AWT.

event.ActionEvent: Este evento indica que el usuario quiere un poco de acción que se produzca.

***event.ActionListener*:** La interfaz receptora recibe eventos de acción.

event.KeyEvent: Un evento que indica que una pulsación de tecla se produjo en un componente.

event.MouseEvent: Un evento que indica que una acción de ratón se produjo en un componente.

MouseListener: La interface receptora se prepara para la recepción de "interesantes" eventos de ratón (prensa, comunicado, haga clic en, entrar, y salir) en un componente.

MouseMotionListener: La interfaz receptora se prepara para la recepción de eventos de movimiento del ratón sobre un componente.

ComponentEvent: Un evento de bajo nivel que indica que un componente se movió, cambiando el tamaño o modificando su visibilidad (también, la clase raíz de los otros eventos a nivel de componente).

ComponentListener: La interfaz receptora se prepara para la recepción de componentes.

WindowEvent: Un evento de nivel bajo que indica que una ventana ha cambiado su estado.

WindowListener: La interfaz receptora se prepara para la recepción de eventos de ventana.

Color: La clase Color se utiliza para encapsular colores en el espacio de color sRGB por defecto o los colores en los espacios de color arbitrarios identificados por un ColorSpace.

geom.AffineTransform: La clase AffineTransform representa una transformación 2D que realiza una aplicación lineal 2D que coordina a otras coordenadas 2D que conserva la "rectitud" y "paralelismo" entre líneas.

GridLayout: La clase GridLayout es un controlador de distribución que establece los componentes de un contenedor en una cuadrícula rectangular.

MouseInfo: La clase MouseInfo proporciona métodos para obtener información sobre el ratón, como la posición del puntero del ratón y el número de botones del ratón.

- **Java.io:** Proporciona para la entrada y salida del sistema a través de flujos de datos, serialización y el sistema de archivos.

  File: Una representación abstracta de las rutas de acceso de archivos y directorios.

  BufferedReader: Lee texto de una corriente de caracteres de entrada, amortiguando caracteres a fin de proporcionar una lectura eficiente de caracteres, matrices y líneas.

  ***FileReader*:** Clase de conveniencia para la lectura de caracteres de un archivo.

FileWriter: Clase de conveniencia para la escritura de caracteres de un archivo.

BufferedWriter: Escribe el texto a un flujo de caracteres-salida, amortiguando los caracteres con el fin de prever la escritura eficiente de los caracteres individuales, matrices y cadenas.

IOException: Señales de que se ha producido una excepción de E/S de una cierta clase.

DataInputStream: Una corriente de entrada de datos permite a una aplicación leer tipos de datos primitivos de Java a partir de un flujo de entrada subyacente de una manera independiente de la máquina.

DataOutputStream: La clase DataOutputStream permite a una aplicación escribir tipos de datos primitivos de Java en un flujo de salida de una manera portátil.

InputStreamReader: La clase InputStreamReader es un puente de flujos de bytes a flujos de caracteres: Se leen bytes y los decodifica en caracteres utilizando un juego de caracteres especificado.

PirntStream: La clase PrintStream añade funcionalidad a otra secuencia de salida, es decir, la capacidad de imprimir representaciones de diversos valores de datos convenientemente.

- **Java.util:** Contiene el marco de las colecciones, las clases legacy collection, model, event, date y time en las instalaciones, internacionalización, y las clases de diversos servicios públicos (una cadena tokenizer, un generador de números aleatorios y una matriz de bits).

  Scanner: Un escáner de texto simple que puede analizar los tipos y cadenas primitivas utilizando expresiones regulares.

ArrayList: Implementación de tamaño variable-matriz de la interfaz de la lista.

Random: Un ejemplo de esta clase se utiliza para generar una corriente de números pseudoaleatorios.

concurrent.Semaphore: Un semáforo de conteo.

concurrent.locks.ReentrantLock: La clase ReentrantLock tiene una exclusión mutua con el mismo comportamiento básico y la semántica como el bloqueo implícito de un monitor para acceder usando métodos y declaraciones sincronizados, pero con capacidades ampliadas.

zip.ZipFile: Proporciona clases para leer y escribir los formatos ZIP y GZIP de archivos estándar.

Collections: Esta clase se compone exclusivamente de métodos estáticos que operan o devuelven colecciones.

InputMismatchException: Lanzado por un escáner para indicar que la señal recuperada no coincide con el patrón para el tipo esperado, o que la señal está fuera del rango para el tipo esperado.

NoSuchElementException: Lanzada por el método nextElement de una enumeración para indicar que no hay más elementos en la enumeración.

- **Java.net:** Proporciona las clases para la implementación de aplicaciones de red.

Socket: Esta clase implementa sockets de cliente (también llamados simplemente "tomas").

ServerSocket: Esta clase implementa sockets de servidor.

UnknownHostException: Lanzado para indicar que la dirección IP de un host no se pudo determinar.

***URL*:** La clase URL representa un localizador uniforme de recursos, un puntero de un recurso en la World Wide Web.

InetAddress: Esta clase representa una dirección de Protocolo de Internet (IP).

- **Java.security:** Proporciona las clases e interfaces para el marco de seguridad.

  MessageDigest: Esta clase MessageDigest ofrece a aplicaciones la funcionalidad de un mensaje de un algoritmo resumen, como MD5 o SHA.

- **Javax.swing:** Proporciona un conjunto de componentes "ligeros" (todo el lenguaje Java), en la medida de lo posible, el labor de la misma en todas las plataformas.

  JOptionPane: *La clase* JOptionPane hace que sea fácil para que aparezca un cuadro de diálogo estándar que solicita a los usuarios de un valor o les informa de algo.

  JFrame: Una versión extendida de java.awt.Frame que añade soporte para la arquitectura de componentes JFC/Swing.

  JPanel: La clase JPanel es un contenedor ligero genérico.

  JScrollPane: Proporciona una vista desplazable de un componente de peso ligero.

  JTextArea: La clase JTextArea es un área multi-línea que muestra el texto sin formato.

  JButton: Una implementación del tipo de botón "push".

  JCheckBox: Una implementación de una casilla de verificación - un elemento que puede activarse o desactivarse, y que muestra su estado para el usuario.

  JLabel: El área de visualización de una cadena corta de texto o una imagen, o ambos.

JSlider: Un componente que permite al usuario seleccionar gráficamente un valor deslizando un botón dentro de un intervalo acotado.

ImageIcon: Una implementación de la interfaz de iconos que pinta iconos a partir de imágenes.

JFileChooser: La clase JFileChooser proporciona un mecanismo sencillo para que el usuario elija un archivo.

JMenu: Una implementación de un menú, una ventana emergente que contiene la clase JMenuItems que se muestra cuando el usuario selecciona un elemento en la clase JMenuBar.

JMenuBar: Una implementación de una barra de menú.

JMenuItem: Una implementación de un elemento en un menú.

KeyStroke: Una pulsación de tecla representa una acción clave en el teclado, o dispositivo de entrada equivalente.

- **Javax.imageio:** El principal paquete de la API de Java de imagen de E/S.

  ImageIO: Una clase que contiene métodos estáticos de conveniencia para localizar ImageReaders y ImageWriters, y la realización sencillos de muestras de codificación y decodificación.

MANUAL DE USUARIO

Una vez instalado todos los componentes necesarios para la utilización de la aplicación, se detallará paso a paso el manejo del programa con todas sus características y opciones.

Para ejecutar el programa, nos dirigimos a la carpeta donde está contenido el ejecutable junto con sus archivos gráficos. La ubicación de dicho ejecutable no es relevante, ya que no es necesario colocarlo en ningún directorio raíz predeterminado. Lo que si es necesario es que dicho ejecutable esté siempre acompañado de las texturas pre compiladas utilizadas en la aplicación.

Abrimos el archivo ejecutable, y al hacerlo, se iniciará la aplicación.

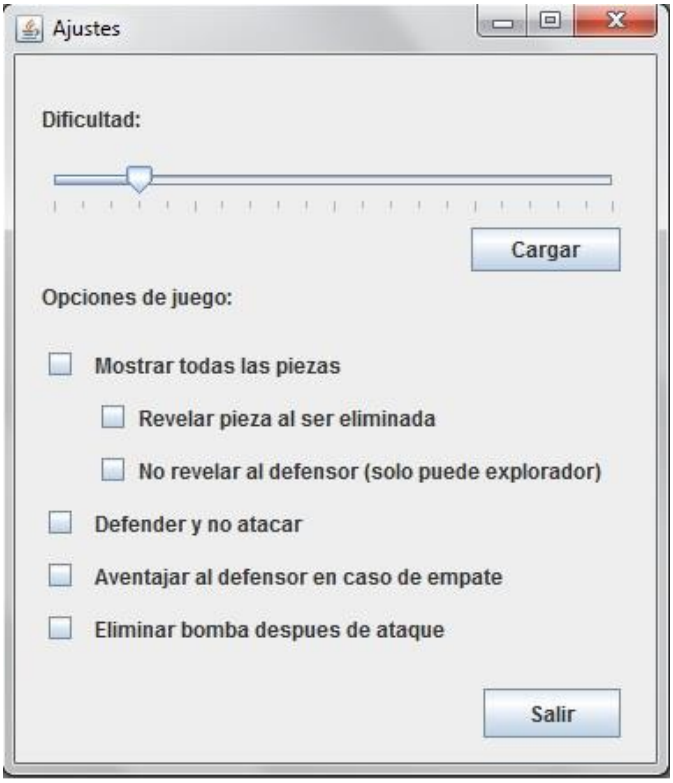
La aplicación está dividida en 4 pantallas diferentes:

Pantalla de bienvenida: Es la primera pantalla que aparece. En ella visualizamos la imagen de inicio.

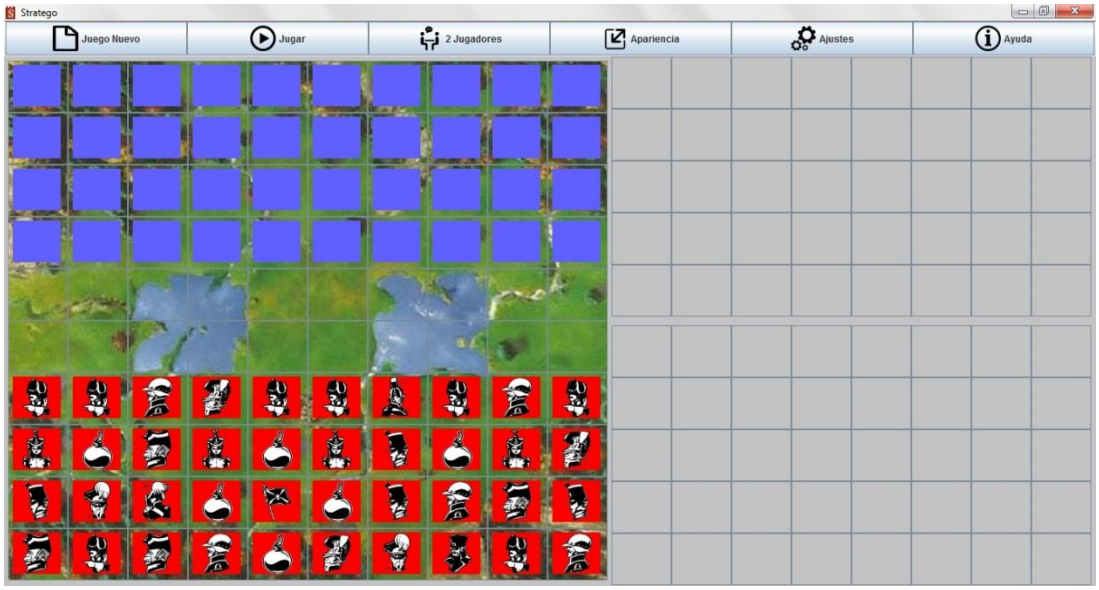
Para avanzar solo tenemos que pulsar en **Juego Nuevo**.

Tablero: Ahora tenemos que colocar las piezas.

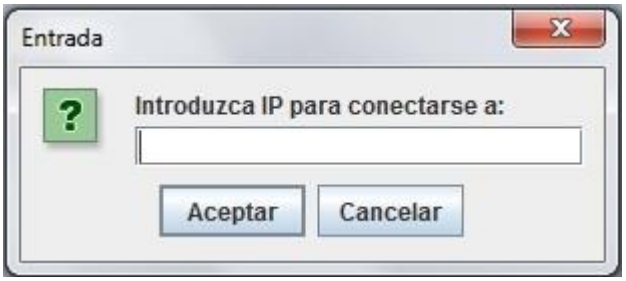
Ajustes: Después de colocar las piezas solo nos queda establecer las opciones de juego, la dificultad o las estrategias.



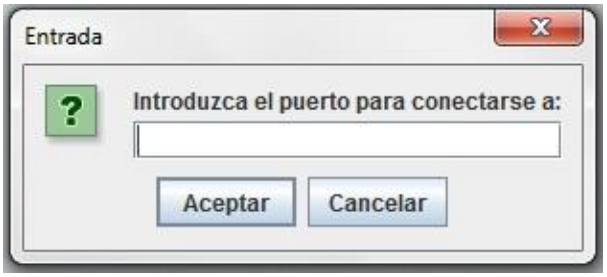
Jugar: Una vez establecido todo ya solo nos queda empezar la partida dándole a **Jugar**



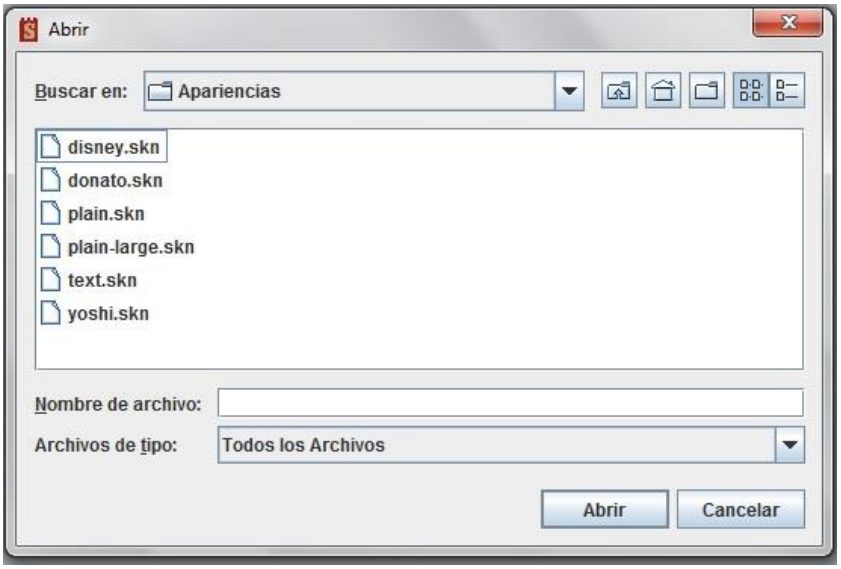
Dos jugadores: Tras pulsar **2 Jugadores** te pide que introduzca la IP del jugador contrario:



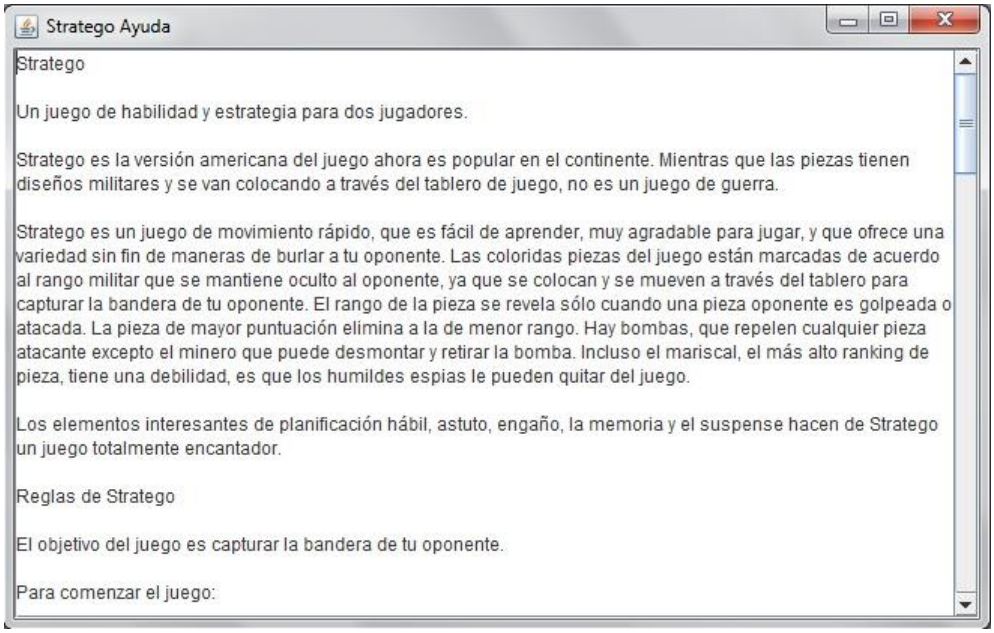
Y tras pulsar aceptar tenéis que meter los dos el mismo puerto y volver a pulsar aceptar. Y empezara la partida.



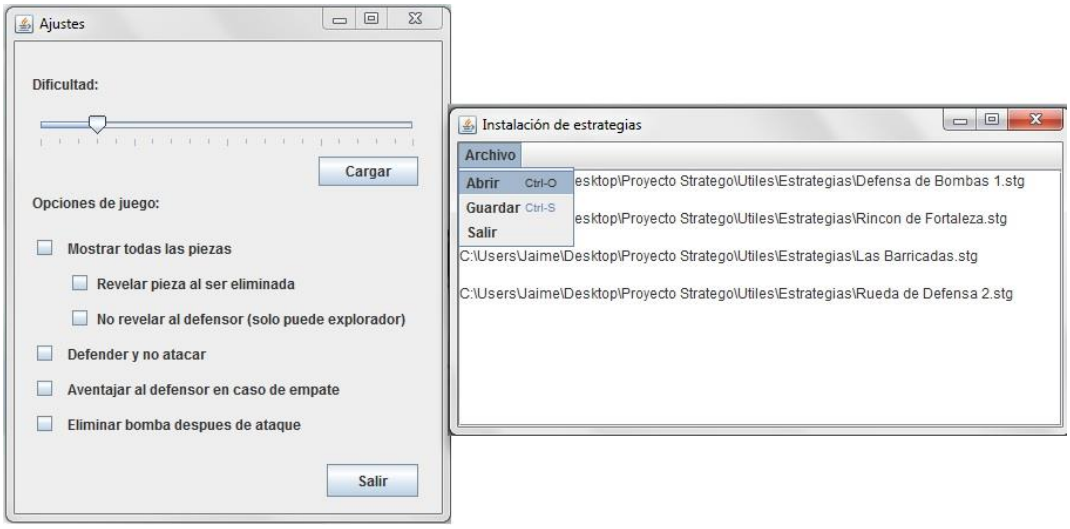
Apariencia: Tras pulsar el botón de **Apariencia** solo hay que localizar la carpeta donde están guardadas las apariencias ya diseñadas.



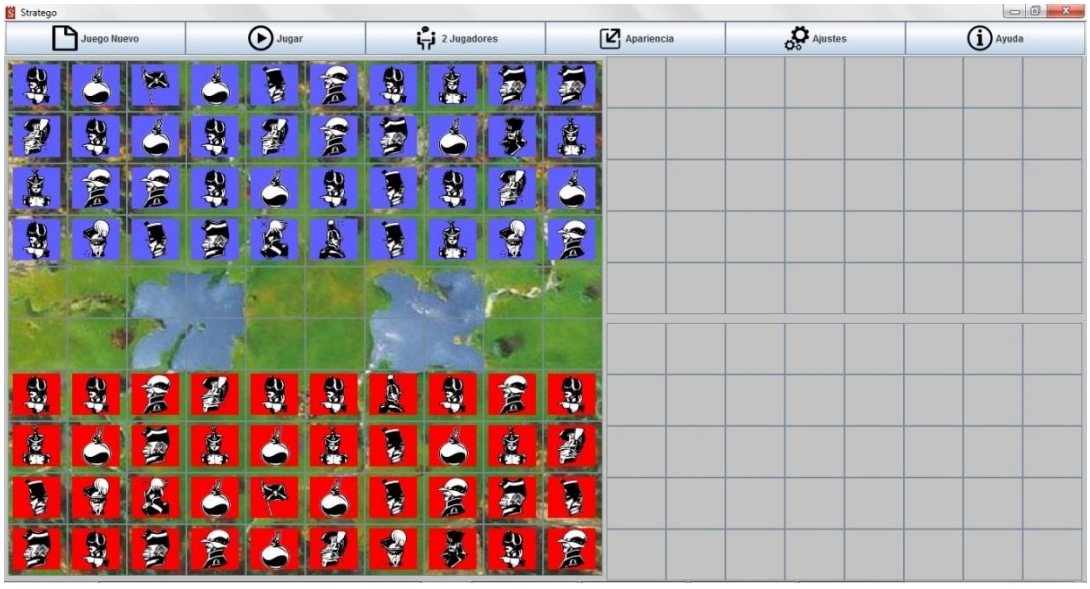
Ayuda: Tras pulsar el botón de **Ayuda** se abre una ventana con una breve explicación del juego, con sus reglas, piezas, movimientos y ataques.



Cargar Estrategia: Tras pulsa el botón de **Ajustes** solo hay que pulsar *Cargar* y se abre una ventana como esta donde tienes que pulsar *Abrir* y buscar la carpeta donde tienes las estrategias guardadas y pulsar Guardar, tiene la opción de poder seleccionar una o las que quieras y ya la computadora si tiene más de una decidirá cual a través de una función random una al azar para cada jugador.



Tras tener cargada unas estrategias podremos empezar la partida.

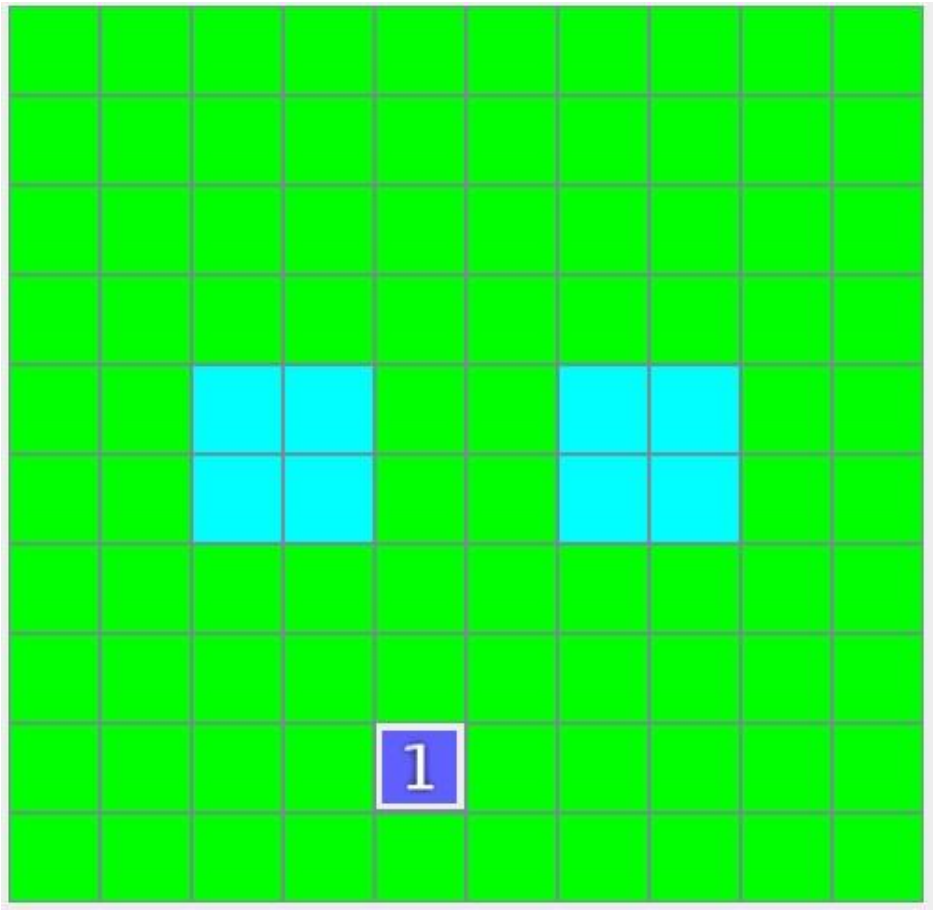


PRUEBAS

En esta sección presentamos diferentes pruebas de la aplicación que hemos realizado para corroborar su funcionamiento.

Caso de prueba 1:

En este caso hemos obtenido el tablero comprobando que el objeto puede ocupar cualquier casilla menos las celestes que son como lagos por donde no se puede pasar.

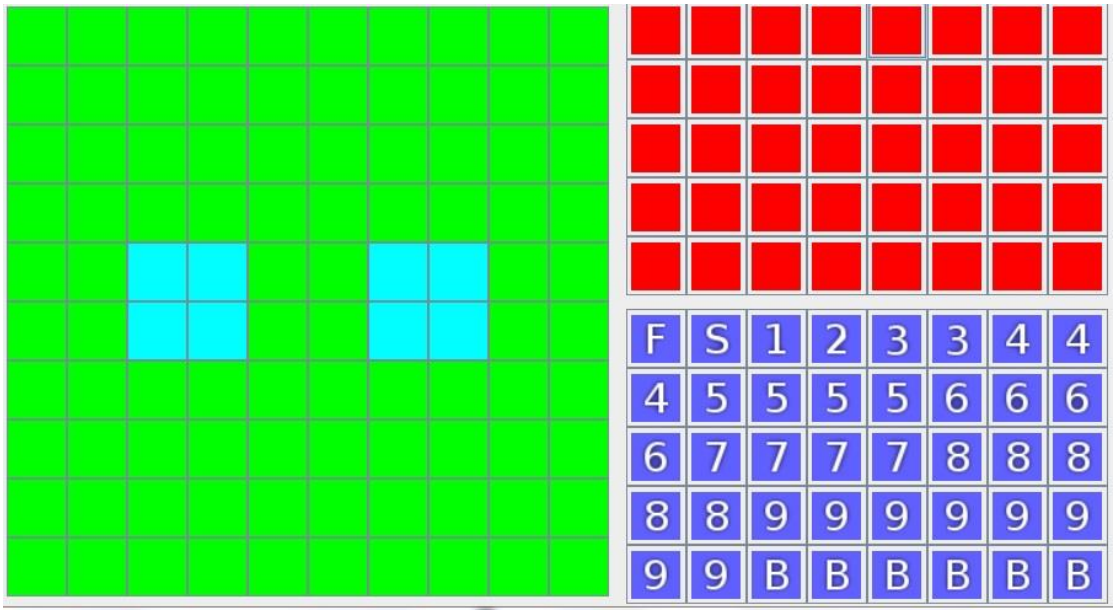


Primer tablero diseñado

Después de comprobar que se cumple todo procedemos a insertar las piezas en el tablero.

Caso de prueba 2:

Colocar a la derecha del tablero un panel donde aparecerán las piezas para ser colocadas en el tablero



Tablero y piezas

Una vez que comprobamos y vemos que podemos colocar todas las piezas en el tablero pasaremos a poder darles movimiento a lo largo del tablero.

Caso de prueba 3:

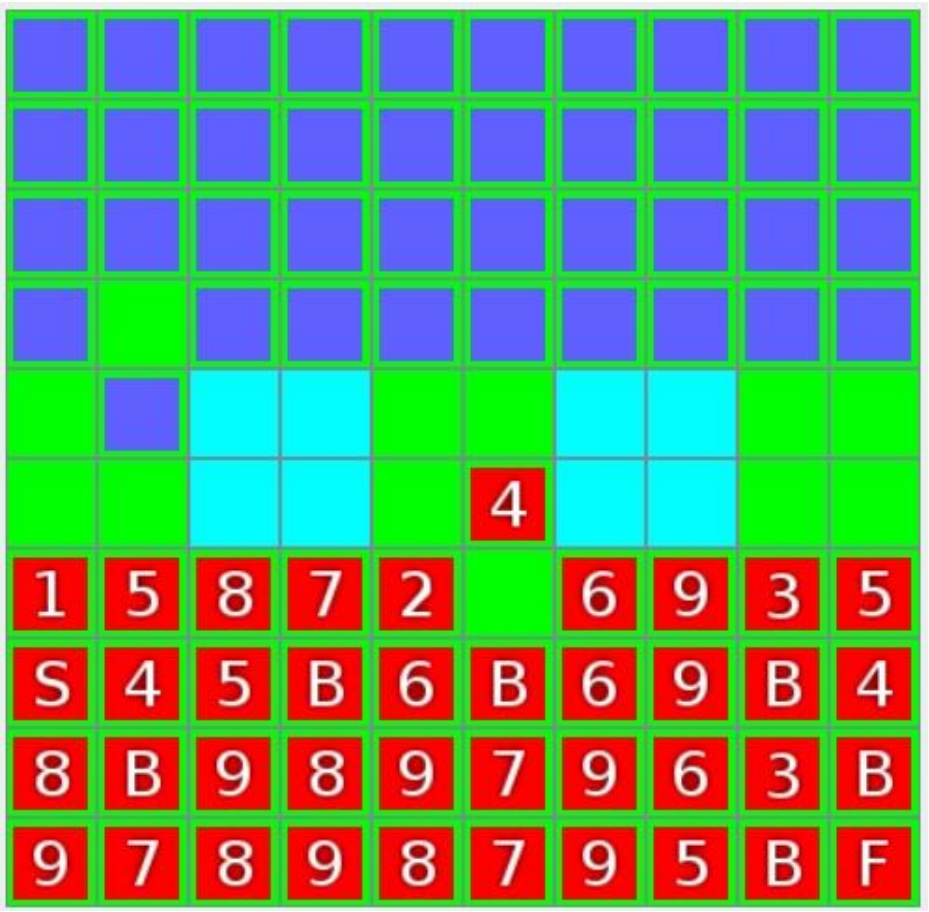
A la hora de dar movimiento a las piezas nos hemos encontrado con más problemas de la cuenta que al final han sido solucionados satisfactoriamente.

Lo primero fue hacer que las piezas se pudieran mover por turnos hacia arriba, abajo, izquierda o derecha, lo cual fue algo sencillo.

Los problemas vinieron a la hora de establecer los turnos que al principio no conseguía el resultado deseado pero fueron corregidos y las piezas se movían respetando sus turnos.

Caso de prueba 4:

Después de conseguir realizar los movimientos básicos de una pieza me encontré con otro problema que se trataba de cuando empezabas el juego no podía empezar con las piezas rojas y si con las azules, algo que me llevo más tiempo de lo esperado, pero que gracias a varias idea que encontré en *"sourceforge"* de varios proyectos similares pude entender un método que realizaba una acción parecida a la buscada y después de hacer varias pruebas conseguí hacer funcionar el juego con normalidad y pudiendo elegir el color que desees para empezar.



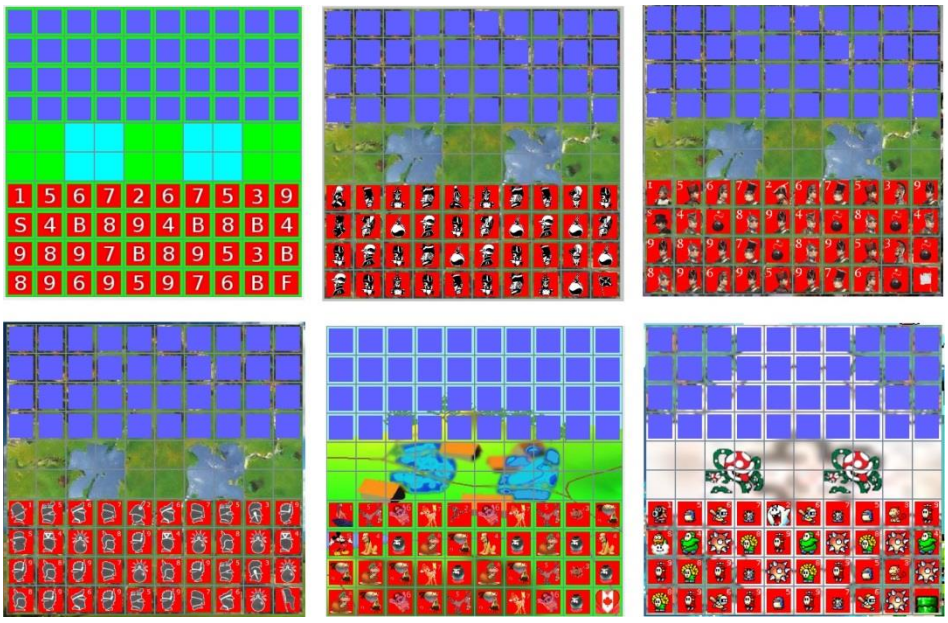
Comienzo de partida

Caso de prueba 5:

Una vez conseguido que el juego funcione con las reglas básicas, procedemos a insertarle una apariencia más acorde con la realidad del juego.

Apariencia final del juego

Después de establecer una imagen para hacer el juego más parecido al original, descubro en sourceforge varias apariencias que son compatibles con el juego y lo hacen más ameno.



Apariencias

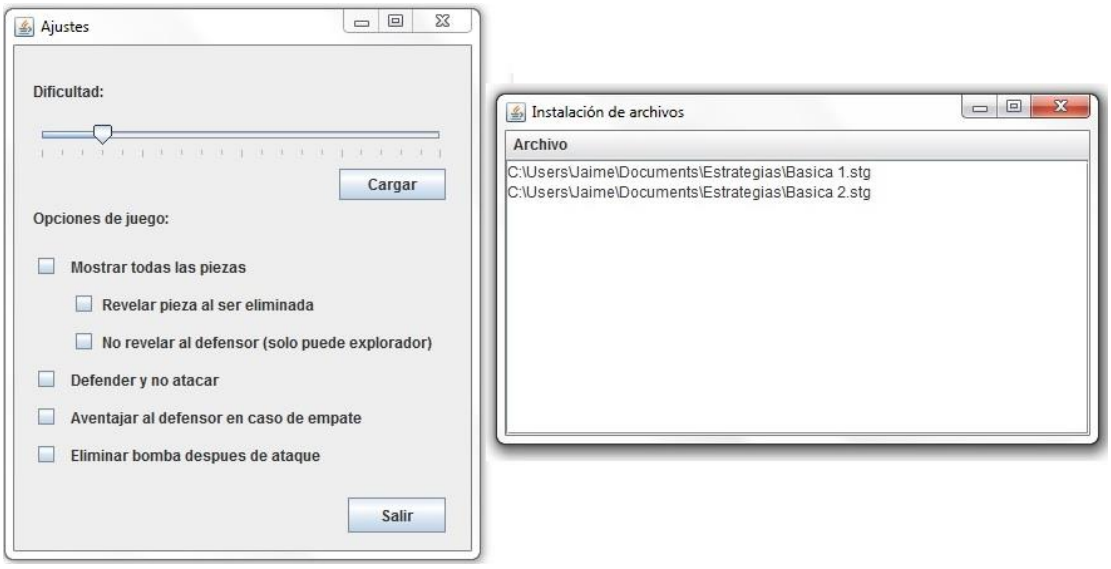
Caso de prueba 6:

Cuando ya tengo todos los movimientos básicos, solo nos queda estudiar la inteligencia artificial de la computadora para que haga movimientos con sentido y poder jugar una partida interesante.

La inteligencia artificial fue desarrollada a través de varias clases que están en el paquete de *"Juegos"* con ayuda de varios proyectos encontrados en *"sourceforge"* conseguimos desarrollar un algoritmo que recorra todas las opciones posibles de movimientos con sentido.

Caso de prueba 7:

Después se desarrollaron los ajustes donde puedes elegir varias opciones opcionales, la dificultad del juego y cargar varias estrategias de inicio para ambos jugadores o solo la computadora y colocar tus piezas como desees.



Ajustes y Estrategias

Caso de prueba 8:

La realización del juego para dos jugadores y como la opción de hacerla en un solo ordenador no era aceptable, ya que entonces se podría hacer trampas viendo donde están colocadas las piezas del rival. Nos decidimos por la opción de establecer un juego conectado en red a través de una conexión Cliente-Servidor abriendo los dos el mismo puerto e introduciendo las IPs de cada uno en el otro PC.

COMPARACIONES CON OTRAS ALTERNATIVAS

A la hora de hacer el proyecto estudie varias alternativas para poder realizar un proyecto que me resultara interesante y además pudiera aprender cosas nuevas en el mundo de la programación y las aplicaciones.

La primera alternativa que se me ocurrió fue desarrollar una aplicación para *"Android"* de un juego de futbol basado en lanzamientos de penaltis o faltas a dianas en movimiento, pero me encontré con demasiados problemas que no pude solucionar, donde el más importante fue desarrollar el movimiento del balón a la hora de lanzarlo contra las dianas, y tras varias semanas sin avanzar opte por cambiar de proyecto.

La segunda alternativa fue otra aplicación *"Android"*, pero esta vez la idea la elegí entre las opciones que ofrecía mi tutor, que era diseñar una aplicación que se basaba en hacer fotografías a los componentes de los alimentos de un supermercado y tras analizar los caracteres, reconocerlos y poder decirte si es bueno para tu consumo. Habiéndote registrado previamente en la aplicación rellenando una serie de campos en los que indicas los alimentos que no tendrías permitido comer. Pero también me encontré con varios problemas a la hora de desarrollar la manera de que el móvil reconociera los caracteres de una fotografía, ya que el móvil del que disponía en ese momento tenía una cámara con mala calidad y no pude desarrollar bien la idea como me hubiera gustado por lo que decidí cambiar de proyecto.

La tercera alternativa fue una idea de un amigo para desarrollar una página web a través de un gestor llamado *"Drupal"* que contiene varios módulos que facilitaban la creación de páginas web. La idea se basaba en una red social privada en la que solo tuvieran acceso a tu información las personas registradas en tu comunidad. Pero después de crear la página web con los módulos proporcionados intente crear un módulo propio que representara todos los módulos descargados, ya que no tenía merito usar los módulos que habían, ya que sino el proyecto estaría basado en arrastrar los módulos que te gusten a la página hasta dejarla como había pensado. Pero la creación de ese modulo me llevo mucho tiempo y muchos problemas por lo que opte por cambiar por última vez de proyecto y hacer uno que me fuera más familiar y sencillo de realizar.

Aunque he perdido mucho tiempo haciendo estos proyectos, he aprendido bastante dentro de cada campo de cada proyecto y tengo como metas personales finalizarlos en cuanto concluya la carrera y adquiera más conocimientos en ambos campos.

CONCLUSIONES

El desarrollo de un proyecto de una aplicación es algo que me interesaba mucho y después de intentar desarrollar varias ideas de aplicaciones con *"Android"* o una página web con *"Drupal"*, teniendo muchos problemas para diseñarlas, ya que eran lenguajes nuevo para mí y me costó avanzar como esperaba.

Al final decidí usar un lenguaje conocido como *"Java"* y hacer una aplicación para PC.

Me decante por el juego Stratego porque es un juego conocido para mí y me gustaba la idea de poder desarrollarlo. Ya que contaba con más conocimientos que los anteriores y más ayuda de compañeros que saben desenvolverse mejor en *"Java"* ya que es el lenguaje que nos han enseñado en la universidad. Y aun así he mejorado bastante en ese lenguaje ya que he aprendido a desenvolverme en situaciones nuevas con resultados satisfactorios.

Cabe destacar que la gran mayoría de lenguajes son muy parecidos, ya que *"Android"* también se puede desarrollar con el eclipse al igual que *"Java"*, pero añadiéndole plugins con todas las librerías de android. Por lo que es más fácil solucionar cualquier problema siempre que se te ocurran soluciones correctas, que no fue mi caso en los proyectos anteriores.

Son muchos los problemas que me han surgido a lo largo de todos los proyectos, en los que en algunos encontré varias maneras de resolverlos eligiendo siempre la manera más adecuada y menos costosa y sin embargo también tuve otros problemas que no les vi la solución que esperaba.

Todos estos problemas surgidos, se puede decir que ayudan a poder decidir por sí mismo las decisiones más correctas, y a investigar cuál es la mejor opción teniendo en cuenta todas las posibilidades.

A lo largo de la carrera, al tratarse de asignaturas de no más de 9 créditos, las implementaciones que nos pedían que realizáramos no eran de tan semejante envergadura. Por tanto, al producirse errores dentro de la tarea por hacer, era más sencillo corregirlos, ya que simplemente por tamaño y número de líneas, era mucho más fácil encontrarlo y solucionarlo.

Para este proyecto, los problemas de un cambio de una variable en una parte del código pueden ocasionar problemas considerables y en muchos casos encontrar ese error entre tantas líneas de código parece una locura. Todo esto enseña a ser más cuidadoso a la hora de realizar cambios en alguno de los archivos pertenecientes al proyecto. También nos hace precisar de un cierto orden, ya que sin un orden concreto sería imposible que fuera legible, incluso por mí mismo, el código fuente de esta aplicación.

Una las cosas más importantes que me ha demostrado este proyecto, es la gran utilidad de los algoritmos utilizados y aprendidos a lo largo de la carrera. Como con simple lógica matemática, se pueden desarrollar soluciones de este calibre. Se puede decir que es una manera muy buena de comprender la utilidad de todas aquellas clases teóricas de algoritmos matemáticos. Además, he aprendido a adaptar algoritmos ya existentes y muy utilizados, para que funcione en un caso para el que inicialmente no está preparado e incluso conseguir mejorarlos y darle una utilidad para el que no estaba pensado.

Después de crear un juego de mesa, me he dado cuenta que todos ellos tienen la misma lógica, y que sabiendo cómo funciona internamente uno, es muy fácil construir otro con características similares. Esto me da pie en un futuro, a seguir avanzando y poder desarrollar más y mejores aplicaciones de este tipo o de un tipo diferente.

En definitiva, este proyecto ha entrado dentro de las expectativas que tenía pensado para finalizar mi carrera y demostrarme a mí mismo que soy capaz de realizar una aplicación de semejante complejidad. Además, no ha sido un proyecto aburrido ni mucho menos, y he aprendido gran diversidad de valores dentro de la programación y lógica matemática.

DESARROLLOS FUTUROS

En un proyecto de este tipo, es imposible decidir cuando está acabado de forma que no se pueda realizar una mejora. Uno de los objetivo de este proyecto era obtener una inteligencia artificial que sea coherente y a la vez veloz en el cálculo.

Este punto medio entre velocidad e inteligencia es la gran pelea que se tiene siempre con este tipo de juegos.

Al seguir investigando, se pueden descubrir diferentes tipos de estrategia que no consuman los recursos que puede consumir un algoritmo como el *Minimax*, según se aumente su nivel de profundidad y se complique su heurística.

Expertos en juegos de mesa, conocen movimientos iniciales y finales que no necesitan cálculos con algoritmos sofisticados. Se tratan de bases de datos que contienen los movimientos posibles mejores para distintas jugadas, por lo que esos cálculos se podrían obviar y acelerar el proceso de forma que la máquina fuera igual o más inteligente aún.

El uso de otro algoritmo, o la combinación de varios, también podría llevar a resultados mejorados. Todo se trataría de realizar un estudio más amplio poder contar con la opinión experta de jugadores que conozcan las diferentes estrategias para poder "enseñárselas" a la computadora.

Además de mejorar la velocidad e inteligencia, se podría plantear la inclusión de algún que otro botón que permita una mejora en el estudio de los movimientos, como un botón de marcha atrás, demo o algo similar. En este proyecto se ha intentado diseñar el botón de Demo pero sin resultados satisfactorios dejando el código comentado de tal manera que en un futuro se pueda conseguir finalizar.

La inclusión de un tiempo de reloj sería una mejora interesante ya que esto obligaría al jugador humano a no tener todo el tiempo del mundo para realizar su movimiento, lo que le daría un poco más de realismo al juego.

Por otro lado, y pensando como producto final, después de realizarle las mejoras comentadas anteriormente, esta versión de juego de estudio, podría ser convertida en una versión de juego comercial eliminando todos esos factores que condicionan los cálculos. Si nos quedamos con el nivel de profundidad más válido, y la heurística que se considere más oportuna, añadiéndole todas esas mejoras posibles, se podría obtener una versión que el usuario no tendría porque preocuparse de nada, simplemente de poder completar la partida satisfactoriamente. Ésta es otra visión de posible aplicación futura de este proyecto.

Son muchas las aplicaciones y mejoras que se le puede dar a este proyecto. Aquí se exponen las que a día de hoy caben ser realizadas en un futuro. Pero conforme se va avanzando, se va viendo un campo mucho más amplio de mejoras y cambios. Todo depende del tiempo que se emplee y la dedicación utilizada.

BIBLIOGRAFÍA

La bibliografía utilizada ha sido puramente obtenida de la web, ya que, aparte de la necesitada para entender el funcionamiento y la historia del juego a desarrollar, la información sobre programación de cualquier tipo abunda en Internet.

Historia y reglas del juego:

http://es.wikipedia.org/wiki/Stratego

Ideas de otros proyectos similares realizados con mi tutor José Ramón Portillo en el departamento de matemáticas:

https://forja.rediris.es/users/josera/

La creación del tablero:

https://code.google.com/p/gamepoo/source/browse/trunk/src/game/model/Tablero.java?r=29

Para el estudio de nuevas librerías de JAVA:

https://docs.oracle.com/javase/6/docs/api/

http://jackb10g.blogspot.com.es/2009/05/librerias-swing-y-awt.html

http://www.htmlpoint.com/guidajava/java_14.htm

http://es.slideshare.net/aoteroe/libreras-de-java

http://es.slideshare.net/lauriz19cour/java-gui-la-librera-swing-7932755

Para obtener office gratuito, obtuvimos la licencia en:

http://msdn.microsoft.com/en-us/office/office365/howto/setup-development-environment

Para el desarrollo en red para dos jugadores usaremos los sockets de Java siguiendo la siguiente información para crear la conexión servidor y cliente:

https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

http://codigoprogramacion.com/cursos/java/103-sockets-en-java-con-cliente-y-servidor.html#.VHyabDGG91Z

http://nereida.deioc.ull.es/~cleon/doctorado/doc06/doc06/html/node9.html

http://todojava.awardspace.com/ejemplos-java.html?desc=ClienteServidorSockets

http://sourceforge.net/projects/x-mus/files/

https://code.google.com/p/stratego-game-server/source/browse/trunk/src/main/java/dnl/games/stratego/server/StrategoGame.java?r=3

Para el resto del juego obtuvimos las ideas en varios proyectos ya creados de los que obtuvimos muchas ideas:

http://sourceforge.net/p/stratego/code/HEAD/tree/

http://sourceforge.net/p/java-stratego/code/HEAD/tree/

http://sourceforge.net/p/estratego/code/HEAD/tree/

Con todo esto y muchas horas de estudio ha sido posible la finalización de este proyecto.