ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

Grado de Ingeniería del Software

Algoritmo MiniMax para juegos de cartas de más de dos jugadores

Realizado por:

José Antonio Sosa Cifuentes

Dirigido por:

José Ramón Portillo Fernández

Departamento

Departamento de Matemática Aplicada I

Sevilla, 19 de junio de 2020

Contenido

Figuras

Cuadros

Resumen

El presente documento representa el trabajo de ocho meses dedicado a la implementación del juego de cartas ‘Anarquía’ usando el lenguaje de programación ‘Python’. Además, se ha realizado una inteligencia artificial capaz de jugar a este juego.

Empezaremos detallando los temas principales de este proyecto, que consistirán en la descripción del entorno de desarrollo usado, junto con la explicación tanto del juego de mesa a implementar, ‘Anarquía’, como del algoritmo para la inteligencia artificial usado, ‘MiniMax’.

Una vez explicado esto, nos centraremos en detallar los requisitos para realizar este proyecto, junto con el análisis principal de la aplicación, tanto de el por qué se ha usado el lenguaje ‘Python’, como del diseño de todos los aspectos que conforman el conjunto del programa. También se detallará una guía de usuario, además de la implementación necesaria para ejecutar el programa.

Por último, hay que comentar que en los apartados finales se realizará una comparativa con otros lenguajes de programación, mostrando los puntos fuertes y flojos de estas alternativas, junto con la razón de por qué se acabó usando ‘Python’ sobre otras herramientas. Para finalizar, se hará un análisis temporal de este proyecto, junto con las conclusiones y posibles trabajos futuros que pudieran mejorar este proyecto.

Capítulo 1

Definición de objetivos

Este Trabajo de Fin de Grado consiste en la implementación del juego de cartas ‘Anarquía’, usando el lenguaje de programación Python. Para conseguir esto, necesitaremos:

- Estudiar las distintas posibilidades que tenemos a la hora de representar un juego con un ordenador.
- Aprender sobre la representación del problema en cuestión como un arbol de decisiones.
- Conocer los procedimientos para la búsqueda de la mejor resolución que podamos encontrar en dicho arbol de decisiones.
- Investigar sobre la forma de implementación que necesitará nuestro problema.
- Profundizar en el conocimiento del algoritmo MiniMax, y su posible uso en juegos con varios jugadores.
- Implementar en Python el algoritmo MiniMax.
- Aprender sobre la forma en la que realizar algoritmos de toma de decisiones basados en MiniMax para más de dos jugadores.
- Adaptar dicho algoritmo para que funcione con nuestro juego de cartas.
- Investigar sobre métodos para realizar interfaces gráficas en Python.
- Realizar una interfaz gráfica usando el método más adecuado para nuestra aplicación.
- Buscar y encontrar las herramientas necesarias para llegar a la solución de los problemas que se identifiquen en este Trabajo de Fin de Grado.
- Aplicar las actitudes, conocimientos y habilidades adquiridos en estos estudios universitarios de Ingeniería del Software, así como la creatividad necesaria para poder abarcar el reto que supone este Trabajo de Fin de Grado.

Capítulo 2

Antecedentes

Como es lógico, nuestro proyecto tendrá tres pilares principales sobre los que se sujetará. Uno es el lenguaje de programación sobre el que se desarrollará, en este caso Python, por otro lado, el juego de cartas en sí, Anarquía, y por último, el algoritmo de búsqueda, MiniMax.

2.1. Lenguaje de programación Python

Para realizar nuestro proyecto, se decide que se va a usar el lenguaje Python, usando el entorno de desarrollo integrado (o IDE para abreviar) 'PyCharm'. A parte de esto, para la parte de interfaz de usuario, decidimos hacerla con la librería de Python 'Kivy'.

2.1.1. Python

Python **[12]** es un lenguaje de propósito general, interpretado y de alto nivel, que permite al programador a usar distintos estilos de programación para crear programas con el nivel de complejidad que se desee. La forma de codificación podría considerarse casi como si fuera un lenguaje humano.

El desarrollo inicial de Python comenzó con Guido van Rossum a finales de la década de 1980 en el centro para las Matemáticas y la Informática en los Países Bajos. Esta persona ha sido el principal autor de este lenguaje de programación, y ha sido hasta el 12 de Julio de 2018 la persona encargada en decidir la dirección en la que irá el desarrollo del lenguaje. En este momento, deja a la comunidad de Python la forma en la que decidirán los nuevos parches y las nuevas funcionalidades que se tomarán en Python.

Debido a que Python es un lenguaje paradigmático, **[13]** permite que los programadores que usen este lenguaje puedan alcanzar sus objetivos usando distintos estilos de programación:

orientado a objetos, imperativo, funcional o reflectivo. Gracias a esto, Python puede usarse para el desarrollo web, desarrollo de videojuegos, programación numérica y más.

Además de esto, Python posee atributos que lo convierten en una opción de desarrollo más rápida que la ofrecida por otros lenguajes de programación:

- Python es un lenguaje interpretado que excluye la necesidad de compilar código antes de ser ejecutado, ya que ese proceso se realiza mientras se ejecuta el código. Ya que Python es un lenguaje de programación de alto nivel, abstrae muchos detalles sofisticados del lenguaje de programación. Python se centra tanto en su abstracción, que el código escrito en este lenguaje puede ser comprendido fácilmente hasta por los programadores más novatos.
- El código en Python tiende a ser más corto que en otros lenguajes. Aunque el desarrollo en este lenguaje sea rápido, el tiempo de ejecución es un poco más lento comparado con otros lenguajes. Sin embargo, estas diferencias de velocidad, con las velocidades de computación actuales casi solo se ven reflejadas en benchmarks, no en operaciones del mundo real.

2.1.2. PyCharm

PyCharm es un IDE enfocado a la creación de código en el lenguaje Python, **[11]** desarrollado por la empresa checa JetBrain que, entre sus funcionalidades, nos podemos encontrar características como la compleción de código, soporte a el trabajo de frameworks web como Django o Flask, herramientas científicas, como la posibilidad de integrar IPython Notebook o la existencia de consolas interactivas, o la posibilidad de poder trabajar con otras tecnologías a parte de Python, como pueden ser JavaScript o HTML.

2.1.3. Kivy

Kivy es una librería open-source de Python dedicada a la creación de interfaces de usuario multiplataforma.

Se puede usar para crear aplicaciones con interfaz de usuario que se ejecuten tanto en Windows, Linux, OS X, Android, iOS o Raspberry Pi, sin necesidad de modificar el código.

Además de esto, podemos usar la mayoría de los protocolos y dispositivos, como WM_Touch, WM_Pen, Mac OS X Trackpad and Magic Mouse, Mtdev, Linux Kernel HID, o TUIO.

2.2. Anarquía

La anarquía [2] como concepto aparece en la Grecia clásica, con un significado en general negativo, pues la anarquía como sistema político quiere decir que no hay un gobierno que cuide de sus ciudadanos, lo que llevaría al caos.

No obstante, hasta la Ilustración del siglo XVIII no se volvió a retomar tal concepto, pero esta vez con aires esperanzadores, pensando que, sin gobierno, las personas son más libres.

La persona que más influenció en esta línea fue **Mijaíl Bakunin.** Este filósofo ruso anarquista fue el que sentó las bases teóricas del movimiento anarquista actual.

Para Bakunin, Las personas deberían vivir en pequeños grupos, en los que no se producirían ninguna diferenciación social entre las personas que vivan en ese grupo, dando como principio ético fundamental el valor del trabajo.

Con esto, las comunas podrían **autoorganizarse**, **autogestionarse** y preservar la armonía entre todos los miembros de la comuna.

Como vemos, la anarquía consiste en la igualdad de todos, sin ventajas para nadie y sin sentimiento de inferioridad de una persona frente a otra. Estas características las comparte el juego de cartas que vamos a usar para nuestro Trabajo de Fin de Grado, el cual comparte nombre e ideologías con el sistema político, y que explicaremos a continuación.

El juego de cartas **Anarquía** [7] puede ser descrito como una parodia de los juegos de cartas basados en turnos, dado que aquí, todos juegan sus cartas al mismo tiempo, en vez de usar el método tedioso de usar turnos para cada uno de los jugadores, ya que “es antidemocrático, dando un privilegio a quien vaya primero que no se ha merecido, siendo apto solo al entretenimiento decadente de la sociedad elitista jerarquizada”.

Este juego apareció por primera vez en el libro **Original Card Games** en 1977.

2.2.1. Componentes del juego

Para jugar a este juego necesitaremos:

- Cinco jugadores.
- Una baraja de cincuenta y dos cartas de cuatro palos, con una clasificación A-K-Q-J-10-9-8-7-6-5-4-3-2 para cada palo.

2.2.2. Elección de estrategia de juego

Al comenzar el juego, a cada jugador se le reparten diez cartas, y en base a esas cartas, cada jugador tiene que decir qué estrategia elegirá para contar su puntuación. Estas estrategias serán:

- Picas: 2 puntos por carta que el jugador se lleve de picas.
- Corazones: 2 puntos por carta que el jugador se lleve de corazones.
- Rombos: 2 puntos por carta que el jugador se lleve de rombos.
- Tréboles: 2 puntos por carta que el jugador se lleve de tréboles.
- Sin palo: 1 punto por cada carta que el jugador se lleve sin importar el palo de la carta.
- Miseria: 1 punto por cada carta que el jugador se lleve sin importar el palo y que sea menor de 10.

2.2.3. Antes de comenzar a jugar

Si contamos el número de cartas repartidas, nos damos cuenta de que hemos repartido cincuenta cartas, pero quedan todavía otras dos que no se han repartido. Esto es así porque ahora en la primera ronda se juega la primera carta no repartida antes de que nadie juegue nada, y esta se la llevará el jugador que eche la carta de valor más alto de ese palo. Una vez hecho esto, se hecha la siguiente carta, y por el mismo procedimiento, la segunda carta se la llevará el jugador con la carta de mayor valor de ese palo.

Una vez hecho esto, todos los jugadores se llevarán las cartas que han jugado a su mano, y las cartas ganadas a su baraja de cartas ganadas.

2.2.4. Comienza el juego

Tras repartir las últimas dos cartas, se procederá a iniciar el juego.

En cada turno, cada jugador juega "simultáneamente". Esto quiere decir que cada jugador elige una carta y la pone en la mesa boca abajo. Una vez todos hayan escogido su carta, todos al mismo tiempo desvelan su carta. Entonces, las cartas ganadas serán las siguientes:

- Si dos o más jugadores juegan al mismo palo, el jugador con la carta de mayor valor se llevará tanto su carta como la de los demás que jugaron el mismo palo.
- Si un jugador juega a un palo que los demás no han jugado, el jugador se llevará solo su carta.

Por lo tanto, una jugada ganada será aquella en la que el jugador se llevará a su baraja de cartas ganadas de una a cinco cartas, y una ronda perdida será aquella en la que no se lleve nada.

Un ejemplo de jugada podría ser el siguiente:

Supongamos que somos el Jugador 1 y al jugar nuestra carta nos encontramos con lo siguiente:

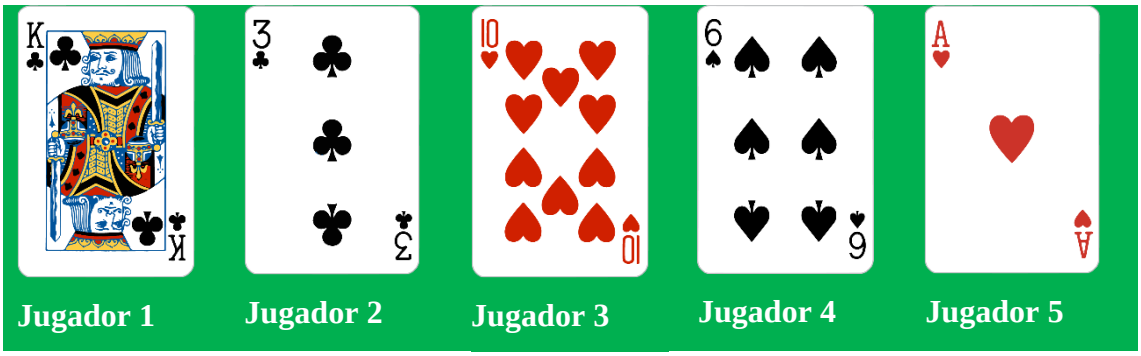


FIGURA 1 EJEMPLO DE JUGADA EN EL JUEGO 'ANARQUÍA'

Como podemos ver, hemos jugado el rey de tréboles, el jugador 2 el 3 de tréboles, el jugador 3 el 10 de corazones, el jugador 4 el 6 de picas y el jugador 5 ha jugado el as de corazones.

Aquí, nosotros nos llevaremos el rey de tréboles y el tres de tréboles, el jugador 5 se llevará el as de corazones y el 10 de corazones, y el jugador 4, al ser solo él el que jugó una carta de picas, se llevará el 6 de picas.

2.2.5. Puntuación

Una vez todos los jugadores se quedan sin cartas que jugar, comienza la cuenta de los puntos que hemos conseguido.

Supongamos que hemos jugado con una estrategia de corazones, y acabamos con una baraja de cartas ganadas como esta:

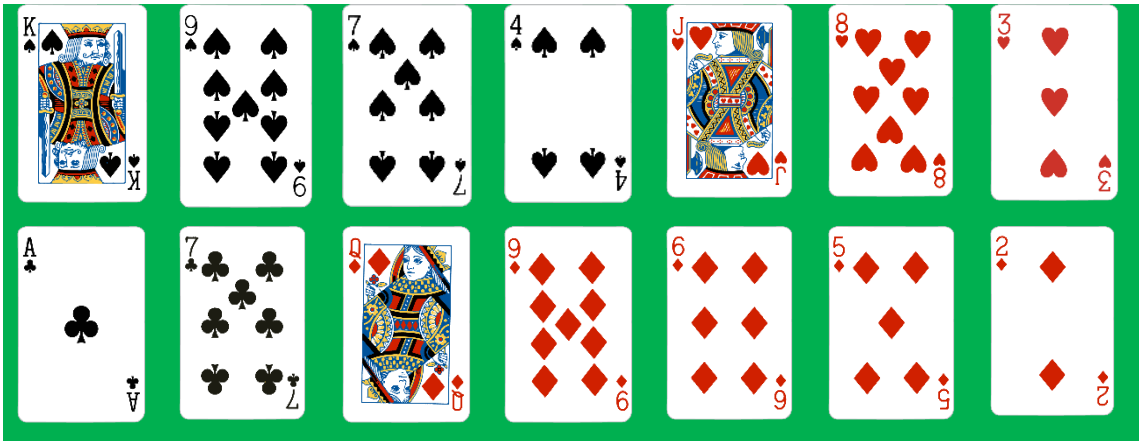


FIGURA 2 EJEMPLO DE CARTAS GANADAS EN EL JUEGO 'ANARQUÍA'

Como podemos apreciar, si puntuamos 2 puntos por cada carta de corazones, acabaríamos con 6 puntos, pero si hubiéramos jugado a diamantes nos hubiéramos llevado 10 puntos y si hubiéramos jugado sin palo, nos hubiéramos llevado 14 puntos.

Todas las imágenes de cartas que se han usado y se va a usar son de uso público y se pueden encontrar en **[6].**

2.3. MiniMax

MiniMax [7] es un teorema ideado por **John Von Neumann** que parte de la premisa de que "un juego es una situación conflictiva donde uno tendrá que tomar una decisión sabiendo que los demás también lo harán, y en la que el resultado de este conflicto vendrá dado por las decisiones tomadas". Además, también se afirma que siempre habrá una forma racional de acción siempre que los oponentes tengan objetivos totalmente opuestos.

La demostración a esta afirmación se denomina **teoría MiniMax,** y consiste en la búsqueda de la jugada idónea teniendo en cuenta que el contrincante intentará buscar la jugada que más nos entorpezca.

En el algoritmo **MiniMax,** la búsqueda se hará por medio de un arbol de decisiones, que estará definido por: [8]

- **Estado inicial**: El lugar en el que nos encontramos en el juego para tomar la decisión
- **Operadores**: Jugadas posibles que se pueden realizar. En nuestro juego de cartas, los operadores que tenemos serían cada una de las cartas que tenemos en nuestra mano.
- **Condición terminal**: Determina cuándo se acaba el juego, que en nuestro caso será cuando a todos los jugadores se les acaben las cartas de su mano.
- **Función de utilidad**: Cuando se llega a una condición terminal, se le da un valor numérico, dependiendo de si se ha ganado, perdido o empatado en el juego. Lo normal es que este valor sea -1 si el jugador ha perdido, 1 si ha ganado y 0 si ha empatado. Sin embargo, debido a la forma en la que funciona nuestro algoritmo, el resultado de esta función será el número de puntos del jugador que decide la carta a jugar.

Conocido esto, el funcionamiento teórico de **MiniMax** será el siguiente:

- Primero se genera un arbol de decisión de las posibles decisiones que se puedan tomar.
- A cada uno de los valores terminales se les da un valor dada por su función de utilidad.
- A partir del nodo terminal se irá comunicando recursivamente al nodo anterior cuál es el nodo óptimo que se puede alcanzar si se toma un camino determinado, alternando en cada momento dependiendo del jugador que tenga que jugar si la función busca maximizar o minimizar la puntuación, intentando así, maximizar la puntuación de nuestro jugador, y minimizando la del adversario.
- Al final se devolverá la jugada que se deberá jugar para conseguir la máxima puntuación posible.

Hay que aclarar que este es el funcionamiento teórico del algoritmo MiniMax. En la realidad, hay pocos juegos en los que se pueda uno permitir llegar hasta el final del arbol de decisiones. Nuestro juego de cartas es un ejemplo perfecto. Al final, la forma en la que se adapta este algoritmo la veremos más adelante, en el diseño de la aplicación,, pero por ahora, digamos que usaremos una mezcla de las alternativas que expondremos a continuación. Para darnos cuenta de cómo de imposible es de realizar los cálculos teóricos de MiniMax en la actualidad, vamos a ver cómo se comporta MiniMax teórico con nuestro juego de cartas:

MiniMax es un proceso que consumirá demasiado tiempo dependiendo del tipo de juego que se esté jugando. En nuestro caso, por ejemplo, tenemos unos 5 jugadores a los que se les reparten 10 cartas a cada uno. La toma de decisiones en base a este algoritmo, tal como está propuesto, tomaría demasiado tiempo para un ordenador actual por la cantidad de ramas a calcular, ya que nos encontramos que por cada juego (teniendo en cuenta de que en cada juego se reparten las cartas de una manera diferente y que, por lo tanto, cada juego será distinto) podemos encontrarnos unos 220.825 finales distintos en el estado inicial. Por lo tanto, tenemos que averiguar una forma para optimizar este sistema para que una decisión no nos lleve demasiado tiempo.

Para evitar que el ordenador tarde demasiado tiempo en calcular la decisión óptima, podemos optar por varias alternativas que, aunque puede ser que no alcancen a llegar a la opción óptima, podemos llegar a alcanzar una cierta inteligencia en la elección de la jugada a realizar:

2.3.1. Limitar la profundidad de búsqueda

Esta opción, como indica el nombre, consistirá en limitar la profundidad a la que el algoritmo buscará.

Si desde el nodo inicial en el que estamos necesitamos para llegar al final pasar por unos 20 nodos, y tenemos un límite de búsqueda de 5 nodos, no llegaremos al final, si nos quedaremos a medias en el árbol de decisión.

Esto hace que tengamos que cambiar la forma en la que calculamos la función de utilidad, ya que, si nos encontramos en mitad del juego, el resultado de todas las ramas será el de empate, cosa que no tendrá sentido a la hora de elegir un camino u otro en el arbol de

decisiones. Para solucionarlo, tendremos que basar el cálculo de esta función en una heurística que permita encontrar la mejor opción en base a futuro. Un ejemplo de heurística para un juego de mesa conocido podría ser en el tres en raya, donde una forma de encontrar la mejor opción en base a futuro podría ser el número de líneas de tres que podríamos formar menos el número de líneas que podría formar el contrincante.

Como podemos suponer, con esto no sabremos cuál es la opción óptima para llegar al final del juego con la máxima puntuación posible, pero lo que si nos permite es Averiguar la mejor jugada para tener la máxima puntuación en el futuro próximo.

2.3.2. Poda Alfa-Beta

Por otro lado, la poda Alfa-Beta **[9]** se basa en evitar ramas que no nos garanticen una buena puntuación.

Para ello, definimos dos parámetros, que serán alfa y beta. Alfa representará el valor óptimo para el camino de maximización, mientras que beta será el valor más favorable para el camino de minimización.

Estos parámetros los inicializaremos de forma que alfa sea el menor valor posible, y beta sea el mayor valor posible.

A partir de aquí, el procedimiento será el siguiente:

- Generamos el arbol de decisiones, que será hasta el final si no implementamos límite de profundidad, o hasta el límite de profundidad o el final si implementamos dicho límite de profundidad.
- Recorremos el arbol generado hasta el final.
- Vamos actualizando alfa y beta si vemos que en un nodo la puntuación de alfa es menor que la del nodo si estamos en un nodo maximizador y beta si el nodo es menor que éste y estamos en un nodo minimizador.
- Si en algún punto del algoritmo vemos que beta es menor o igual que alfa, podamos y procedemos a no buscar más en esa rama.

Para entender mejor el método de poda, vamos a ver un ejemplo **[10]** (La aplicación usada para realizar los árboles de decisión los pueden encontrar en **[3]**):

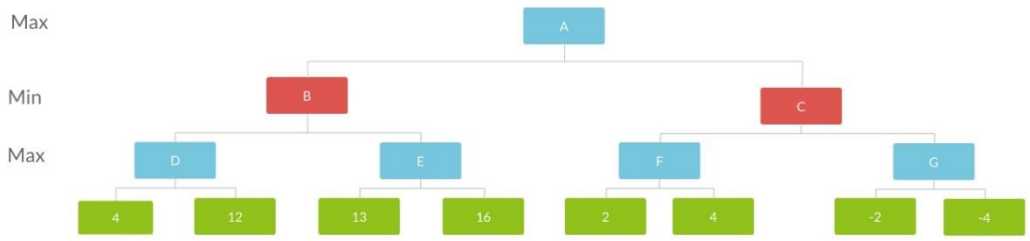


FIGURA 3 EJEMPLO DE ARBOL DE DECISIONES PARA EXPLICAR LA PODA ALFA-BETA. INICIO

- Como podemos ver en el árbol de decisiones, empezamos en A, donde alfa vale –**INFINITO** y beta vale **INFINITO**. Estos valores se pasan a los nodos inferiores. En A el maximizador tendrá que elegir el máximo entre B y C. Por lo tanto, A llama primero a B.
- En B, el minimizador debe elegir el mínimo entre D y E, por lo que primero elegirá a D.
- En D, miramos a nuestros hijos, que resulta que son finales. Miramos primero al de la izquierda, que nos da un valor de 4. Por lo tanto, ahora el valor de alfa en D es 4.
- Para ver si merece la pena mirar el nodo de la derecha, vemos si beta <= alfa. Esto es falso, por lo que seguimos la búsqueda.
- D ahora mirará al hijo de la derecha, que devolverá 12. En D, alfa será el máximo entre el valor anterior, 4, y el del nuevo hijo, 12. Por lo tanto, ahora el valor del nodo D será 12.
- D le devolverá un valor de 12 a B. En B, beta será igual al valor mínimo entre el valor anterior, INFINITO, y el nuevo, 12. Por lo tanto, nos quedamos con 12. B entonces buscará en E a ver si podemos conseguir un valor menor a 12.
- En E, alfa será -INFINITO y beta será 5. Ahora en E buscamos a su hijo de la izquierda que da un resultado de 13. Entonces, alfa será el máximo entre el valor anterior, -INFINITO, y el nuevo, que es 13, por lo que nos quedarnos con 13. Para ver si nos interesa continuar con la búsqueda, comprobamos si beta es menor que alfa. Como beta es 5 y alfa es 13, beta <= alfa, por lo que dejamos de buscar, y en E devolvemos 6 a B, sin importar lo que valga la rama de la derecha.
- E devuelve un valor de 6 a B. En B, vemos que beta será el valor mínimo entre el valor que tenía antes, 12, o el nuevo, 13. Por lo tanto, en B el valor de beta con el que nos quedaremos será 12, y el valor de B será igualmente 12.

Entonces, por ahora nuestro árbol quedaría así:

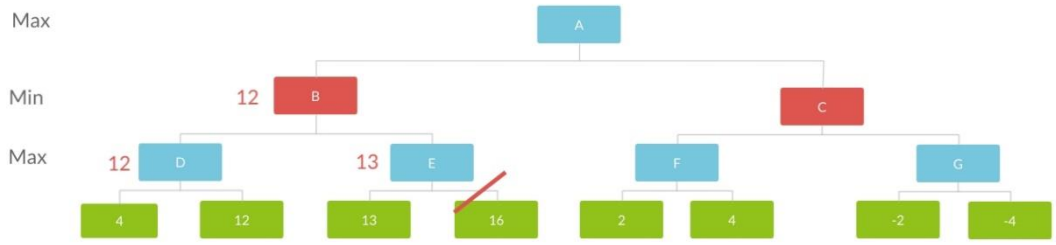


FIGURA 4 EJEMPLO DE ARBOL DE DECISIONES PARA EXPLICAR LA PODA ALFA-BETA. EXPLORADA PRIMERA RAMA

- B devolverá 12 a A. En A, alfa será el máximo valor entre el que tenía anteriormente, -INFINITO, y el que le viene de B, que es 12. Por lo tanto, alfa sería igual a 12. Una vez hecho esto, buscamos en C a ver si podemos conseguir un valor superior a 12.
- En C, alfa es 12 y beta INFINITO. C busca en F.
- En F, alfa es 12 y beta INFINITO. En F buscamos el valor de su hijo izquierdo, que nos da un valor de 2. Como alfa es el máximo valor entre el valor que teníamos, 12, y el nuevo, 2, nos quedaremos con 12.
- Ahora, en F buscamos el valor del hijo de la derecha, que es 4. Como 4 sigue siendo menor que 12, nos quedamos con 12.
- F devolverá a C un valor de 4. En C, beta es el valor mínimo entre el valor actual, INFINITO, y el nuevo, 4, por lo que nos quedaremos con 4. Aquí, como beta es menor que alfa, ya que alfa es 12 y beta es 4, cortamos la búsqueda y no miramos en la rama de G.
- La lógica de este comportamiento es la siguiente: Al minimizador en C se le garantizó un valor de 4 o menos. Sin embargo, El maximizador ya tenía asegurado un valor de 12 si eligiera B. Entonces, ¿para qué elegiría el maximizador C, teniendo una mejor opción en B? Con esto nos ahorramos unos cuantos ciclos de cómputo, ya que no tendremos que calcular el resultado de un subárbol que no nos daría ningún beneficio.
- Entonces, C devolverá un valor de 4 a A. Por lo tanto, el mejor valor para A será el máximo valor entre la anterior alfa, 12, y el que nos ha devuelto el cálculo nuevo, que es 4. Por lo tanto, nos quedamos con 12.
- Así, llegamos a la conclusión de que el valor óptimo que el maximizador puede conseguir es 12.

Para finalizar, nuestro árbol de decisiones quedaría así:

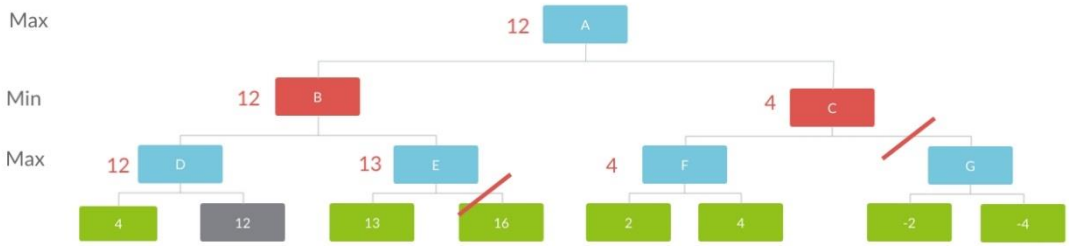


FIGURA 5 EJEMPLO DE ARBOL DE DECISIONES PARA EXPLICAR LA PODA ALFA-BETA. FINAL

Capítulo 3

Requisitos de la aplicación

3.1. Requisitos funcionales

La aplicación por desarrollar debe cumplir ciertos requisitos funcionales. Estos serán los siguientes:

- Funcionar en sistemas operativos usados actualmente.

  Esta parte es fácil de conseguir, ya que, gracias al hecho de que Python es un lenguaje de código libre que sigue Python, lo único que hace falta para que el código funcione en cualquier ordenador es instalar Python y las librerías necesarias para que funcione el código.

- La máquina debe ser tanto escenario como árbitro del juego de mesa.

  Para poder realizar una inteligencia artificial que sea capaz de jugar al juego de cartas que vamos a usar para nuestro proyecto, primero necesitaremos definir lo necesario para que el juego pueda desarrollarse. Cosas como la mano de cartas que puede usar un jugador, la cuenta de puntos, o cosas como cuándo se acaba la partida tienen que ser definidos. Con esas reglas creadas en nuestro proyecto, la máquina que ejecute este código será el escenario de nuestro juego de cartas, a la vez que será el que decidirá cuándo un jugador podrá jugar y qué cartas serán las que pueda usar.

- Permitir partidas contra la propia máquina.

  Además de que un ordenador pueda jugar a este juego de cartas por sí mismo, la idea es que un jugador de carne y hueso pueda jugar a este juego sin la necesidad de buscarse a otros cuatro amigos que quieran jugar a este juego

3.2. Requisitos no funcionales

Además de lo anteriormente mencionado, nuestro proyecto deberá cumplir los siguientes requisitos:

- La aplicación se elaborará en el lenguaje Python.

  Realmente, este requisito es trivial, ya que podría haberse hecho en cualquier otro lenguaje de programación. Sin embargo, las razones por las que se ha elegido hacer este Trabajo de Fin de Grado se expondrán en el siguiente apartado.

- La máquina deberá tener un cierto nivel de inteligencia en la toma de decisiones, haciendo sentir al jugador en la medida de lo posible que está jugando contra otra persona.

  En este proyecto, lo esencial de nuestra aplicación será que un ordenador pueda jugar una partida de nuestro juego de cartas sin que un ser humano le tenga que decir qué es lo que tenga que jugar. Por lo tanto, necesitaremos tener desarrollada una inteligencia artificial que sea capaz de dar una respuesta que hubiera podido dar una persona estando en la misma situación del ordenador.

- La toma de decisiones no puede tomar demasiado tiempo.

  Necesitaremos que la función de toma de decisiones sea lo más rápida posible para no hacer esperar al jugador más tiempo de lo que necesitaría un humano.

3.3. Casos de uso

1. Desarrollo de una partida de 'Anarquía':

   A la hora de ejecutar el programa, el desarrollo de cualquier partida tendrá que ser el siguiente:

   - Se crea una baraja de cartas.
   - Se baraja la carta.
   - Se reparten las cartas a todos los jugadores.
   - Cada jugador elige la estrategia que usará durante la partida.
   - Los jugadores eligen la carta que jugarán en cada turno.
   - Cuando se acaben las cartas de todos los jugadores, la partida se acabará y se mostrará la puntuación de todos los jugadores.

2. Elección de estrategia:

Durante el trascurso de una partida, los jugadores necesitan elegir una estrategia en la que basarán sus jugadas en la partida. El funcionamiento de esto será el siguiente:

- Se recorren todas las cartas de la mano del jugador.
- Dependiendo del palo y el número de la carta, se valorará más o menos una estrategia u otra.
- Se devolverá la estrategia con mayor puntuación.

3. Elección de carta a jugar:

   A la hora de ejecutar el programa, necesitamos realizar la elección de las cartas que los jugadores jugarán. El funcionamiento de esta función será la siguiente:

   - Exploramos el arbol de decisiones que se irá generando conforme vayamos probando cartas a jugar.
   - Al llegar al final del arbol que se genera, devolveremos el resultado para el jugador de haber tomado las decisiones de esa rama.
   - Conforme a la forma en la que funciona MiniMax, y nuestra implementación de tal algoritmo, devolveremos al final la mejor jugada, con la que conseguirá mejor puntuación el jugador

   Cabe destacar que todos los pasos descritos aquí se detallan mejor en la parte del documento dedicada al diseño de la aplicación, en el capítulo 5.

Capítulo 4

Análisis de la aplicación

4.1. Elección de lenguaje

Ya que, en la propuesta de este Trabajo de Fin de Grado no hay restricciones en cuanto al lenguaje de programación a usar, tenemos la opción de elegir el lenguaje en el que queremos que se desarrolle nuestro proyecto. Por lo tanto, a partir de aquí tenemos dos caminos: o usamos un lenguaje de programación nuevo para nosotros, o usamos uno en el que estemos más familiarizados.

Durante la carrera de Ingeniería del Software impartida aquí, en la Universidad de Sevilla, se han impartido asignaturas en las que se usan distintos lenguajes de programación. Algunos eran de propósito general, como Java, C, o Python, y otros lenguajes usados para propósitos más específicos, como Matlab, o PHP.

Por otro lado, estando como estudiante erasmus en Noruega, en Universitetet i Tromsø, aprendí un lenguaje más, llamado Ruby, y que en algunas ocasiones es bastante similar a Python.

Esto nos deja una infinidad de lenguajes sin explorar, como podrían ser Visual Basic, Go, Perl, Rust… Usar uno de estos lenguajes sería algo muy instructivo, ya que no se ha llegado a enseñar ninguno de estos en el grado, y podría ayudar en el futuro a la hora de buscar trabajo. Sin embargo, el aprendizaje que esto supondría llevaría consigo una carga temporal que, usando un lenguaje ya conocido, no merecería la pena.

Por lo tanto, la ruta escogida será la siguiente: usar un lenguaje ya conocido. Entre los lenguajes aprendidos durante el grado, usaremos un lenguaje de propósito general. Por lo tanto, tendremos 4 alternativas: Python, Java, C y Ruby.

Entre estos lenguajes, uno de los lenguajes que más destaca por su afinidad por los desarrolladores en la encuesta realizada por StackOverflow en 2019 [5] es Python el segundo mejor valorado por los encuestados. En comparación, C se encuentra en el puesto 24, Java en el puesto 18, y Ruby en el puesto 21. Por otro lado, entre los lenguajes más

odiados, nos encontramos que Python no se encuentra en dicha lista, Java se encuentra en el décimo puesto, C en el cuarto, y Ruby sería el séptimo de los lenguajes más odiados.

Teniendo en cuenta la popularidad de Python, podemos llegar a la conclusión de que Python será el lenguaje para usar si lo que queremos es rapidez no solo a la hora de desarrollar, por la forma en la que está hecho el lenguaje a la hora de programar, sino que, si surge una duda, será la tecnología en la que se encontrará una solución más fácilmente si en cualquier momento surgiera una duda. Además de esto, será más fácil de encontrar librerías que nos faciliten el desarrollo de la aplicación. Por lo tanto, la elección de lenguaje a usar será Python.

Por otro lado, uno de los puntos a realizar, aparte de un algoritmo de elección de jugada para el juego de cartas, queremos realizar una interfaz de usuario que permita visualizar con facilidad la situación del juego.

Para conseguir hacerlo, se han investigado distintas posibles librerías de Python para crear interfaces de usuario, entre los cuales destacaremos dos: Pygame y Kivy.

Por una parte, Pygame es una librería de Python ideada para el desarrollo de videojuegos. A la hora de trabajar con esta librería, uno se da cuenta de que la forma de desarrollar una aplicación con esta librería es muy simple, pero si uno quiere hacer cosas como animaciones, se da uno cuenta de que todo tiene que ser gestionado por uno mismo, y que todo tiene que estar contenido en un mismo bucle “while”. Claramente, tiene que haber una mejor forma de hacer esta parte.

Aquí es donde entra Kivy. En vez de usar un bucle, y meter ahí todo el contenido del programa, la forma de desarrollo de Kivy está basado en objetos, pudiendo separar las funcionalidades del código. Además de eso, incluye objetos prefabricados como botones, sliders, cuadros de texto… que pueden ser personalizados a gusto del programador. Además, con eventos que forman parte de la librería, como las animaciones se pueden codificar simplemente diciendo hasta dónde tiene que moverse en la pantalla el objeto a animar, el tiempo que tiene que durar dicha animación, y opciones que pueden enriquecer la animación, como cambiar el tamaño del objeto, el tipo de transición de la animación…

Capítulo 5

Diseño de la aplicación

5.1. Los inicios

Para empezar a pensar cómo se diseñará un juego, lo primero que tendremos que hacer será pensar en los elementos que lo conforman.

Por una parte, tenemos los componentes que dan sentido a nuestro juego de cartas. Estos elementos son los jugadores, las cartas, y la mesa que ejerce como intermediaria para hacer el intercambio para que las cartas pasen de un jugador a otro.

Por otra, tenemos la inteligencia del sistema. Esto no solo incluye la forma en la que los jugadores juegan sus cartas, sino que, además, define la forma en la que un jugador juega su carta, cómo recibe una carta jugada, cómo se deduce quién se lleva qué cartas, cómo se calculan los puntos de cada jugador…

Y, por último, la representación gráfica del conjunto. Aquí, tendremos que definir las posiciones de todos los jugadores, la información que queremos que aparezca sobre ellos, o cómo queremos que se representen los pasos del juego.

Una vez tenemos todos estos puntos claros, lo que nos queda por hacer será realizar el diseño de los componentes que conforman nuestro proyecto.

5.2. Los componentes esenciales

Para tener una diferenciación clara de todos los elementos que conforman nuestra aplicación, se ha decidido crear objetos que contengan la información necesaria de cada uno de los elementos del proyecto.

5.2.1. El objeto carta

En un principio, las cartas se representaron como una cadena de caracteres que tenían el siguiente formato: dos números, que representaban el valor de la carta, y una letra mayúscula, que representaría el palo de dicha carta.

Esta representación al principio fue suficiente para empezar a trabajar. Sin embargo, esto causó problemas a la hora de realizar el algoritmo de búsqueda, dado que al intentar explorar el arbol de decisiones, uno tenía que deshacer jugadas. Con este sistema de representación, uno no sabía cosas como a quién le perteneció la carta, quién es el actual dueño de la carta, o la imagen de la carta para su representación en la interfaz gráfica. Es por ello por lo que se decidió cambiar la representación de las cartas y hacer objetos propiamente dichos.

En el objeto **'Card'**, representando a las cartas de nuestro sistema, nos encontramos las propiedades que definirán de forma completa a una carta. Por un lado, tenemos el valor y palo de la carta, representados por enumerados, si la carta está en la baraja o no, representado con un booleano, y la imagen, representado con una cadena de caracteres. Estos valores serán los principales para la representación de una carta. Por otro lado, tendremos propiedades extra, como quién es el primer propietario, el turno en el que ha sido jugada, si ha sido ganada por un jugador, quién es el que tiene esta carta, y la posición en la mano.

5.2.2. La baraja

En el objeto **'Deck'** representamos a la baraja de cartas que usaremos en el juego. Aunque solo tiene una propiedad, **'cards',** una lista representando las cartas de la baraja, lo metemos en un objeto por la facilidad de crear funciones que solo afecten a la baraja. Por ejemplo, al crear un objeto del tipo **'Deck',** automáticamente se pobla la lista de cartas con todas las cartas necesarias para jugar una partida. Además de esto, contamos con la función de poder sacar un número **'X'** de cartas de la baraja, y otra para mezclar las cartas.

5.2.3. El jugador

En el objeto **'Player'** tendremos toda la información necesaria para identificar a un jugador. Dentro de esta información tendremos el nombre del jugador, la mano de cartas que podrá

jugar, las cartas que ha ganado, su puntuación, la carta que ha seleccionado, si es controlado por el ordenador o no, su estrategia, que podrá elegir entre la opción de jugar para conseguir cartas de un palo en concreto, cualquier carta, sin importar el palo ni el valor de la carta, o las cartas de cualquier palo que sean menores o iguales a diez.

5.3. La inteligencia del sistema

Una vez tenemos todos los elementos que interactuarán entre si definidos, necesitamos, por fin, de darles una inteligencia que sea capaz de jugar a una partida de **'Anarquía'.** Para conseguirlo, vamos a ir paso por paso. En este caso, para que el juego pueda empezar, lo primero que necesitaremos será crear una función que encuentre la estrategia ideal para el jugador:

5.3.1. Elegir estrategia

Para elegir una estrategia que se adapte a la mano que nos ha tocado, lo primero que haremos será, obviamente, mirar a todas las cartas.

Por cada estrategia, tendremos un contador. Al ir carta por carta, dependiendo del tipo de carta, sumaremos un número u otro dependiendo del palo y del valor.

- Si estamos contando para un contador pendiente del palo, y si la carta es del palo que estamos revisando, sumaremos dos puntos en el contador si la carta tiene un valor mayor de diez. Sumaremos un punto en el contador si la carta tiene un valor entre 5 y 10, y no sumaremos nada si la carta es menor de 5.
- Si estamos contando para el contador pendiente de la estrategia **'no suit'**, sumaremos 0.75 puntos en el contador si la carta tiene un valor mayor de diez. Sumaremos 0.5 puntos en el contador si la carta tiene un valor entre 5 y 10, y no sumaremos nada si la carta es menor de 5.
- Si estamos contando para un contador pendiente de la estrategia **'misère'**, sumaremos un punto en el contador si la carta tiene un valor entre 7 y 10. Sumaremos 0.5 puntos en el contador si la carta tiene un valor entre 5 y 6, y no sumaremos nada si la carta es menor de 5.

Al terminar contando los puntos de cada estrategia, elegiremos la opción que tenga el contador con mayor puntuación.

5.3.2. Elección de carta

El jugador, una vez tiene definida la estrategia con la cual ganará puntos, lo que tenemos que hacer es conseguir una forma de elegir una carta que tenga cierta inteligencia.

Para conseguirlo, adaptaremos el algoritmo 'Minimax' con poda alfa-beta para que funcione con varios jugadores. Para conseguirlo, el algoritmo funcionará de la siguiente manera:

- Lo primero será declarar los valores iniciales que necesitaremos dar a nuestra función de búsqueda. Estos valores serán:
    - El estado inicial en el que se producirá la evaluación.
    - La profundidad a la que se explorará en el arbol de decisión.
    - El valor de Alfa, que al principio de la llamada a esta función siempre valdrá el valor mínimo que pueda tener, en este caso hemos puesto que sea -1000.
    - El valor de Beta, que se iniciará con el valor máximo que pueda tomar la evaluación. Este valor se ha decidido que sea 1000, que es un valor que nunca se alcanzará por medios normales.
    - El jugador al que le tocará jugar una carta en la situación actual.
    - El jugador que hace la primera llamada a la función de evaluación. Esta será la variable que decida si la situación en la que nos encontremos en nuestro algoritmo si lo que buscamos es maximizar o minimizar la jugada. Si esta variable es igual que el jugador al que le toca jugar una carta, maximizará la jugada, mientras que, si es distinta, lo minimizará.
- Comenzamos con el primer jugador.
- Exploramos el arbol, jugando cartas de cada jugador hasta que lleguemos al límite de profundidad o el juego haya terminado. Es entonces cuando evaluamos la validez de este camino. Para ello, ejecutamos la función de evaluación.
    - Esta función, como sabemos, debe estimar la validez de una jugada dándole un valor numérico. En nuestro caso, la función implementada ha sido el cálculo de los puntos del jugador que está buscando la carta que le dé la máxima puntuación posible. Así, el jugador que busca maximizar la jugada

intentará buscar su máxima puntuación, mientras que el resto de los jugadores intentará minimizarla.

    - Si en el algoritmo estamos mirando la posible jugada del primer jugador, el valor alfa será el máximo entre el alfa actual y la puntuación total de la siguiente rama.
    - Si en el algoritmo estamos mirando la posible jugada del resto de jugadores, el valor beta será el mínimo entre la beta actual y la puntuación de la siguiente rama.
- Si en cualquiera de los casos beta es menor o igual a alfa, podamos y rompemos el bucle.
- Al final, al haber valorado todas las opciones, si estamos valorando las opciones del primer jugador, devolveremos alfa.
- Si estamos valorando las opciones de los jugadores que no son el primero, devolveremos beta.

La forma en la que hemos conseguido adaptar el algoritmo para que un algoritmo de búsqueda del tipo MiniMax funcione en un entorno con varios jugadores ha sido la de reducir un juego multijugador a uno de dos jugadores haciendo una asunción 'paranoica' [4]. Esto quiere decir, que a la hora de evaluar las opciones que tiene un jugador, se considerará que el resto de los jugadores ha hecho una coalición contra el jugador que tiene que decidir qué va a jugar. Por lo tanto, el resto de los jugadores se comportarán como si fueran uno, y su objetivo será minimizar la puntuación de este, mientras que el jugador que está decidiendo su jugada intentará maximizarla. Así, el valor de alfa y beta vendrá dado por la puntuación del jugador que decide qué carta jugará.

5.4. La interfaz de usuario

A la hora de realizar una interfaz de usuario, empezó el desarrollo usando la librería de **Python 'Pygame'**. Al principio, parecía que esta librería serviría para realizar el proyecto. Sin embargo, Al no tener experiencia en el uso de esta librería, y las pocas facilidades que da la librería en cuanto al uso de la pantalla (ya que todo lo que quieras hacer, tiene que ser codificado, sin presets que poder usar), además de que antes de empezar esta parte del código se hizo la estructura del juego sin pensar en cómo se haría esta parte, hicieron que se tuviera que pensar en una mejor forma de hacer esto.

Es por ello, que se buscó una mejor forma de hacer esta parte. En un principio, se pensó en usar un motor gráfico como **Godot.** Sin embargo, tras unos días haciendo tutoriales sobre este, uno se da cuenta de que al no haber hecho nada en este tipo de 'frameworks' el tiempo que sería necesario para aprender a usar **Godot**, aprender a realizar una inteligencia artificial en este, y adaptar la inteligencia ya hecha iba a ser más perjudicial que adaptar el programa para poder usar otra librería de Python.

Al final, la decisión de cómo implementar una interfaz gráfica se resolvió usando la librería **Kivy**, más fácil de usar que **Pygame** y sin una diferencia tan grande del flujo de trabajo como con **Godot**.

Una vez tenemos resuelta la cuestión de cómo vamos a realizar la interfaz gráfica programáticamente, toca pensar en cómo se verá la aplicación.

El diseño que usaremos será el típico de los juegos de cartas hechos para ordenador existentes. Así, dejaremos la parte baja de la ventana para representar al jugador controlado por el usuario, mientras que en la parte superior pondremos a los jugadores controlados por la inteligencia artificial, dejando el centro de la pantalla para la zona de juego, donde mostraremos las cartas jugadas por cada uno de los jugadores.

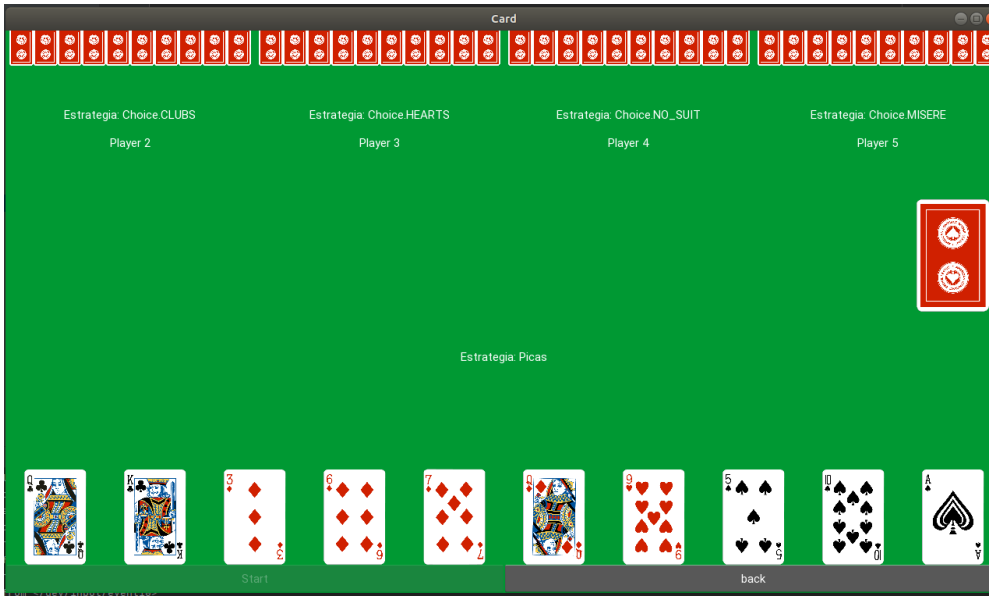


FIGURA 6 DISEÑO DE LA INTERFAZ GRÁFICA

Como podemos ver, nuestro diseño no es demasiado innovador, pues es usado por infinidad de juegos de cartas, pero cumple su función de dar la información necesaria al usuario para poder jugar una partida de 'Anarquía'.

La interacción del usuario con los diferentes botones de la aplicación serán los que determinen el flujo de la partida, que será el siguiente:

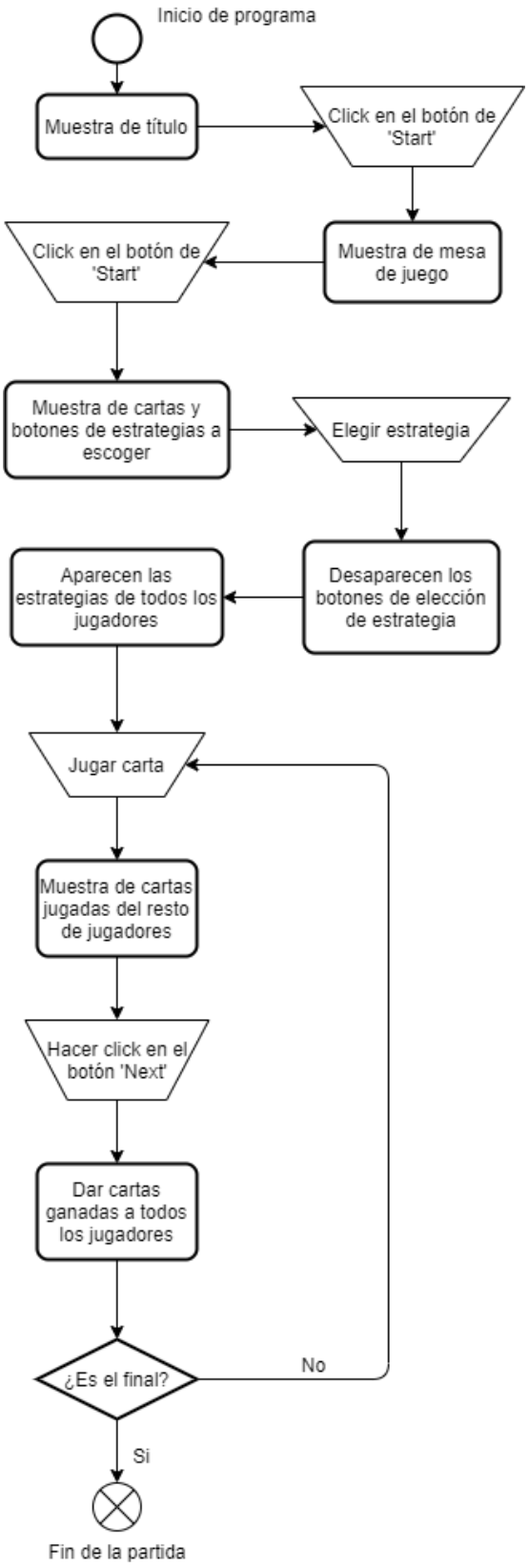


FIGURA 7 DIAGRAMA DEL TRANSCURSO DE UNA PARTIDA DE 'ANARQUÍA'

Capítulo 6

Implementación

Todo el código del programa lo podremos encontrar en el siguiente repositorio de github:

https://github.com/ghostedMonster/TFG/tree/master/anarquia

Para poder ejecutar el código, necesitaremos tener instalado en nuestro sistema una versión de Python que se encuentre entre la 3.6.9 y la 3.7.7. Es posible que en el futuro esta restricción no tenga que cumplirse y que funcione en versiones superiores de Python. Sin embargo, debido a las librerias usadas para este proyecto, no es posible ejecutar la interfaz gráfica en versiones de Python superiores a la 3.7.7. Si estamos usando un sistema operativo en base Linux, lo más normal es que lo tengamos instalado. Por si acaso, para asegurarnos de si está instalado o no, abrimos una terminal, y tecleamos el siguiente comando:

python3 -V

Si tenemos una versión del lenguaje de programación instalada, nos debería aparecer la versión que tenemos instalada.

Si por algún casual no tuviéramos instalado Python, puede visitar la página oficial de Python: https://www.python.org/downloads/ , e instalar una versión válida de Python en nuestro ordenador.

A parte de esto, si queremos ejecutar la versión gráfica del juego de cartas, necesitaremos instalar Kivy, una librería de Python que podemos encontrar en los repositorios pip. Si no tenemos instalado pip, puede ir a la página oficial de Pip: https://pip.pypa.io/en/stable/installing/ y ver cómo instalarlo en su máquina.

De la misma forma, nos vamos a la página oficial de Kivy: https://kivy.org/#download para instalar de manera correcta kivy en nuestra máquina.

Una vez hecho esto, nos vamos a la carpeta de nuestro juego, y n una terminal, ejecutamos:

python3 try_mini_max.py

Y se nos ejecutará una versión de nuestro juego de cartas que juega sola en una terminal.

Si ejecutamos:

python3 main.py

Podremos jugar nosotros a una versión del juego de cartas que tiene una interfaz de usuario amigable.

Capítulo 7

Manual de usuario

Debido a que el juego **'Anarquía'** es muy simple en su ejecución, la interfaz se ha hecho de la misma forma, con un diseño intuitivo y fácil de usar.

Para empezar, al iniciar el programa se podrá ver la siguiente pantalla:

FIGURA 8 INTERFAZ GRÁFICA. PRIMERA PANTALLA

Al hacer clic en el botón de inicio, aparecerá lo siguiente:

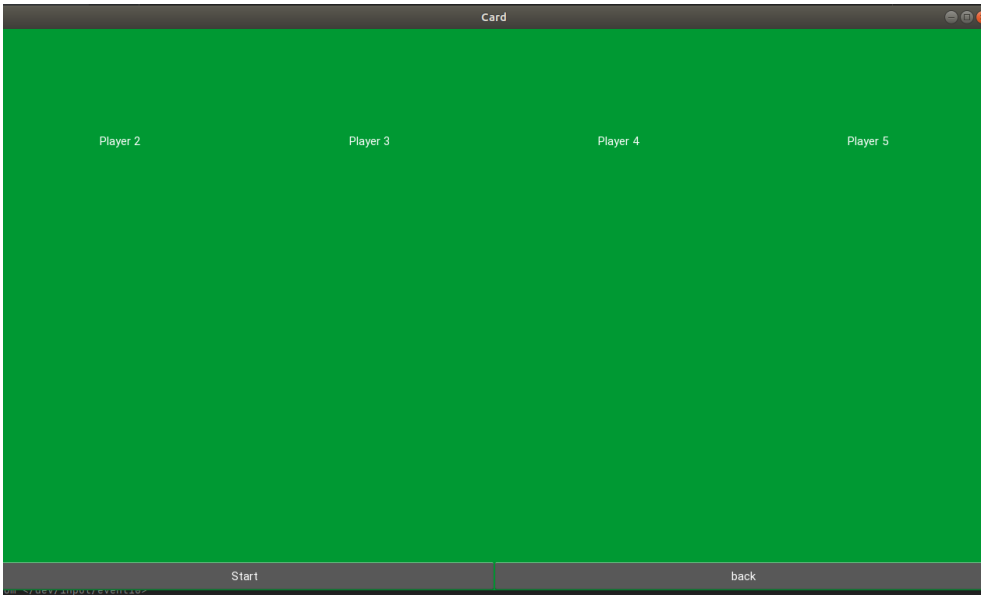


FIGURA 9 INTERFAZ GRÁFICA. MESA DE LA PARTIDA

Como podemos ver, en la parte de abajo podemos ver dos botones. Uno sirve para comenzar la partida, mientras que el otro sirve para volver a la pantalla anterior. Si pulsamos en el botón de iniciar partida:

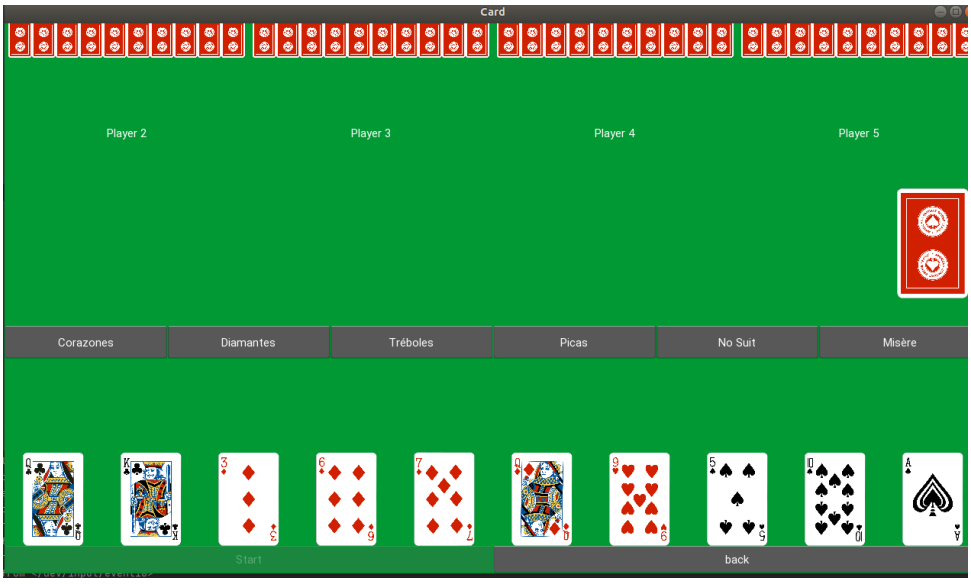


FIGURA 10 INTERFAZ GRÁFICA. PRINCIPIO DE LA PARTIDA

Veremos que todo el mundo ha recibido sus cartas, a la vez que aparecerán seis botones, cada uno representando cada una de las estrategias posibles a elegir por el jugador. Si pulsamos en cualquiera de ellos:

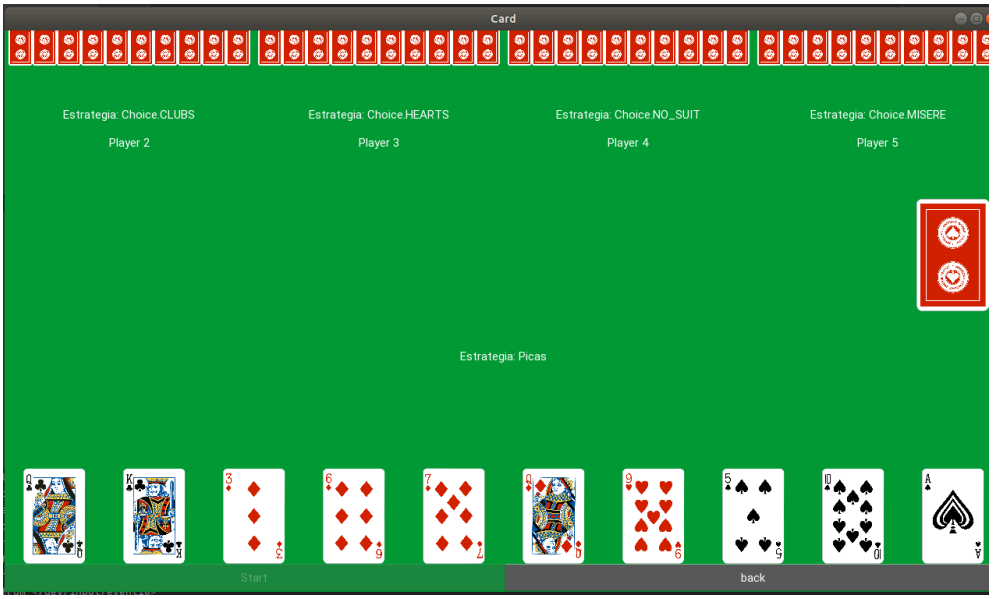


FIGURA 11 INTERFAZ GRÁFICA. ESTRATEGIA ELEGIDA

Los botones de elección de estrategia se eliminarán, dejando en su lugar no solo la estrategia de nuestro jugador, sino la del resto de jugadores. Una vez que las estrategias están definidas, lo que queda es empezar a jugar cartas. Para ello, haremos clic en la carta que queramos jugar:

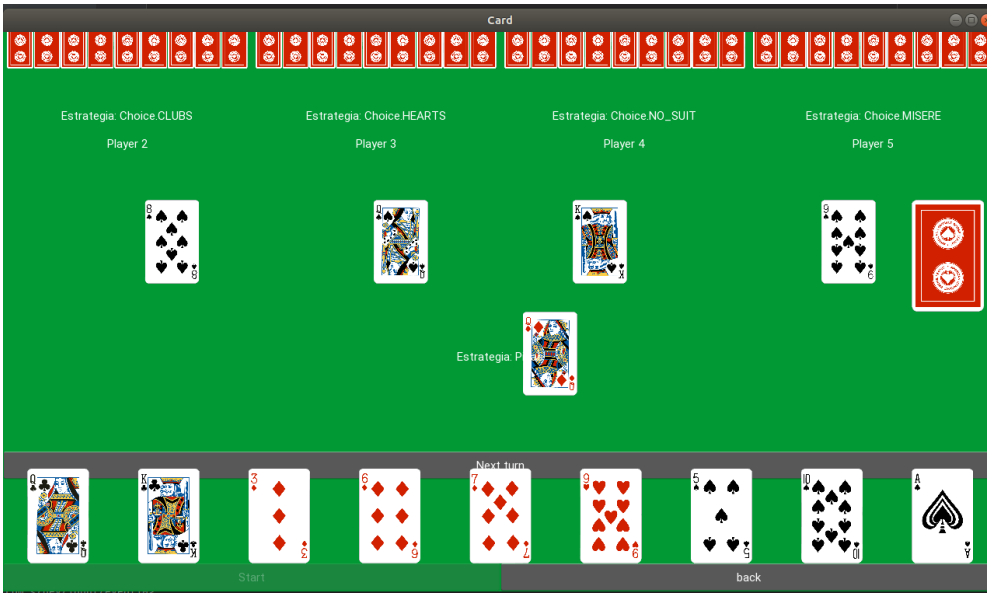


FIGURA 12 INTERFAZ GRÁFICA. CARTA JUGADA

Al haber jugado una carta, lo primero que ocurrirá será que se mostrarán todas las cartas jugadas de todos los jugadores, como vemos en la imagen. Además, aparecerá un botón para pasar al siguiente turno. Al hacer clic a dicho botón:

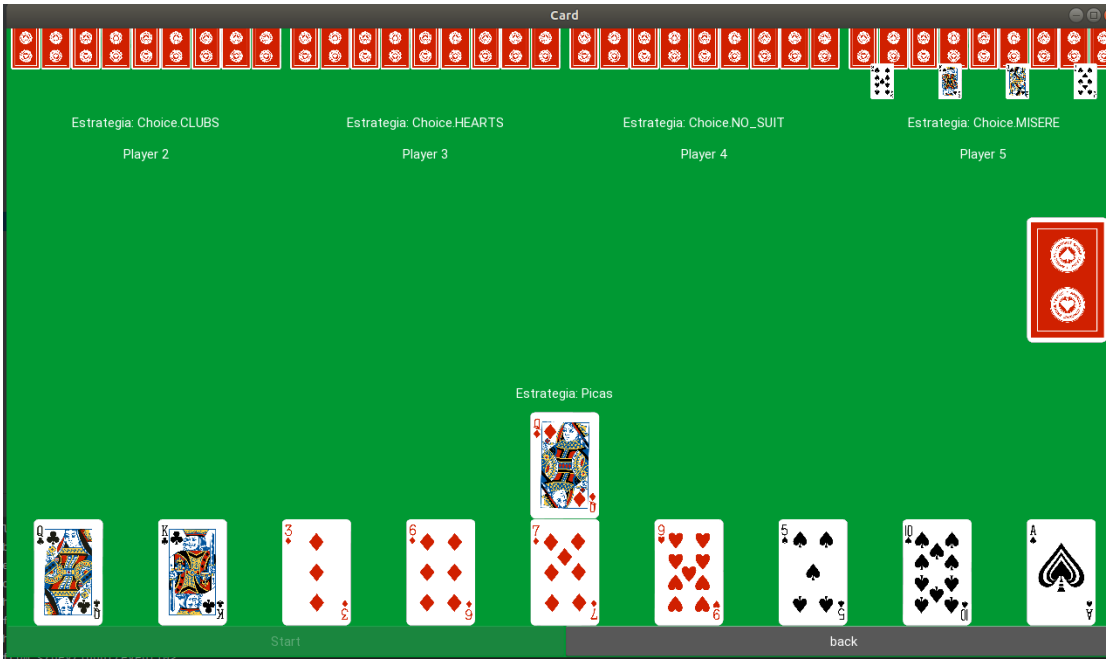


FIGURA 13 INTERFAZ GRÁFICA. CARTAS GANADAS

Todas las cartas jugadas se devolverán a los jugadores que hayan ganado esa ronda. Además, el botón para pasar de ronda desaparecerá, dado que estamos en una nueva ronda. Podremos seguir jugando cartas, hasta que nos quedemos sin cartas. Una vez eso ocurra, al dar clic al botón de siguiente:

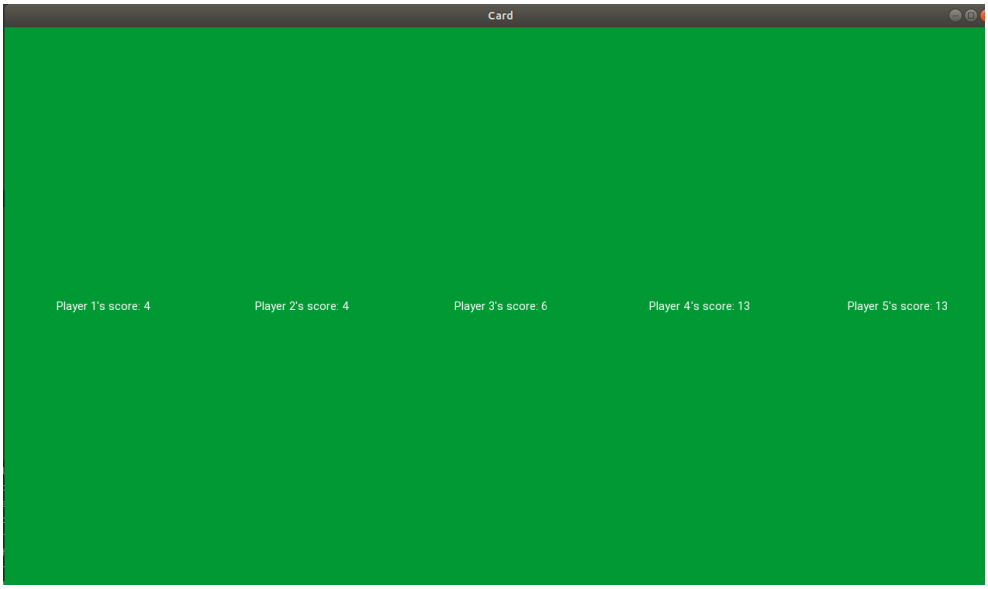


FIGURA 14 INTERFAZ GRÁFICA. PUNTUACIONES

Veremos las puntuaciones finales de todos los jugadores.

Capítulo 8

Alternativas tecnológicas

A la hora de elegir la tecnología a usar para nuestra aplicación, se ha hablado de distintas tecnologías que han sido usadas durante el grado de Ingeniería del Software impartida en la Universidad de Sevilla. Como lenguaje de programación de programación de carácter general, además de Python, se ha hablado también de Java, quizás el lenguaje más usado para enseñar en esta escuela. Es por ello por lo que vamos a comparar a Java con Python, [14] viendo cuales son las diferencias entre estos dos lenguajes, y las características que han hecho que al final el desarrollo de la aplicación se realizara en Python.

8.1. Tipado estático contra tipado dinámico

Una de las diferencias más grandes entre estos dos lenguajes es la forma en la que se manejan las variables. Por un lado, tenemos a Python, que declara sus variables de forma dinámica, mientras que Java lo hace estáticamente.

¿Pero esto qué quiere decir? Ahora mismo vamos a ver qué quiere decir el tipado estático y dinámico:

8.1.1. ¿Qué es el tipado?

El tipado es la aplicación de un tipo a una variable dada. Los tipos de datos ayudan a contextualizar datos para los lenguajes de programación. Por ejemplo, si sumamos dos números enteros, el resultado será la suma de esos dos números (por ejemplo, 2 + 2 = 4). Sin embargo, si fueran dos cadenas de caracteres, el resultado de la suma sería la concatenación de las dos cadenas de caracteres (por ejemplo, “2” + “2” = “22”, o “hello” + “world” = “helloworld”).

Si una parte de los datos dados es de tipo incorrecto, puede llevar a errores en la ejecución del código. La diferencia entre el tipado estático y dinámico es importante para comprobar los errores de tipo.

8.1.2. Tipado dinámico

El tipado dinámico consiste en comprobar los errores por tipo mientras se va ejecutando el código. Por ejemplo, miremos este código:

```
foo = “a string”
eggs = 2
if eggs + 2 > 3:
    print(“well done”)
else:
    print(foo + 1)
```

FIGURA 15 EJEMPLO DE TIPADO DINÁMICO

Aunque la suma entre un número entero y una cadena (como vemos al final, en print(foo + 1)) de un error de tipo, no lo dará en este caso, pues debido a que Python hace un tipado dinámico, y la ejecución del “else” nunca llegará a ejecutarse.

8.1.3. Tipado estático

El tipado estático, por otro lado, necesita comprobar el tipo de sus variables antes de ejecutar el código. Las variables deben ser identificadas antes de que el código sea compilado. Por ejemplo, si en Java tuviéramos un código similar al comentado anteriormente en Python, veríamos que, al intentar compilar el código, obtendríamos un error por intentar sumar un entero con una cadena de caracteres.

8.2. Espaciado en blanco

Python, al contrario de otros lenguajes, usa sangrías para separar fragmentos de código en bloques. Por otro lado, Java, como muchos otros lenguajes, separa los bloques de código usando llaves, definiendo el inicio y el fin de dicho bloque. La ventaja de usar sangrías es que fuerza al programador a escribir código de forma que sea fácil de leer, además de evitar errores derivados de olvidar una llave.

8.3. Rendimiento

En este aspecto, cuando nos referimos a una carrera entre Python y Java de velocidad rendimiento, en la mayoría de los casos, Java le gana a Python. Esto ha sido probado en benchmarks en los que, en algunos casos, Java es más de un orden de magnitud más rápido que Python.

Sin embargo, esto no quiere decir que Python no sea un lenguaje de programación menos válido, ya que con implementaciones como CPython, podemos conseguir mejor rendimiento, y existen servicios como Youtube, Spotify o Quora funcionando en Python. Esto demuestra que el lenguaje puede ser usado a una gran escala.

Además, mientras Python pierde en velocidad, le gana a Java en flexibilidad. Como es generalmente más fácil de usar, Python puede ayudar en la mejora de productividad en el desarrollo tanto en equipo como en solitario.

8.4. ¿Por qué se ha elegido Python al final?

Tras haber visto algunas diferencias entre Python y Java, queda resolver la cuestión que se nos presenta a continuación: ¿Cuál es la razón por la que hayamos preferido Python sobre Java?

La razón es simple: Es muchísimo más cómodo de usar que Java. Cosas como el no tener que acordarse de abrir y cerrar llaves a la hora de hacer un bucle, y la facilidad para leer el código en este lenguaje han hecho que se prefiera usar Python sobre Java.

Capítulo 9

Análisis temporal y costes de desarrollo

9.1. Planificación

Al ser el Trabajo de Fin de Grado una asignatura en la que se considera que tiene que ocupar un espacio temporal de unos doce créditos ECTS, donde se hace la equivalencia de que cada crédito representa unas 25 horas de trabajo, se ha estimado que todo el trabajo correspondiente a este trabajo se realizaría en unas trescientas horas. Así, la planificación temporal se quedaría:

Documentación	Redacción de la memoria	80 h
	Búsqueda de información	20 h
Análisis y diseño	Diseño de objetos	20 h
	Estudio de tecnologías nuevas	20 h
	Análisis de los requisitos	20 h
Implementación	Implementación base de los objetos del juego	45 h
	Implementación de la inteligencia artificial	45 h
	Implementación de la interfaz gráfica	50 h
	Total	300 h

CUADRO 1 ESTIMACIÓN DE TIEMPO USADO

Esta fue la planificación inicial para este proyecto. Sin embargo, al no haber tenido nunca la experiencia de haber hecho un trabajo de estas dimensiones, estas estimaciones no pudieron cumplirse. Para entender cuáles fueron los problemas para que estas estimaciones se cumplieran, vamos a ver qué es lo que sucedió:

- Al principio del desarrollo, se pensó en un diseño de los objetos principales del juego (el jugador, la carta y la baraja) de manera errónea, ya que no eran objetos en

un principio. Eran simples listas de cadenas de caracteres que, al haber avanzado un poco en el desarrollo, se determinó que no valía para nada lo que había hecho y tuvo que ser reemplazado, cambiando los elementos que usábamos para representar a estos elementos básicos para nuestra partida por objetos Python que pudieran ser referenciados sin problemas.

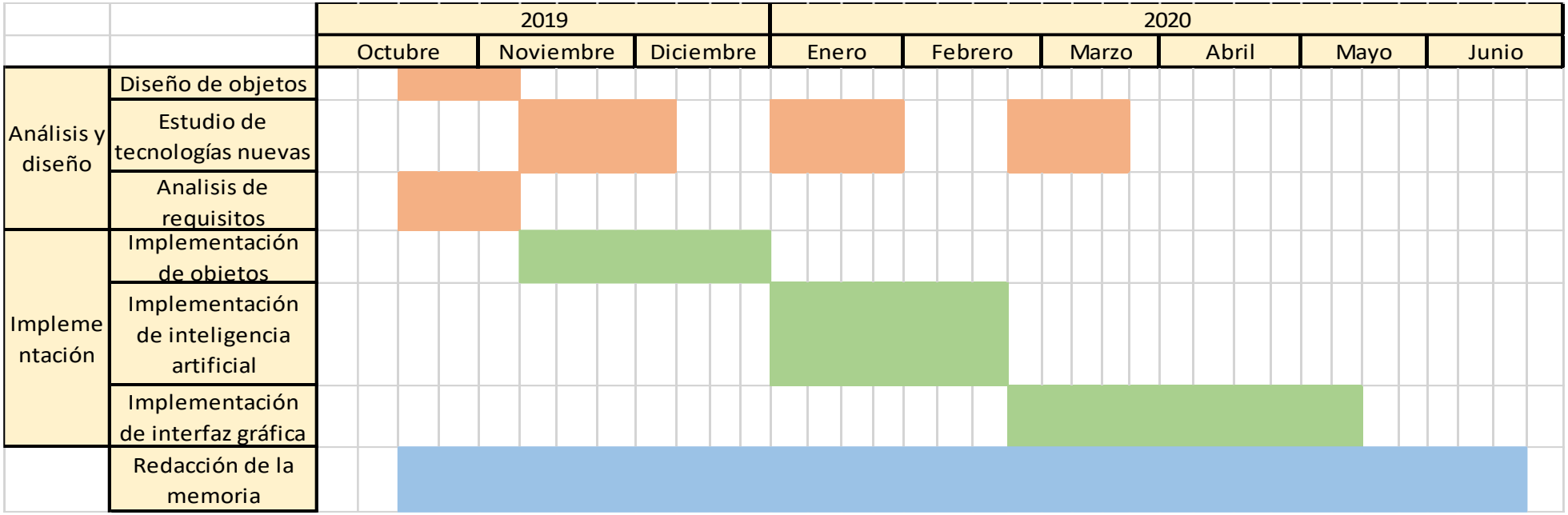
- Por otro lado, aunque se hizo este primer cambio de la estructura básica del proyecto, se vio que el diseño en el momento no era lo suficientemente bueno para poder ser usado en una interfaz gráfica. Por lo tanto, tuvo que hacerse otro rediseño a los objetos de la aplicación para que pudieran ser usados por la interfaz gráfica.
- Además de esto, a la hora de realizar la interfaz gráfica, sobre todo por la falta de experiencia no se escogió una librería adecuada para lo que se iba buscando. Es por ello por lo que se buscó otros métodos para hacer esta parte del código. Tras unas semanas buscando alternativas, se encontró que Kivy era la forma mas sencilla de poder hacer esta tarea con Python.

Una vez vistos los problemas encontrados a la hora de realizar el trabajo, el tiempo usado para el trabajo fue el siguiente:

Documentación	Redacción de la memoria	90 h
	Búsqueda de información	20 h
Análisis y diseño	Diseño de objetos	25 h
	Estudio de tecnologías nuevas	40 h
	Análisis de los requisitos	20 h
Implementación	Implementación base de los objetos del juego	45 h
	Implementación de la inteligencia artificial	60 h
	Implementación de la interfaz gráfica	70 h
	Total	370 h

CUADRO 2 TIEMPO USADO

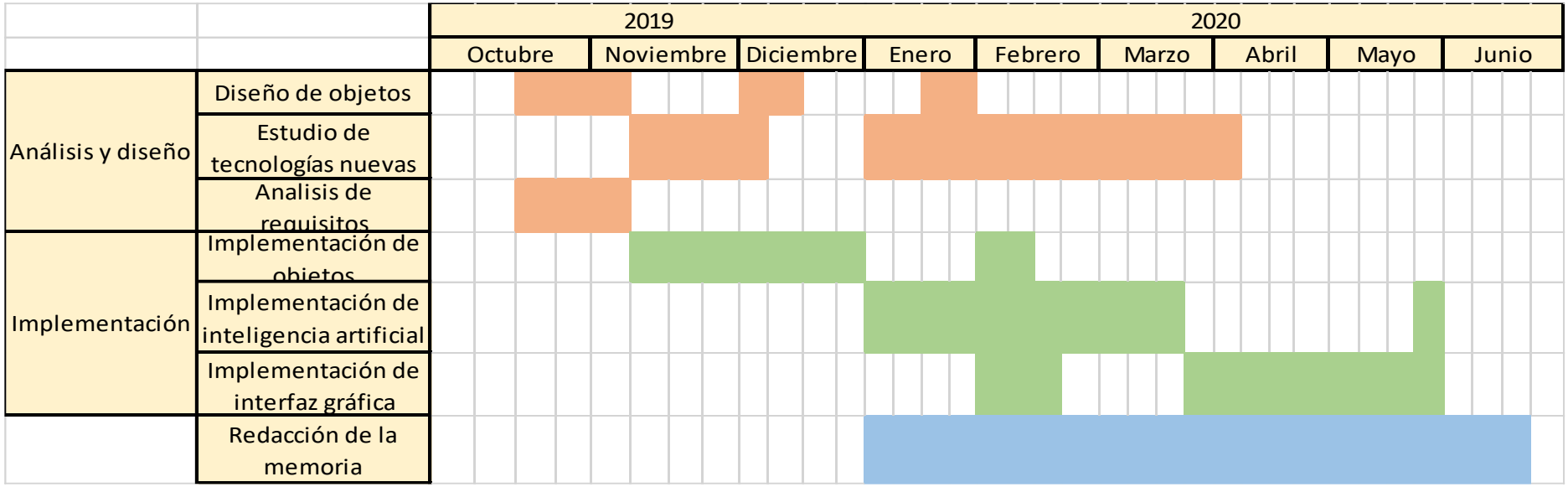
9.2. Estimación de tiempos en Gantt



CUADRO 3 DIAGRAMA DE GANTT DE ESTIMACIÓN DE TIEMPO

9.3. Realidad del proyecto

Aunque se estimó la progresión del trabajo como hemos visto arriba, el trabajo en si ha ido de la forma siguiente:



CUADRO 4 DIAGRAMA DE GANTT SOBRE EL TIEMPO REAL USADO

Capítulo 10

Pruebas

A la hora de desarrollar una aplicación, la parte más importante del trabajo es asegurarse de que el código no solo funciona en la máquina en la que se está trabajando, sino que funciona en equipos con distintas características. Es por ello por lo que el proyecto ha sido probado en distintas máquinas.

Es por ello por lo que se ha probado en la mayoría de los dispositivos que se tienen al alcance. Para empezar, la máquina de desarrollo, un portátil que cuenta con un procesador Intel Core i5 de séptima generación de 2.50 GHz de cuatro núcleos, 12 GB de memoria RAM usando Ubuntu 18.04 como sistema operativo. A parte de este, se ha probado en otro ordenador portátil, esta vez corriendo en Debian, con 8 GB de memoria RAM, y un Intel Core i5 de cuarta generación de 2.50 GHz de cuatro núcleos. Además, también ha sido probado en un ordenador de sobremesa con un procesador AMD Rizen 5 2600 de seis nucleos de 3.4 GHz y 16 GB de memoria RAM, usando el sistema operativo Windows 10. En todos ellos, el programa funciona sin problemas, claro está, instalando todo lo necesario para que el código funcione.

Por otro lado, a la hora de probar en busca de bugs o similares, la única forma que ha habido de comprobar que el código ha sido bien formado ha sido ejecutando el código, o con la función de debug presente en PyCharm. Debido a la dificultad de poder probar que una inteligencia artificial hace bien su trabajo, juntado a su vez con la imposibilidad de probar que una interfaz gráfica presenta todos los elementos sin problemas, ha resultado en que se decidiera al final no realizar pruebas automatizadas.

Por ejemplo, a la hora de probar la función de elección de la estrategia que va a usar un jugador, lo que se hizo fue un programa de prueba que hace una baraja, la mezcla y la reparte a todos los jugadores. Una vez repartida, cada jugador elige una estrategia. Una vez hecho esto, se imprime

en pantalla tanto la elección elegida, como la mano de cartas que tiene el jugador. Como vimos anteriormente en la explicación de cómo funcionaba la función de búsqueda de estrategia, usa una serie de contadores para elegir la estrategia a usar, y para cada contador, una lista para determinar el peso de la carta para ese contador. Al principio, las listas de pesos para las cartas fueron [2, 1, 0] para los contadores de palo, [1, 0.5, 0] para 'no suit', y [1, 0.5, 0] para 'misère', entendiendo que para las listas de pesos de palos y 'no suit' el primer peso es para cartas de valor comprendidos entre el 10 y el as, el segundo para valores entre el 5 y el 9, y el tercero para valores inferiores a 4.

Con estos valores, la mayoría de las veces el programa elegía estrategia a escoger la de 'no suit', y a veces la de 'misère'. Era raro ver que alguno de los jugadores eligiera algún palo como estrategia. Como podemos suponer, esto no es un resultado aceptable, ya que, aunque sea posible que necesitemos estas estrategias, no es realmente la primera opción a la hora de escoger una estrategia. Por lo tanto, hubo que ir probando distintos valores en los pesos hasta que se encontró una opción que da resultados más realistas. Al final, los pesos que se usaron fueron [2, 1, 0] para los contadores de palo, [0.75, 0.5, 0] para 'no suit', y [1, 0.5, 0] para 'misère'.

Por otro lado, a la hora de comprobar que el algoritmo de decisión funciona correctamente, se amplió el programa realizado para comprobar el funcionamiento de la prueba de la elección de estrategia para que pueda completar una partida completa de nuestro juego. Este programa, al terminar un turno de la partida, imprimirá por pantalla la mano del jugador, la estrategia, las cartas ganadas por el jugador, los puntos en esa ronda y la carta que se jugará en el turno por ese jugador.

A la hora de haber terminado una primera versión del algoritmo, una vez se intenta ejecutar por primera vez, uno se da cuenta de que no es capaz de terminar un turno, o al menos no es capaz de terminar un turno en seis horas. Tras semanas buscando la razón de por qué está pasando esto, uno se da cuenta de que, en esta primera versión del algoritmo, no hay ningún tipo de límite en la profundidad de búsqueda. Resulta que el algoritmo, intentando explorar todas las posibles posibilidades, no llega a terminar en todo este tiempo. Esto hizo que, en la siguiente versión de este algoritmo, se limite el número de turnos. Con este cambio, limitando la profundidad a un turno encontramos que el ordenador termina una partida en unos 5 segundos. Sin embargo, si intentamos aumentar el limite a dos turnos, parece ser que nuestro algoritmo

no es capaz de terminar el primer turno en horas. Teniendo en cuenta que una decisión no debería llevar más de unos segundos para que, a la hora de que un jugador real se enfrente a la máquina no se piense que el programa se ha quedado colgado.

Sin embargo, en este punto el desarrollo de este algoritmo no ha terminado. Todavía nos queda realizar la poda alfa-beta.

Tras unas semanas trabajando en varias cosas del proyecto, llegamos al final con una versión de poda alfa-beta. Sin embargo, a la hora de probar el algoritmo, vemos que no reduce el tiempo de ejecución con respecto a la versión anterior. Y es que, viendo los resultados de las elecciones escogidas por el algoritmo para jugar cartas, siempre juega la última carta de la mano del jugador. Esto nos lleva a pensar que el algoritmo, en este momento no funciona todo lo bien que debería, y recorre todo el arbol sin encontrar una opción óptima, devolviendo siempre la última opción investigada, en este caso la última carta.

Después de un tiempo siguiendo la investigación de qué es lo que ocurre con nuestro algoritmo, conseguimos que el algoritmo funcione correctamente, no explorando todo el arbol, cosa que queremos evitar, y dando unas elecciones con cierto sentido. Además, con respecto al rendimiento de este algoritmo final, vemos que es mucho más eficiente, pudiendo terminar una partida buscando en profundidad a un nivel en aproximadamente un segundo, y a dos niveles de profundidad alrededor de 10 minutos, cosa que en el algoritmo anterior no era capaz de terminar en horas.

Como la parte que más tiempo consume en la decisión de la carta a jugar explorando a dos turnos de profundidad es el principio del juego, se ha decidido que, para poder ofrecer una experiencia más satisfactoria al usuario final, y aportar un poco más de profundidad a la búsqueda en el algoritmo, se ha decidido que las primeras jugadas se jueguen con un nivel de profundidad de un turno de profundidad hasta que en la mano, los jugadores tengan cinco cartas, momento en el que el algoritmo buscará por la opción óptima mirando a dos turnos de profundidad.

Capítulo 11

Conclusión y desarrollos futuros

11.1. Conclusiones

Tras haber terminado el proyecto, queda comentar que ha sido un proyecto entretenido a la vez que educativo. Difícil en ciertos momentos, pero bastante gratificante.

Por un lado, el desarrollo de un juego de cartas es divertido, sobre todo si el que lo hace le gustan los juegos de cartas, cosa que he redescubierto tras hacer este proyecto. Aprender a hacer una interfaz en Python mientras se hace un juego de cartas es una experiencia bastante entretenida, sobre todo si uno empieza sin saber a hacer estas cosas.

Es por esto por lo que la experiencia, además de entretenida, ha sido muy instructiva. El hecho de que el desarrollo de este programa haya hecho que haya tenido que aprender a hacer interfaces gráficas ha hecho que uno se haya interesado más en el desarrollo de este tipo de características. Además, con la obligación de revisitar la inteligencia artificial y su implementación de esto en un juego de cartas de mas de dos jugadores ha sido un soplo de aire fresco que he cogido con mucho gusto, interesándome más en este tipo de usos de la programación.

Por supuesto, el haber tenido que realizar todo esto sin haber tenido una referencia de cómo se realiza este tipo de programas ha hecho que el desarrollo haya sido difícil. Para empezar, no he tenido muchos documentos de referencia que expliquen cómo se realiza una inteligencia artificial que use Minimax como su algoritmo para jugar a un juego de más de dos jugadores. Por esto, quizás esta no es la mejor forma de adaptar este algoritmo, pero es la que mejor he considerado para el sistema que he estado creando durante este curso.

Pero con todo, el resultado final es algo de lo que puedo estar orgulloso de haber terminado. Es cierto que podría haber sido programado mejor, y que la visualización de este sistema es mejorable, el resultado ha sido un producto del que puedo estar contento de haber hecho. Ha

sido mi primera vez en haber hecho muchos de los elementos que forman todo el conjunto, y el resultado ha sido algo que, aunque no sea perfecto, he sido capaz de terminarlo, y estoy bastante contento con ello.

11.2. Trabajos futuros

Está claro que, aunque se ha gastado mucho tiempo en el desarrollo de este proyecto, ha habido cosas que se han quedado en el tintero y que no han sido posible ser realizadas. No obstante, cabe destacar que existen muchas tareas que se podrían realizar para mejorar el programa desarrollado. Entre estas tareas, podemos destacar cosas como:

- Refactorizar el código para eliminar partes de código duplicado, a la vez que eliminar funciones y variables que al final no llegan a usarse.
- Añadir el paso del reparto de las cartas que sobran al principio del juego.
- Mejorar el rendimiento del algoritmo, a la vez que la inteligencia de este para que no tarde tanto en llegar a una decisión y que dicha decisión parezca más que ha sido hecha por una persona.
- Además de mejorar el algoritmo de decisión, podría darse un cierto carácter a cada uno de los personajes, dando valores aleatorios a cada uno de ellos para que jueguen de forma más arriesgada o defensiva.
- Implementar distintos tipos de evaluación a la hora de valorar la validez de una jugada para poder comparar el uso de la función de evaluación actual con otro tipo de evaluación.
- Añadir selectores de dificultad para los jugadores.
- Dar la posibilidad de cambiar los nombres de los jugadores.
- Poner tutoriales en la aplicación para dar una idea de forma más directa al usuario de cómo va el juego.
- Usar efectos de sonido o músicas de fondo.
- Dar la posibilidad de empezar una nueva partida en cualquier momento en la interfaz.
- Hacer ejecutables para plataformas como Windows o Android.
- Implementar el juego en una herramienta más dedicada a la creación de videojuegos, como Unity, Godot o GameMaker Studio.

- Se podría implementar el juego para que pudiera ser jugado por varios humanos a la vez a través de internet.

Referencias

[1] David Parlett (1977) *Anarchy*, Fecha de consulta: 20/10/2019. De https://parlettgames.uk/oricards/anarchy.html

[2] José Pablo Tobar Quiñones. (27/9/2017) *Anarquía*, Fecha de consulta: 20/5/2020, de sistemaspoliticos.org Sitio web: https://sistemaspoliticos.org/anarquia-idea-politica/

[3] Visme. *Aplicación para la realización del diagrama del árbol de decisiones*, Fecha de consulta: 12/4/2020, de Visme Sitio web: https://my.visme.co/

[4] Maarten P.D. Schadd, Mark H.M. Winands. Best-Reply Search for Multi-Player Games, Fecha de consulta: 10/6/2020, de *Maastricht University* Enlace: https://dke.maastrichtuniversity.nl/m.winands/documents/BestReplySearch.pdf

[5] StackOverflow. (24/5/2019) *Encuesta de StackOverflow sobre sus usuarios en 2019*, Fecha de consulta: 9/6/2020, de StackOverflow Sitio web: https://insights.stackoverflow.com/survey/2019#correlated-technologies

[6] Comunidad de Wikimedia Commons. (7/7/2018) *Imágenes de cartas sin copyright*, Fecha de consulta: 15/11/2019, de Wikimedia Commons Sitio web: https://commons.wikimedia.org/wiki/Category:SVG_playing_cards

[7] Comunidad de Wikipedia. (28/10/2019). *MiniMax*. Fecha de consulta: 5/3/2020, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Minimax

[8] Carlos Eduardo Plasencia Prado. (No especificado. El comentario más antiguo en este artículo es del 27/6/2016). *MiniMax*, Fecha de consulta: 25/10/2019, de DevCode Sitio web: https://devcode.la/tutoriales/algoritmo-minimax/

[9] Comunidad de Wikipedia. (30/8/2019) *Poda Alfa-Beta*, Fecha de consulta: 11/4/2020, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Poda_alfa-beta

[10] Comunidad de Geeks for Geeks. (No especificado. El comentario más antiguo en este artículo es del 18/2/2017) *Poda Alfa-Beta*, Fecha de consulta: 11/4/2020, de Geeks for Geeks Sitio web: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

[11] Jetbrains. *Página web de PyCharm*, Fecha de consulta: 11/4/2020, de JetBrains Sitio web: https://www.jetbrains.com/es-es/pycharm/

[12] Comunidad de Wikipedia. (26/3/2020) *Python*, Fecha de consulta: 11/4/2020, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Python

[13] Equipo de Techopedia. (28/5/2019) *Python*, Fecha de consulta: 11/4/2020, de Techopedia Sitio web: https://www.techopedia.com/definition/3533/python

[14] David Zomaya. (Marzo de 2020) *Python contra Java*, Fecha de consulta: 15/5/2020, de Udemy Sitio web: https://blog.udemy.com/python-vs-java/?utm_source=adwords&utm_medium=udemyads&utm_campaign=DSA_Catchall_la.EN_cc.ROW&utm_content=deal4584&utm_term=_._ag_88010211481_._ad_437497337007_._kw__._de_c_._dm__._pl__._ti_dsa-449490803887_._li_9049232_._pd__._&matchtype=b&gclid=CjwKCAjwztL2BRATEiwAvnALcqxCHYmZwHHaVnq9h9TT15XCo08EF5XsjlJ_L_qQbmJbdn8dOlikfRoC9wEQAvD_BwE