FACULTAD DE CIENCIAS

Facultad de Ciencias

Proyecto Fin de Carrera

INTELIGENCIA ARTIFICIAL APLICADA AL POKER TEXAS HOLD´EM

(Artificial Intelligence applied to Poker Texas Hold´em)

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Javier González Villa
Director: Marcos Cruz Rodríguez
Codirector: Domingo Gómez Pérez

Julio-2013

Agradecimientos

Espero no dejarme a nadie en el tintero a la hora de nombrar a toda esa gente que me ha ayudado durante estos años, tras los cuales aquí me encuentro.

Primeramente he de agradecer a mi familia, pero en especial a mis padres Constantino Gonzalez y Maria Teresa Villa los cuales siempre me han apoyado y han estado pendiente de mi para que estudie y desarrolle los conocimientos que más me atraen. A mi novia Estela Lopez por todo su apoyo, sin la cual no estaría realizando este preoyecto.

Por otro lado, agredecer a mis amigos, tanto los antiguos como los que he ido haciendo a lo largo de la carrera, por su manera de apoyarme a la hora de hacer este proyecto.

Por último y no menos importante, he de agredecer a Domingo Gomez y a Marcos Cruz, tutores de mi proyecto, la posibilidad que me han dado a la hora de realizarlo y la ayuda que me han proporcionado durante el mismo.

A todos vosotros, muchas gracias.

Resumen

Los juegos de azar y en especial, el Poker Texas Hold´ em representan hoy en día un desafío a la hora de hallar un algoritmo o inteligencia capaz de obtener buenos resultados, como ya se ha conseguido en otros juegos como el ajedrez o las damas.

En lo referente a este proyecto, el objetivo es desarrollar por un lado una herramienta capaz de proporcionar a diferentes usuarios un soporte de simulación que permita estimar la probabilidad de ganar una mano y por otro lado implementar una inteligencia básica que permita al bot interactuar con una aplicación web externa de juego en la modalidad sin límite, y que sea capaz de jugar de forma autónoma.

Todas estas implementaciones que se plantean estarán basadas, a diferencia que otras aplicaciones similares que existen actualmente, en el lenguaje de programación Python y serán capaces de interactuar directamente con una aplicación en Windows, siendo por sus características modulares, capaz de ser extrapolable a otros sistemas.

Adicionalmente y estando al margen de la aplicación, se explican una serie de ampliaciones posibles, capaces de optimizar el rendimiento y mejorar la estrategia de juego seguida por nuestra inteligencia artificial.

Palabras Clave : Python, inteligencia artificial, monte carlo, simulación, Poker texas Hold´ em.

Abstract

Gambling and in particular Texas Hold´ em Poker are a huge challenge for artificial intelligence. Several algorithms are able to compete with the best human players at Chess, checkers and several other games, but this is still not possible in no limit Texas Hold'em Poker.

Here, we develop a tool that is able to calculate the probability of winning a hand through Monte-Carlo simulations. The application can interact with an external web application and manages to play in an autonomous way.

These implementations are based in the Python programming language, unlike other similar applications that currently exist. Our application is able to interact directly with a Windows application, and is modular and can therefore be extrapolated to other systems.

Our poker bot is able to estimate some basic probabilities but its performance can be largely improved. However this implies applying advanced game theory and data mining techniques and is out of the scope of this project.

Keywords: Python, artificial intelligence, monte carlo, simulation, Poker Texas Hold´ em.

Índice

Índice de figuras

Índice de cuadros

1 Introducción

Los juegos representan un interesante reto para la ciencia computacional y la inteligencia artificial. El ajedrez por ejemplo, ha sido ampliamente estudiado en los últimos años, consiguiéndose crear ya en 1997 el ordenador Deep Blue, capaz de competir con buenos resultados frente a los mejores jugadores de ajedrez del mundo.

El Poker en su variedad Texas Hold´ em está centrando la atención de numerosos grupos de investigación como por ejemplo el de la Universidad de Alberta (http://poker.cs.ualberta.ca/) [7]. Otro ejemplo es el Massachusetts Institute of Technology (MIT), donde ofertan un curso de Poker Texas Hold´ em y una competición de bots (http://mitpokerbots.com/).

Hasta ahora se ha conseguido crear bots que pueden competir con buenos resultados en la modalidad de dos jugadores y con límite en las apuestas (http://poker.cs.ualberta.ca/publications/johanson.msc.pdf) [6]. Sin embargo aún no existen bots competitivos en el poker sin límite de apuestas (NLH) ni en las modadildades de más de 2 jugadores. La complejidad del poker Texas Hold´ em se debe a que es un juego de información imperfecta ya que desconocemos las cartas de nuestros adversarios, aunque se pueden estimar con técnicas de minería de datos [8]. Por otra parte, aplicando teoría de juegos se podría generar un bot que jugase de manera óptima frente a otro jugador óptimo. Esta opción sin embargo no permite explotar al máximo a un jugador débil.

Una vez estimado el rango de cartas que asignamos a nuestros adversarios, la probabilidad de ganar puede calcularse mediante simulaciones de Monte Carlo o bien de manera exacta, aunque ésta opción puede ser computacionalmente costosa.

Sin embargo, estimar correctamente la probabilidad de ganar una mano no es suficiente para jugar de manera óptima ya que es necesario determinar la cantidad óptima de fichas que hay que apostar. Esto se puede estimar maximizando el valor esperado de la variable beneficio neto.

Por tanto, vemos que crear un bot de Texas Hold´ em competitivo requiere aplicar Minería de Datos, Teoría de Juegos y estadística. La aplicación de todos estos recursos combinada nos da un sistema que podría ser extrapolable a otros ámbitos de automatización, en los cuales se requiera tomar decisiones en base a datos.

En este trabajo no pretendemos crear un bot competitivo ya que es una tarea extremadamente compleja que llevaría varios años de trabajo, sino programar un bot capaz de interactuar con interfaces de aplicaciones, jugando de manera continuada y estable en la modalidad de NLH con cualquier número de jugadores. Estimaremos las probabilidades de éxito mediante simulaciones de Monte Carlo [2], asumiendo un rango de cartas aleatorio para nuestros adversarios.

2 Conceptos previos: Poker Texas Hold´ em

Texas Hold´ em es el termino que se utiliza para determinar un estilo de juego diferente del tradicional Poker. Este estilo de juego es de los más jugados en todo el mundo y en concreto su modalidad sin límite, la cual es más compleja debido a que no se establecen límites máximos ni en las apuestas ni en el bote. El no fijar límites en cuanto a lo que parámetro de apuesta se refiere complica las cosas a la hora de calcular la cantidad óptima que hay que apostar para maximizar el valor esperado.

Centrándonos en el estilo de juego de esta modalidad, tenemos que decir que pueden participar de 2 hasta 10 jugadores en diferentes tipos de mesa, a los cuales se les impone un sistema de apuesta mínimo rotatorio para obligar a los jugadores a obtener beneficios u obligarlos a tener que retirarse. A dichas apuestas se les denomina blind o ciegas debido a que es una apuesta mínima obligatoria que han de realizarse antes de repartirse las cartas. Siempre hay dos jugadores obligados a pagar las ciegas: la ciega pequeña y la ciega grande, que será el doble de la pequeña, estas apuestas rotan en función de la posición del repartidos en la mesa, siendo la ciega pequeña la que esta a la izquierda del repartidor y la ciega grande la segunda posición a la izquierda del repartidor.

En cuanto a los elementos que intervienen en el juego tenemos que constan de 52 cartas que forman 4 palos diferentes (Diamantes, Corazones, Treboles y Picas), cada cual contiene 13 cartas enumeradas consecutivamente (AS,2,3,4,5,6,7,8,9,10, J,Q,K).

El funcionamiento del juego es complejo y se divide en cinco etapas consecutivas, que representan cinco escenarios totalmente diferentes en los cuales tendremos diferentes número de jugadores así como de cartas centrales levantadas y diferentes apuestas, lo que nos condiciona a la hora de tomar decisiones:

- **Pre-Flop:** Etapa inicial del juego en la cual los jugadores tan solo conocen sus propias cartas (2 cartas) y en la cual la apuesta inicial es las ciegas por defecto. En esta fase se realiza una ronda de apuestas la cual comienza por el jugadores la izquierda de la ciega grande y continua en la dirección de las agujas del reloj, pudiendo en caso de no tener unas cartas aceptables, retirarse del juego hasta que el resto de jugadores acaben dicha ronda y se reparta una nueva serie de cartas. Estas apuestas continuarán hasta que quede una cantidad de jugadores menor o igual a la inicial que hayan apostado la misma cantidad.

- **Flop:** Etapa posterior al Pre-Flop en la cual se destapan tres cartas centrales de las cinto totales comunitarias, con las que los jugadores han de hacer sus posibles combinaciones máximas para obtener una jugada superior a la del resto de adversarios. Esta fase está seguida de una o más rondas de apuestas hasta que se igualen las apuestas como en la fase anterior.

- **Turn:** Fase en la cual se descubre una carta comunitaria más, y que también va seguida de rondas de apuestas al igual que las etapas anteriores.

- **River:** Fase en la cual se descubre la última carta y por la cual ya podemos determinar la jugada máxima propia. Esta fase también contiene rondas de apuestas hasta que solo un jugadores continúe apostando o hasta que igualen las apuestas de tal manera que los jugadores han de enseñar sus cartas para determinar el resultado.

- **Show Down:** Etapa final en la cual se muestran las cartas y se calculan las jugadas máximas de los jugadores restantes y se determina el ganador.

Estas son las fases del juego, que representan la estructura más simple de la posible máquina de estados de actuación, sin tener en cuenta otros factores del juego como son las combinaciones de las jugadas y su valor, así como de las cartas restantes y de las posibles combinaciones que superen o igualen nuestra jugada máxima. Dichas jugadas y su valor se muestran a continuación:

1. **Escalera Real:** Cinco cartas consecutivas del mismo palo, comenzando en AS y finalizando en 10.
2. **Escalera de Color:** Cinco cartas consecutivas del mismo palo, sin carta mínima ni máxima. En esta jugada, tomaremos el AS con dos valores a la hora de realizarla, contándolo como máximo y mínimo valor a la vez, por lo que podrá ser el principio de la escalera mínima o el final de la escalera de máximo valor.
3. **Poker:** Cuatro cartas del mismo valor y distintos palos.
4. **Full:** Conjunto de un trío y pareja de cartas con el mismo valor, independientemente del palo.
5. **Color:** Cinco cartas del mismo palo, sin necesidad de ninguna otra relación en cuanto a la figura.
6. **Escalera:** Compuesta por cinco cartas consecutivas, independientemente del palo o de la carta máxima o mínima.
7. **Trío:** Tres cartas con el mismo valor.
8. **Dobles Parejas:** Conjunto de dos parejas, cuyas cartas tienen el mismo valor.
9. **Pareja:** Dos cartas con el mismo valor, independientemente del palo.
10. **Carta más alta:** En caso de no conseguir ninguna combinación de las anteriores, la carta más alta dirime el resultado.

En cuanto a lo que nociones básicas sobre este tipo de juego de azar se refiere, estos son los conocimientos básicos para entender el mecanismo de funcionamiento de la aplicación. Una vez conocemos las nociones básicas, hemos de entender que los jugadores pueden ganar la mano de diversas formas. La primera de ella es si el jugadores continua apostando, en cualquiera de las fases del juego y los demás rivales deciden retirarse y no continuar apostando, este ganará independientemente de la jugada que tenga. La segunda forma de ganar, es una vez llegados a la última fase (Showdown) dos o más jugadores y habiendo estos apostado la misma cantidad, se calcula la jugada máxima de las antes explicadas, en función de las dos cartas propias de cada jugador, realizando combinaciones junto con las cinco cartas ya conocidas centrales, las cuales son compartidas por todos los jugadores.

3 Esquema general de la aplicación

En este apartado se tratan cuestiones básicas del funcionamiento externo de la aplicación, así como las funcionalidades que esta nos proporciona y un esquema simple de la composición de los diferentes módulos y de las propiedades específicas que les hacen divisibles y remplazables.

3.1 Funcionalidades generales

Las principales funcionalidades de esta aplicación son dos y son las de proporcionar una interfaz al usuario experto con la cual poder interactuar como se muestra en la Figura 1 y calcular probabilidades de ganar una mano seleccionando el número de jugadores adversarios. También se pueden seleccionar las cartas del flop, por tanto el usuario podrá simular situaciones en tiempo real o proponer casos aleatorios sobre los cuales tenga interés para poder trazar un sistema de juego en diferentes situaciones.

Por otro lado la segunda funcionalidad es más compleja que la anterior pero conserva la misma dinámica, a pesar de que esta segunda funcionalidad está más orientada a usuarios que lo que quieran es automatizar su juego, dejando así que el bot se encargue de determinar la estrategia de juego como se muestra en la Figura 1 evaluando y actuando sobre diferentes escenarios que se irá encontrando a medida que cambia el contexto de juego. Esta funcionalidad tiene unos requisitos temporales muy estrictos debido a que las aplicaciones con las que va a interactuar son totalmente externas a nuestra aplicación y fijan estos requisitos para evitar problemas en el flujo del juego.

3.2 Funcionalidades específicas de cada módulo

La aplicación se subdivide en cuatro submódulos independientes entre sí, de tal manera que si en un momento dado se requiere del reemplazo de uno de los módulos por la necesidad de implementar un algoritmo mejor o por la necesidad

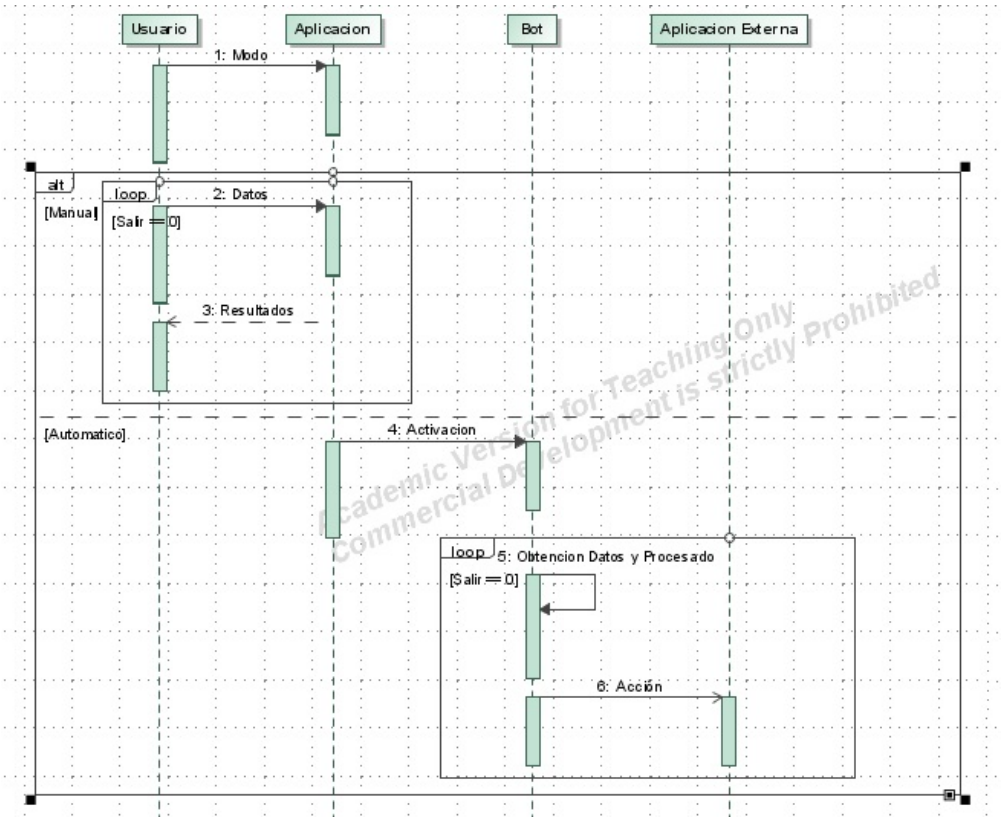

Figure 1: Diagrama de flujo de la aplicación.

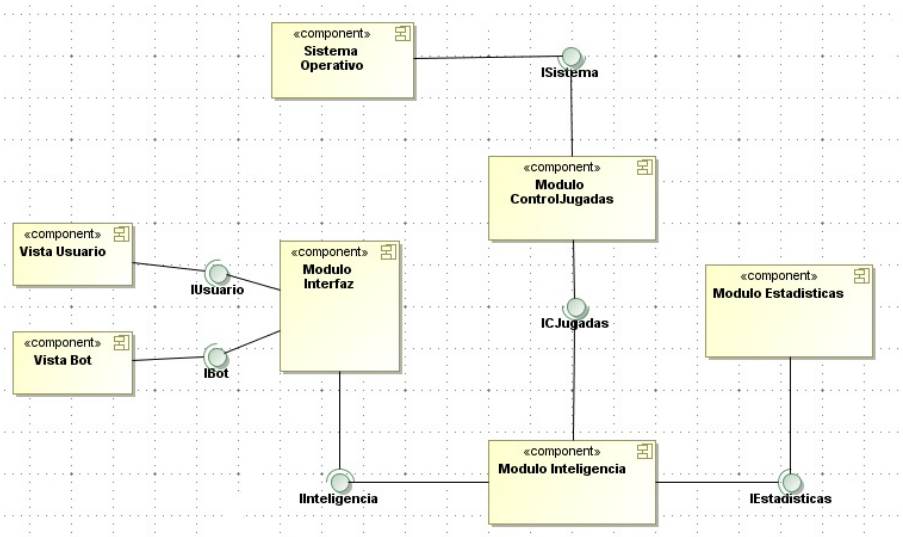

Figure 2: Esquema de interactuación entre módulos.

de adaptación de la aplicación a ser ejecutada sobre otro equipo con otras características diferentes, esto no sea un problema para el resto de la aplicación.

El módulo más externo que nos encontramos es el módulo de la interfaz que es el encargado de proporcionar una visión general al usuario sobre lo que esta sucediendo. Este módulo es el encargado de controlar las entradas del usuario y de proporcionarle una guía acerca de como ser usado. También basa los resultados de sus salidas en llamadas al resto de módulos.

El segundo módulo más externo y con funcionalidades de análisis e interactuación es el módulo de control de jugadas, el cual es el encargado de interactuar, no tanto con el usuario, sino con el sistema operativo y con los periféricos para proporcionar una automatización de tareas, como por ejemplo el manejo del teclado o ratón sin necesidad de que el usuario este prestando atención a la aplicación. Este módulo al igual que todos los demás puede ser sustituido y adaptado para el sistema con el que se requiera la interactuación.

El tercer módulo y ya más especifico para esta aplicación, pero que como ya hemos dicho antes puede ser reemplazado para que la aplicación realice cálculos estadísticos basados en algoritmos diferentes, es el módulo estadístico el cual tiene diferentes funcionalidades como simulación y obtención del valor de las jugadas.

El último módulo que compone la aplicación es el módulo llamado inteligencia y es la gestión central del resto de módulos y recursos encargado de dotar de inteligencia al bot y que pueda funcionar de manera autónoma.

4 Especificaciones de los módulos

Tras ver el funcionamiento básico de los diferentes módulos que componen la aplicación así como su relación a la hora de interactuar, ahora vamos a tratar más en detalle y de manera independiente los diferentes módulos. En referencia a su implementación, todos estos módulos han sido implementados en lenguaje Python sobre su versión 2.6, la cual es bastante estable y nos proporciona las funcionalidades de cálculo requeridas para proporcionar resultados en el tiempo esperado así como otras librerías necesarias para interactuar con el sistema y realizar una interfaz adecuada a nuestros requisitos.

4.1 Módulo Interfaz

Como ya dijimos, este módulo es el que interactua de manera directa con el usuario a la hora de elegir la funcionalidad que el usuario quiera ejecutar. Desde un punto de vista de usuario el módulo proporciona una interfaz simple principal al usuario en la cual este puede seleccionar la opción que desee, como se puede ver en la Figura 3. Tras esto la interfaz proporciona diferentes subinterfaces para el cálculo de estadísticas en función de diferentes parámetros especificados por el usuarios como se ve en la Figura 4 y en la Figura 5.

Como podemos ver, de las interfaces antes citadas, la automática es la más simple y solo necesitamos abrir una mesa sobre la que jugar y seleccionar en el lanzador la opción automática para la aplicación con la que queremos interactuar. A pesar de como se ve en la imagen, esta interfaz nos permite interactuar de manera automática con diferentes aplicaciones externas, actualmente solo esta implementada para la aplicación de PokerStars.

Fijándonos ahora en la interfaz manual podemos observar que contiene diferentes campos con diversos formatos de entrada de datos por parte del usuario. Estos campos y sus restricciones de formato se explican a continuación de manera detallada:

- **Cartas Propias:** Entrada de datos más importante y crucial, sin la cual no podríamos simular. Esta entrada de datos esta restringida a un único valor por campo que tenga un formato igual a un número del 1 al 13 para indicar la figura y una letra a continuación (C,T,D,P,c,t,d,p) para indicar el palo de la carta. Otra restricción es que los valores de ambos campos de esta sección han de ser diferentes, para no repetir las cartas. Este formato al igual que en el resto de campos será comprobado y en caso de no cumplir los requisitos antes mencionados, se indicará al usuario mediante un campo de escritura informativa, el fallo que ha cometido. Posteriormente se analizará el valor del dato introducido, transformándolo a un entero con el que ser capaces de trabajar, todo esto sin que el usuario necesite saberlo.

- **Jugadores:** En este campo se podrá introducir datos de cualquier longitud, divididos por comas para indicar el nombre de los jugadores o simplemente una referencia para denotar el número de rivales. En el caso de estar vacío o incumplir el formato especificado, al igual que en el resto de parámetros se indicará al usuario el error que esta cometiendo.

- **Cartas Centrales:** Subsección de entrada de datos, la cual no es necesario cumplimentar forzosamente a la hora de simular, ya que en caso de que no introduzcamos ningún valor, el programa se encargara de simular aleatoriamente las cartas centrales. Este campo es el más restrictivo debido a la variedad de posibles datos de entrada. La primera reestricción del campo es que los valores introducidos cumplan el mismo formato que se especifica al introducir las cartas propias, con la peculiaridad de que este campo al igual que el de jugadores permite múltiples valores divididos por comas. Dichos valores no pueden ser superiores a cinco debido que la reestricción propia del juego de que el número máximo de cartas centrales es cinco. Una vez comprobadas estas reestricciones propias del campo se ha de comprobar que cada valor introducido, no es repetido. Esta comprobación no solo se hace en función del campo, sino que también se comprueba con las cartas introducidas en el campo Cartas Propias. Al igual que el resto de campos, de cometer un error este nos indicaría cual es.

- **Manos:** Indica el número de veces que queremos simular las diferentes cartas de los rivales antes enumerados, para dar más o menos fiabilidad a la simulación. Este campo al igual que el campo simulaciones solo acepta números enteros.

- **Campo informativo:** Barra que se encuentra encima del tablero de juego y en la cual la aplicación nos irá mostrando diferentes mensajes de error en función de el error cometido, ya sea de formato o por falta de datos necesarios para comenzar la simulación.

- **Tablero de Juego:** Dicho componente no permite la entrada de datos, y es meramente informativo, ya que será donde se nos muestren las probabilidades de ganar la mano, tanto para nosotros como para los rivales introducidos. Este campo se refresca cada vez que damos al botón aceptar, variando las salidas gráficas en función de los valores de los campos antes citados.

La implementación de este módulo a parte de ser realizada en Python 2.6 [12] requiere de una serie de librerías externas que nos proporcionan la generación de dicha interfaz que son Tkinter [11] para la creación de la interfaz, PIL(Python Imaging Library) [9], tkFont nos proporciona las fuentes necesarias y otras más triviales como time, math o string para operaciones sencillas.

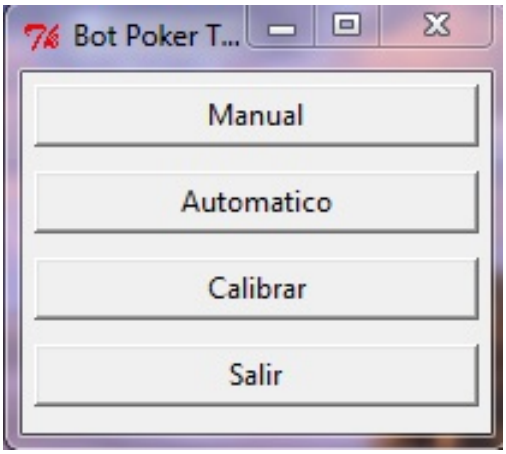

Figure 3: Interfaz principal de la aplicación.

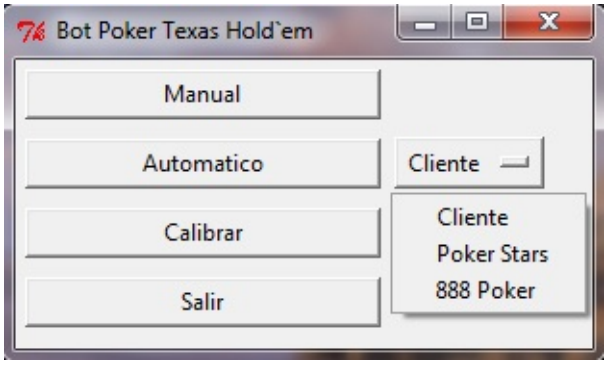

Figure 4: Subinterfaz automática de la aplicación.

Este módulo a parte de proporcionar la interfaz, es el encargado de realizar llamadas a diferentes funciones del módulo Inteligencia para obtener resultados que mostrar al usuario o para activar funciones de automatización contenidas en dicho módulo.

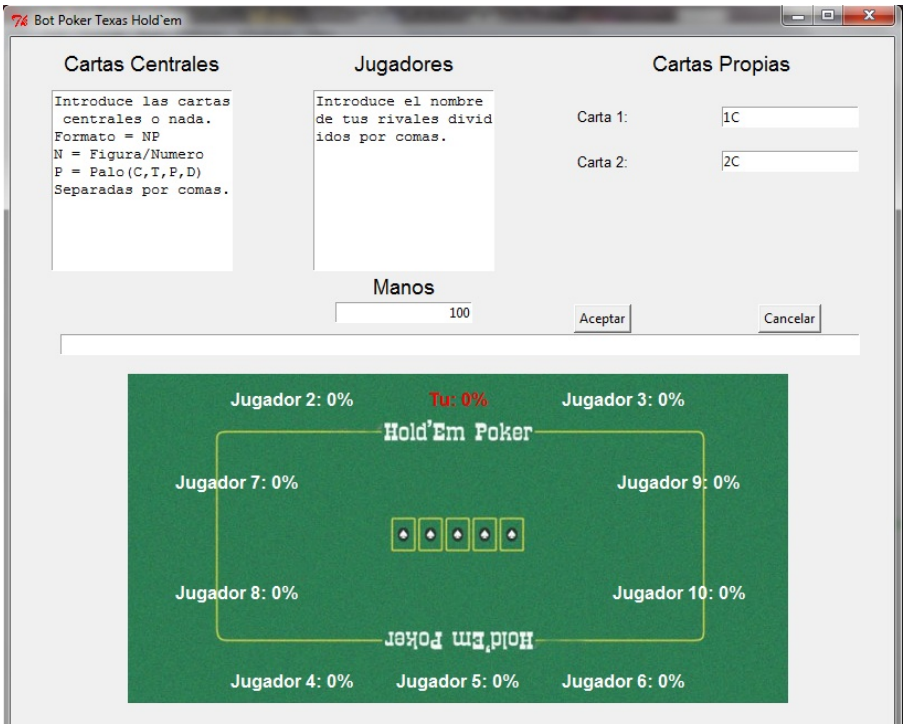

Figure 5: Subinterfaz manual de la aplicación.

4.2 Módulo Estadísticas

Este módulo junto con el módulo de inteligencia forman la parte central de la aplicación, debido a que este módulo es el encargado de simular así como de determinar la mejor jugada en función de combinaciones de las cartas proporcionadas. Este módulo es sobre el cual el módulo inteligencia obtendrá los datos necesarios para la toma de decisiones del bot.

4.2.1 Algoritmo de identificación de jugadas

Este es de los dos algoritmos contenidos en este módulo el más básico en cuanto a necesidad, debido a que este algoritmo es el que calcula en función de las cartas dadas (cartas del jugador y cartas centrales) la mejor jugada en función de un sistema de descartes comienza comprobando si con las cartas disponibles podemos formar la mejor jugada posible (escalera de color) y en caso de no tenerla pasa a la segunda mejor jugada (poker) y así sucesivamente hasta llegar a carta alta si fuese necesario.

Al ser un sistema de descartes secuencial obtenemos una alta probabilidad de que una misma jugada tenga que comprobarse para más de un caso debido a que se comienza con las jugadas más altas, las cuales tienen un indice menor de aparición lo que hace que tengamos que comprobar también las jugadas inferiores o de menor valor las cuales tienen una probabilidad de aparición mayor. El cálculo de estas probabilidades es diferente para cada tipo de jugada,

debido a los requisitos que esta nos impone para ser formada. A continuación se muestra una tabla con las probabilidades de realizar dicha combinación. Como vemos en la Tabla 1 las probabilidades están calculadas en función de las posibles combinaciones así como de las combinaciones totales. El número de combinaciones totales se calcula a continuación y es a partir del cual obtenemos las probabilidades de la Tabla 1.

$$C_{52,5} = \binom{52}{5} = \frac{52 \cdot 51 \cdot 50 \cdot 49 \cdot 48}{5!} = 2598960 \; combinaciones$$

El resto de cálculos en función de cada jugada, son dependientes de la jugada en sí, debido a que cada combinación de cartas tiene una probabilidad diferente. Dichas probabilidades mostradas en la Tabla 1 se explican más detalladamente a continuación.

1. **Escalera Real:** Con cada palo de la baraja se puede formar una única escalera real y al haber cuatro palos solo tenemos cuatro posibles combinaciones.

2. **Escalera de Color:** Con cada palo de la baraja podemos formar diez posibles escaleras, a las cuales hemos de descontar las cuatro propias de la Escalera Real, con lo que nos dá el número de combinaciones calculado.

3. **Poker:** Las posibles combinaciones de esta jugada son dependientes tanto de las posibles trece combinaciones de cuatro cartas iguales, como de las cuarenta y ocho cartas posibles en el hueco restante, por lo cual su producto nos dará las posibles combinaciones ya calculadas.

4. **Full:** Este caso es similar al del poker pero con la diferencia de que tendremos por un lado las doce posibles combinaciones para formar la pareja, multiplicado por la probabilidad de que aparezca una de esas doce cartas de diferente palo y posteriormente las posibles combinaciones de tríos que tenemos que una vez calculadas la de las parejas, tenemos que solo serán posibles cuatro trios por pareja. Debido a que tenemos trece cartas posibles para formar el trío el producto de todos los factores previos nos dá el resultado calculado.

5. **Color:** Las posibles combinaciones para conseguir color son dependientes del número de palos que es igual a cuatro y del número de posibles combinaciones de trece cartas del mismo palo de cinco en cinco. El resultado del producto de ambos es el número de combinaciones posibles para dicha jugada.

6. **Escalera:** En el caso de la escalera es un poco diferente debido a que si tomamos una carta como comienzo, tenemos diez comienzos posibles y posteriormente a partir de dicha carta tenemos para cada carta consecutiva cuatro posibles, una por palo, y de esta manera proseguimos hasta completar las cinco cartas, y dándonos así las posibles combinaciones.

Prioridad	Jugada	Combinaciones	Probabilidad
1	Escalera Real	4	1,539 e-6
2	Escalera de Color	36	1,385 e-5
3	Poker	624	2,4 e-4
4	Full	3744	1,44 e-3
5	Color	5108	1,965 e-3
6	Escalera	10200	3,924 e-3
7	Trío	54912	2,112 e-2
8	Dobles Parejas	123552	4,753 e-2
9	Pareja	1098240	0,422
10	Carta Alta	1302540	0,5011

Table 1: Probabilidades de combinaciones de cartas

7. **Trío:** En este caso de manera similar al full, tenemos que para cada una de las trece cartas posibles, cuatro tríos. De las dos posiciones vacias que no forman trío restantes tenemos que obtener dos cartas de las cuarenta y ocho restantes, dando así que el producto de ambos factores nos da las posibles combinaciones.

8. **Dobles Parejas:** Para esta jugada primeramente hemos de calcular las combinaciones posibles de cartas en parejas de las trece cartas posibles y posteriormente calcular las combinaciones de parejas de diferente figura a la ya creada para posteriormente calcular las posibles combinaciones de cartas para la carta restante que ha de ser diferente de las parejas ya formadas.

9. **Pareja:** La pareja representa una manera sencilla de cálculo de sus posibles combinaciones teniendo en cuenta para cada carta de las trece posibles, las posibles parejas que se pueden hacer. Una vez calculado esto, solo hemos de calcular las posibles combinaciones de colocar tres cartas a partir de las cartas restantes.

10. **Carta Alta:** Una vez conocemos todas las posibles combinaciones anteriores y el total de las mismas que se pueden hacer con las cartas de la baraja solo hemos de restar ambas cifras para obtener el número de posibles combinaciones de la jugada de carta más alta.

Para solventar la repetición inecesaria de la busqueda del valor máximo teniendo que pasar por todas las fases de búsqueda hemos, analizado cada jugada independientemente y de esta manera hemos podido concluir que casi todas las jugadas tienen unas condiciones diferentes a las otras y que por lo tanto se pueden comprobar mediante una única sentencia que nos permite no tener que entrar en todos los casos de búsqueda, si no tan solo en aquellos que cumplan con los requisitos. Estos requisitos se refieren a la relación de las cartas entre sí y con las demás, obteniendo así las relaciones por separado de cada carta con

Jugada	Requisito Palo	Requisito Figura
Escalera Real	maxp ≥ 5	maxf ≤ 3
Escalera de Color	maxp ≥ 5	maxf ≤ 3
Poker		maxf == 4
Full		maxf == 3 & element == 2
Color	maxp ≥ 5	
Escalera		maxf ≤ 3
Trío		maxf == 3
Dobles Parejas		maxf == 2 & element == 2
Pareja		maxf == 2
Carta Alta		

Table 2: Requisitos relacionales para creación de jugadas

las demás en función de la figura y del palo como se representa en la Tabla 2, la cual contiene los parámetros maxf y maxp que representan, en el primer caso la cantidad máxima de cartas con la misma figura, y en el segundo caso el número de cartas máximo con el mismo palo. Cada número de requisito representará en el caso de la relación con el palo la cantidad de cartas de las que estamos tratando que son del mismo palo a la carta analizada y en cuanto a la relación de las figuras contendrá para cada carta el número de cartas con igual figura a dicha carta. Una vez obtenidas dichas relaciones, sacamos el máximo de ambas que es el que nos servirá para filtrar la búsqueda. Tras el análisis de la relación entre las cartas y de los requisitos de las jugadas, especificados en la Tabla 2, tenemos que cada caso de búsqueda es totalmente independiente del resto, por lo que nuestro algoritmo será más óptimo en cuanto a tiempos se refiere.

Una vez que estos requisitos son comprobados y se sabe de manera precisa la posible o posibles jugadas que se pueden hacer con dichas cartas, se comprueba de manera más exhaustiva que cumplan carta a carta los requisitos necesarios para que sea la mejor jugada. Una vez se a comprobado que esa jugada es la mejor que podemos conseguir con la combinación de las diferentes cartas, a esta jugada se le asigna un valor numérico en función de las cartas máximas de la jugada que nos permitirá más adelante distinguir cada jugada y saber cual es de mayor valor. La asignación de este valor a la jugada es necesario debido a que cada jugada tiene una estructura diferente y se han de estandarizar para su posterior análisis en el algoritmo de simulación.

$$Valor = 10^4 \cdot ValorJugada + 10^2 \cdot CartaJuego + CartaMaxima$$

Esta fórmula, no sólo nos permite obtener un valor estándar con el que poder comparar diversas jugadas para obtener cual es mayor, si no que también nos permite obtener mediante operaciones sencillas el valor de la jugada que representa el tipo de jugada que hemos hecho, la carta de juego que es la carta máxima dentro de la jugada que tenemos y la carta máxima externa a la jugada

que nos permite dirimir en caso de obtener la misma jugada que el oponente.

4.2.2 Algoritmo de generación y simulación

En lo referente a la simulación, el algoritmo utilizado está basado en la generación de manera aleatoria de las cartas de los oponentes así como las cartas centrales, el número necesario de veces hasta obtener una probabilidad de ganar aceptable, para lo cual necesitamos un gran número de simulaciones. Estas simulaciones han de ser rápidas debido a que el algoritmo será ejecutado en tiempo real, con un límite de tiempo específico para realizar la jugada.

Primeramente hemos de analizar el número de simulaciones necesarias para obtener un valor adecuado y fiable que nos permita determinar el riesgo con el que estamos tratando. Para ello se realizará un análisis de fiabilidad a partir del cual determinamos el punto en el que el número de simulaciones no influye significativamente en la probabilidad de ganar. En la Figura 6 se muestran los resultados del análisis del número de simulaciones. En esta gráfica podemos ver la raíz del error cuadrático medio de la probabilidad de ganar la mano expresada en porcentaje. Este error nos da una visión global del fallo que podemos cometer en función de las simulaciones que especifiquemos. Dicho error fue calculado como la desviación de la probabilidad real, calculada mediante una herramienta externa. Esta herramienta es Equilab y se utiliza para el cálculo de probabilidades en poker(http://www.pokerstrategy.com/software/10/).

Una vez conocemos el error que cometemos con respecto a la cifra real, tenemos que al no ser necesaria una precisión por debajo de las décimas y teniendo en cuenta que los requisitos impuestos por la aplicación con la que estamos interactuando o por el tiempo de espera en el caso de utilizar la aplicación con su funcionalidad manual, hemos de concretar en el caso del bot que interactua en tiempo real, el número de simulaciones máximo posible que no infrinja los tiempos requeridos por la aplicación externa y que a su vez nos permita obtener una mayor fiabilidad. Este número de simulaciones se obtiene a partir de el análisis de las Figuras 7, 8 donde se puede ver la evolución del tiempo requerido por nuestra aplicación en función del número de simulaciones y los límites que impone la aplicación externa, los cuales son de ocho y dieciséis segundos en función de diferentes límites de tiempo. El límite crítico es el que no podremos sobrepasar.

Como podemos ver en la Figura 7, los tiempos bajo condiciones límite que representan los tiempos con un número máximo de jugadores por lo que nuestro algoritmo ha de hacer simulaciones para más rivales, nos condiciona a hacer un número máximo de 70000 simulaciones lo que implica un error medio de 0,184. Este número de simulaciones máximas como podemos comprobar en la Figura 8 varia en función de las condiciones con las que estemos operando, en dicha gráfica se muestra las condiciones mínimas las cuales nos permiten realizar un número mucho mayor de simulaciones. Este número de simulaciones por tanto hemos de concluir que es directamente dependiente del número de

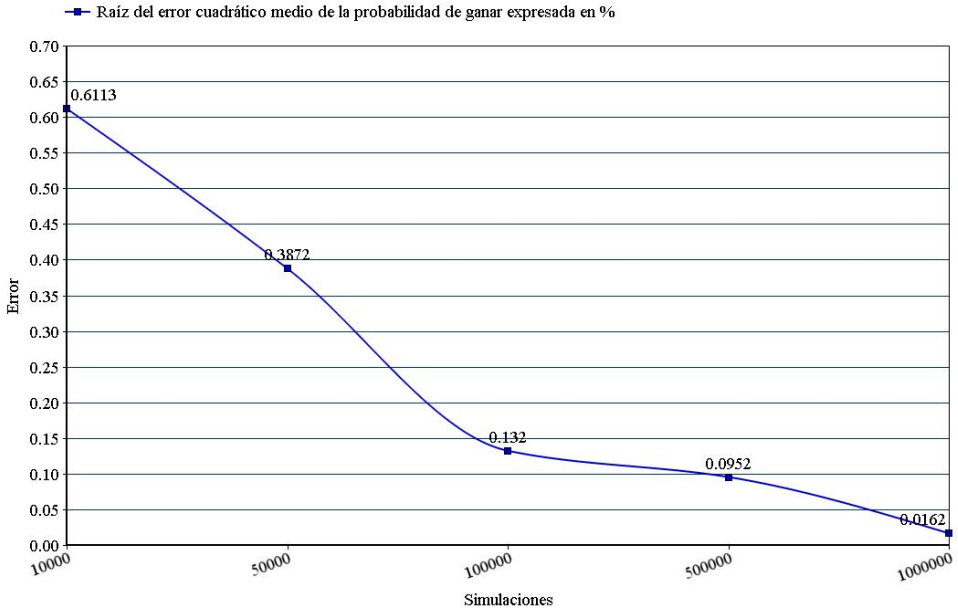

Figure 6: Gráfica de evolución de error de calculo.

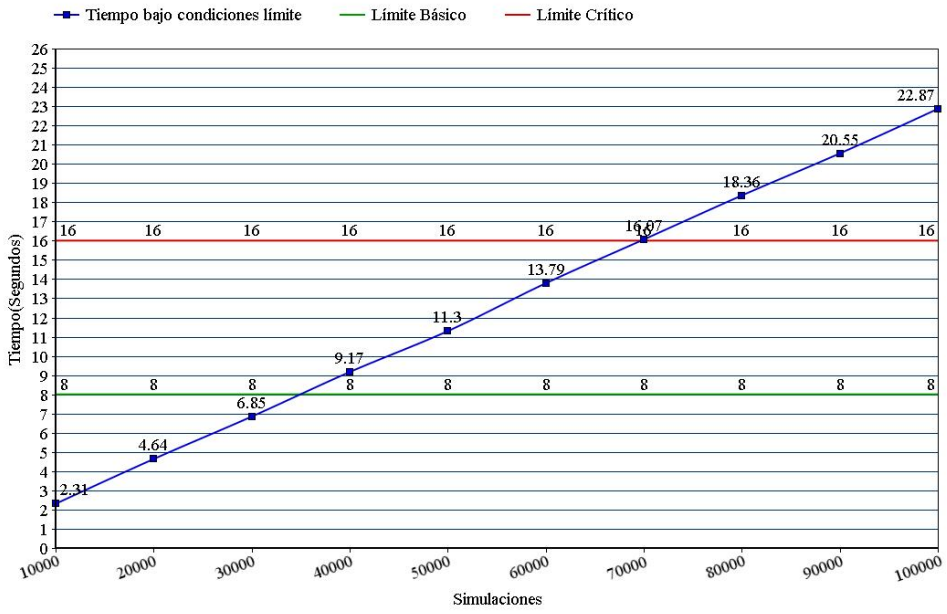

Figure 7: Gráfica de evolución tiempos de ejecución bajo condiciones límite.

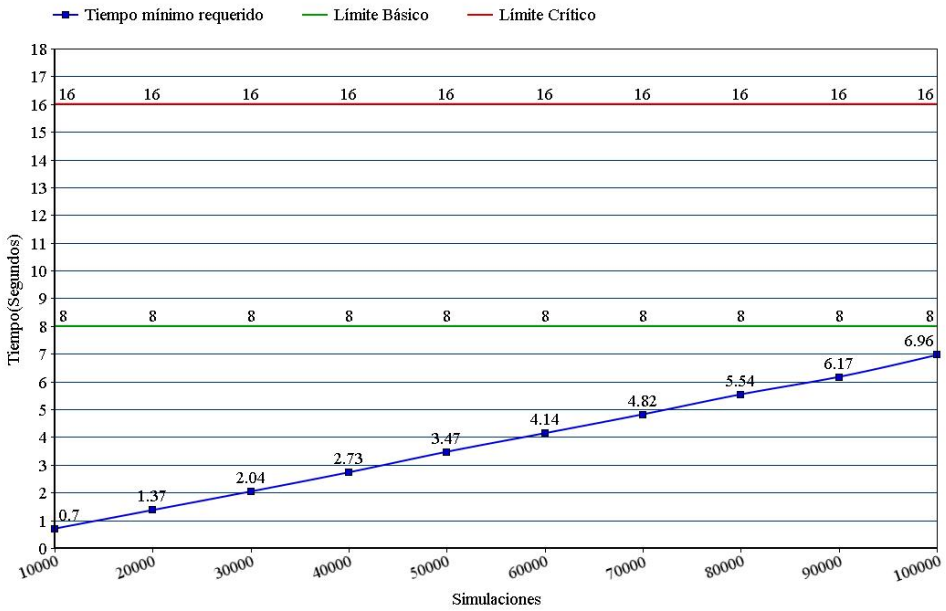

Figure 8: Gráfica de evolución tiempos de ejecución bajo condiciones mínimas.

jugadores. Con lo que tenemos que a menor número de jugadores, mayor número de simulaciones y por tanto un indice de error menor.

Una vez nuestro algoritmo conoce ya sea de mano del usuario o por parte del análisis de tiempos presentado anteriormente, el número de simulaciones, este algoritmo lo único que ha de hacer es simular un número de veces igual a el número de simulaciones especificadas, diferentes manos a jugadores y posteriormente para cada mano ha de simular las cartas centrales restantes no especificadas. De esta manera conseguimos simular por igual manos de jugadores y manos centrales de tal manera que simulamos de manera equitativa posibles jugadas que nos vamos a encontrar, para de esta manera obtener la probabilidad de ganar.

La simulación de estas cartas antes citadas es totalmente aleatoria y sigue un patrón de generación de números aleatorios discretos, en el intervalo de cartas posibles es desde 1 hasta 52, representando estos valores discretos (1,2,3,..,52) cada una de las cartas de la baraja propia del juego de Poker Texas Hold´ em. Dicho intervalo se va acotando progresivamente a medida que vamos simulando las diferentes manos de cada jugador, para obtener de esta manera una simulación más realista, con lo que finalmente nos encontraremos con un intervalo mas acotado que puede contener desde 32 hasta 48 cartas, en función del número de jugadores, a la hora de simular las cartas centrales.

Una vez tenemos las cartas de cada jugador y el intervalo de cartas restantes con dichas cartas que no han sido seleccionadas previamente en la simulación de manos, hemos de simular las cartas centrales en función en este caso, de los datos obtenidos de las cartas centrales. Esta simulación central podrá ser de 0 a 5 elementos en función de la fase de juego en la que nos encontremos, y a medida que tengamos más cartas conocidas iremos acotando más el intervalo de simulación de cartas centrales y por tanto obteniendo resultados más próximos a la realidad.

Tras la simulación de una mano o de las cartas centrales, se restaura el estado previo para volver a encontrarnos con la situación anterior ya sea de cartas de jugadores o de intervalos de simulación y de esta manera poder seguir simulando manos y cartas centrales sin repetir cartas que ya han aparecido previamente.

4.3 Módulo Control de Jugadas

El módulo de control de jugadas representa la interacción directa del bot con el sistema para la obtención de datos y para la implementación de funciones más complejas que requieran de estas llamadas. La principal función de este módulo es hacer que la aplicación interactue con el sistema y obtener independencia de tal manera que podamos portarlo a otro equipo totalmente diferente sin tener que cambiar ningún otro módulo. Este módulo esta basado una serie de librerías propias de Python y de Windows las cuales nos permiten realizar las operaciones necesarias.

1. **Python Imaging Library (PIL):** Permite mediante una serie de clases realizar operaciones con imágenes en Python. Esta librería nos permite procesar diferentes formatos de imágenes [9].

    a. **Image:** Módulo contenido en la librería PIL que nos permite almacenar objetos de tipo imagen con los que posteriormente podremos realizar operaciones. También nos permite importar imágenes de diferentes formatos a parte de los por defecto.

    b. **ImageTk:** Módulo contenido en la librería PIL que nos permite realizar operaciones básicas de formateo sobre las imágenes para ser procesadas posteriormente.

    c. **ImageGrab:** Módulo contenido en la librería PIL con el cual podemos realizar capturas de pantalla en Python que posteriormente podremos formatear.

    d. **ImageOps:** Módulo perteneciente a la librería PIL que nos da una serie de funciones con las que poder operar sobre las imágenes para evaluarlas, de tal manera que podamos comparar imágenes o realizar evaluaciones de estamentos simples.

2. **os:** Librería en Python que nos permite la interactuación con el sistema, dotando a nuestra aplicación de operaciones de lectura y escritura de ficheros del sistema así como otras operaciones como el cambio de caminos y comparación y búsquedas de ficheros. También nos proporciona la ejecución de comandos del sistema operativo.

3. **win32api:** Módulo que nos permite interactuar con el sistema de manera activa, permitiéndonos realizar operaciones de ejecución de comandos, pulsado de teclas y control del cursor para poder acceder a la aplicación con la que queremos interactuar.

4. **win32con:** Módulo que contiene los diferentes valores necesarios para indicar a la librería win32api las operaciones que tiene que realizar, como por ejemplo valor del tipo de evento o obtención de valores de las diferentes comandos que podemos realizar.

4.3.1 Elementos de interactuación con el sistema

Una vez que conocemos los módulos y librerías externas que nos proporcionan métodos con los cuales interactuar con el sistema, ahora hemos de determinar las funcionalidades implementadas por nosotros, las cuales son necesarias para que el bot pueda ejecutarse de manera correcta abarcando todas las posibles operaciones que realizará. Estas funcionalidades propias son simples debido a que nos interesa que sean totalmente independientes del resto de la aplicación, puesto que si fuesen más complejas, no tendríamos esas capacidades sino que tendríamos que reimplementar más módulos del sistema o este mismo módulo, de manera más compleja.

A continuación se muestra de manera más detallada las diferentes funciones implementadas y se explica la necesidad de las mismas para futuros procesos ejecutados en otros módulos:

1. **Pulsado de ratón:** Funcionalidad que abarca el pulsado de todos los elementos del ratón de tal manera que podamos interactuar con la aplicación puesto que dicha aplicación no proporciona ninguna interfaz y solo permite interactuar con ella a partir de periféricos como si de un usuario se tratara.

2. **Ejecución de comandos del sistema:** Esta operación es necesaria pero no con demasiada prioridad debido a que la ejecución de comandos, solo la vamos a utilizar para colocación de interfaces y renombrado de ficheros, cosa que podemos hacer manualmente antes de ejecutar la apliacación.

3. **Operaciones con ventanas:** Implica funcionalidades como cambio de ventana o maximizar y minimizar, que nos serán posteriormente útiles para determinar la posición del lanzador de la aplicación con la interactuamos así como para manejarla por el escritorio a nuestro antojo hasta que este posicionada donde nos interesa.

4. **Operaciones con ficheros:** Nos permite interactuar con el sistema de ficheros para poder por ejemplo abrir imágenes almacenadas con anterioridad las cuales nos servirán de referencia para el análisis de los datos que vamos obteniendo del sistema en tiempo real.

4.3.2 Métodos de obtención de datos

Por otro lado tenemos una serie de funcionalidades proporcionadas por este módulo con las cuales podemos obtener datos del sistema. Mediante la obtención de estos datos conseguimos la información necesaria para poder ejecutar el bot de manera que sepa el estado en el que se encuentra y de qué manera ha de reaccionar en función del contexto que no es más que el procesamiento de estos datos para que puedan ser analizados de manera más sencilla por el bot.

Los métodos comprendidos para la obtención de datos son los siguientes:

1. **Obtención de la posición del cursor:** Esta funcionalidad es de las más necesarias debido a que la aplicación con la que queremos interactuar no nos proporciona métodos para que nuestro bot se relacione con ella, por lo que nos vemos conducidos a utilizar el ratón y teclado a modo usuario de manera automática.

2. **Obtención de datos del sistema:** Extracción de datos del sistema como la resolución de pantalla para el posterior procesado de imágenes o realización de capturas de pantalla, las cuales pueden ser analizadas para determinar posiciones y estados de la aplicación.

3. **Operaciones con imágenes:** Implica la obtención de capturas adaptadas a la región de la pantalla que queramos analizar. También nos proporciona la funcionalidad de la obtención del valor de los diferentes pixeles de una imagen, con la cual poder en otros módulos comparar imágenes.

4.4 Módulo Inteligencia

Dejando a parte la funcionalidad manual que permite al usuario obtener sus propios cálculos y preparar sus estrategias, nos encontramos con el módulo de inteligencia que es el que contiene hablando de manera genérica el modo de comportamiento del bot junto con otras operaciones necesarias para poder determinar dicho modo de actuación.

4.4.1 Algoritmos básicos de análisis

Para determinar el comportamiento del bot, primeramente hemos de dotarlo de una serie de habilidades que le permitirán tanto interactuar con el sistema basándose en el módulo de Control de Jugadas explicado anteriormente, como algoritmos que le permitirán interpretar dichos datos obtenidos y traducirlos

en resultados simples con los que poder determinar una toma de decisiones más clara.
Refiriéndonos a los métodos como funciones o algoritmos que interactuan con el sistema para obtención de datos e interpretación de los mismos así como de los algoritmos que permiten su análisis basados en el módulo de control de jugadas explicado anteriormente, tenemos los siguientes:

1. **Comparador de imágenes:** Algoritmo al cual se le proporcionan dos imágenes capturadas o almacenadas en un fichero previamente y a partir del valor de los pixeles de la imagen proporcionados por el módulo de control de jugadas, calcula el porcentaje de similitud de ambas imágenes. Este algoritmo a partir de el vector RGB de color de cada pixel calcula pixel a pixel la diferencia de la pareja y a partir de los cuadrados da cada valor del vector y hace la raíz cuadrada obteniendo así valores estandarizados con los cuales, a partir de su suma y teniendo en cuenta el tamaño total de la imagen, sus máximos y mínimos valores, obtener el porcentaje de similitud.

2. **Comprobación de turno:** Función simple la cual nos permite, basándose en las funciones de apertura de ficheros y obtención de regiones localidades de capturas de pantalla del módulo control jugadas, así como del algoritmo de comparación de imágenes, obtener comparando imágenes de control previamente guardadas en memoria con las capturadas en tiempo real su similitud y de esta manera determinar si es nuestro turno o no.

3. **Obtención de valores de las cartas propias:** Esta funcionalidad nos la proporciona un algoritmo el cual de manera similar al algoritmo que determina si es nuestro turno, calculando en función de una serie de imágenes de control que corresponden con las 52 cartas de la baraja de poker, y determinando cual de las 52 cartas obtiene un porcentaje de similitud mayor, por lo cual tendremos que es dicha carta. Este proceso lo realiza en paralelo para ambas cartas comparándolas con las ya almacenadas.

4. **Comprobación de cartas de la mesa:** Función que determina, dependientemente de la fase en la que estemos, la cual se le pasa como parámetro, la existencia de dichas cartas en la mesa. Este método es simple y no retorna el valor de la carta, tan solo comprueba de su existencia al igual que lo hace el método de comprobación de turno.

5. **Obtención valor cartas centrales:** Algoritmo que nos permite realizar una operación similar al de comprobación de cartas de la mesa, con el añadido de que este algoritmo no solo determina si existe dicha carta o no, si no que también de manera idéntica al algoritmo de obtención del valor de las cartas propias, obtiene el valor de las cartas centrales.

6. **Cálculo del número de jugadores:** Algoritmo que busca en la aplicación el número de jugadores que continúan jugando excluyéndonos a nosotros

mismos. Este cálculo lo realiza a partir de una serie de imágenes de control, las cuales son buscadas por diferentes sectores localizados de las capturas de pantalla realizadas y a partir de las cuales se obtiene en tiempo real el número de jugadores que intervienen en cálculos posteriores.

Si nos vamos ahora fijando en los método de interactuación con el sistema, no para la obtención de datos, sino para la generación de acciones básicas contra la aplicación objetivo, tenemos diferentes algoritmos que nos permiten una vez tomadas las decisiones a partir de la obtención de datos anterior, interactuar con dicha aplicación de tal manera que abarquemos todas las posibles salidas de la máquina de estados que rige la inteligencia del bot que explicaremos posteriormente:

1. **Raise:** Método que nos permite buscar mediante las funciones de comparación de imágenes y de obtención de capturas explicados anteriormente, determinar si tenemos la opción de resubir la apuesta de otro jugador. En caso de tener dicha opción, la realiza y retorna la conclusión de la operación.

2. **Bet:** Función idéntica a la anterior pero en este caso nos permite comprobar si podemos realizar la operación y en caso de poder hacerla apostar.

3. **Call:** De manera idéntica a las funciones anteriores, esta función nos permite igualar la apuesta de un rival.

4. **Check:** Igualmente a los anteriores nos permite realizar una llamada idéntica a la de call pero en este caso no requiere de igualar la apuesta del rival debido a que estos no han realizado incrementos de la apuesta.

5. **Fold:** Método que nos permite dejar la mano en caso de que no tengamos las probabilidades necesarias para ganar.

6. **Búsqueda de sitio:** Algoritmo inicial que nos permite buscar sitio en la mesa clicando con el puntero en diferentes posiciones clave y retornando el puesto en el cual hemos conseguido un puesto. En caso de que la mesa en la que nos queramos sentar este completa, este algoritmo sigue insistiendo en todos los puestos hasta sentarse o hasta que el usuario determine finalizarlo.

4.4.2 Máquina de estados de comportamiento del BOT

Esta máquina de estados o inteligencia básica, es la que determina en todo momento y abarcando todas las posibilidades la manera de actuar del bot frente a el contexto que se encuentra en cada momento. Los diferentes contextos los dividiremos en fases, correspondiendo las mismas con las fases del juego en sí explicadas en secciones anteriores. Dichas fases se enumeran a continuación:

1. **Fase 0:** Fase inicial del bot, que corresponde con la fase de pre-flop explicada anteriormente. En esta fase hemos de determinar el contexto inicial calculando el número de jugadores de la mesa así como de la posición en la que nos hemos sentado y otros parámetros básicos para que el bot determina en la posición en la que se encuentra.

2. **Fase 1:** Correspondiendo con la fase de flop, el contexto cambia significativamente debido a que tenemos tres cartas centrales descubiertas con las que contar a parte de los datos que hemos de recoger de nuevo al igual que hicimos en la fase 0.

3. **Fase 2:** En esta fase tenemos una carta más debido a su correspondencia directa con la fase de turn anteriormente explicada. Al igual que en las fases anteriores hemos de recalcular el contexto, teniendo en cuenta la influencia del mismo en el recalculado de las estadísticas.

4. **Fase 3:** Ultima fase del proceso del bot en la cual abarcamos las fases de river y show down, debido a que el bot permanecerá en esta fase hasta determinar el resultado de la mano. Una vez tenga dicho resultado podrá reinicializar los valores de los cálculos y de contexto almacenado en fases anteriores y podrá volver a la fase 0 para continuar jugando.

Una vez tenemos especificadas las fases y los posibles escenarios que nos vamos a encontrar, hemos de determinar los criterios a cumplir para el cambio de fase que se apoyan en los métodos de obtención y procesado de datos especificados anteriormente. Dichos criterios son dependientes de la fase en la que nos encontremos así como de el contexto de juego específico del momento en el que necesitemos tomar una decisión a la hora de actuar.

1. **Entrada en Fase 0:** Contando que el momento que el bot entra a interactuar con la aplicación puede no coincidir con el momento en el cual comienza la mano y por tanto la entrada a la fase 0, tenemos que determinar el periodo en el cual el bot no está todavía jugando. Para la entrada en la fase 0 el bot debe determinar cual es su turno, basándonos en la función de turno explicada anteriormente y si tenemos cartas asignadas.

2. **Fase 0 $\longrightarrow$ Fase 1:** Una vez que el bot ha realizado con éxito las operaciones contenidas en la fase 0, en las cuales entraremos con más profundidad en apartados posteriores, se ha de determinar si ha finalizado dicha fase o ha de continuar en la misma. Para ello hemos de comprobar si todos los rivales han finalizado su turno y si posteriormente a que todos ellos finalicen su turno hemos de comprobar si tenemos las tres cartas centrales propias de la fase de flop.

3. **Fase 1 $\longrightarrow$ Fase 2:** Al igual que en la fase anterior hemos de comprobar para el cambio de fase si los rivales han finalizado su turno y si encontramos en la

mesa la cuarta carta central la cual nos permitirá determinar que estamos en la fase de turn.

4. **Fase 2 ⟶ Fase 3:** Esta fase al igual que las dos anteriores se basa en la búsqueda de nuestro turno en el cual determinaremos si están ya las cinco cartas centrales sobre las mesa, requisito que nos permitirá determinar si estamos en la fase 3, de lo contrario y como en el resto de fases permaneceremos en la fase anterior debido a que podemos encontrarnos en un estado de resubida de apuestas.

5. **Fase 3 ⟶ Fase 0:** Estos requisitos no solo determinan la permanencia o no en la fase 3, sino que también determinan si hemos de volver al estado inicial de la máquina de estados, con lo cual hemos de reiniciar los valores de contexto calculados. Estos requisitos son cruciales debido a que si los calculamos mal encontraremos problemas, como por ejemplo que el bot piense que esta en la fase 0, cuando en realidad debería continuar en la fase 3, la cual ya conocemos a ciencia cierta la jugada máxima que tenemos y lo máximo que nos podemos arriesgar, lo que nos acarrearía perdidas. Los requisitos son que tras la comprobación de que hemos finalizado nuestra toma de decisiones el resto de jugadores finalicen sus manos y en la parte central de la mesa no queden cartas. Por otro lado también hemos de comprobar que las cartas propias asignadas son diferentes a las dadas en la mano anterior, con lo cual aseguraremos de manera más precisa que estamos en la fase 0.

Concluido el análisis de las fases de la máquina de estados que rige el comportamiento del bot, hemos de obtener un esquema general, el cual nos permitirá ver de manera más global el funcionamiento lógico del mismo. Dicho comportamiento rige, en función de los requisitos citados previamente, la toma de una serie de acciones en cada caso. Estas acciones se basan en jugadas de poker antes citadas, las cuales nos permiten realizar acciones e interactuar con la aplicación, creando de esta manera un nuevo contexto que se tendrá que analizar de nuevo. Todo este comportamiento queda plasmado en las Figuras 9, 10, 11, 12, 13 que muestran de manera global la máquina estados.

Como vemos en las máquinas de estados de dichas figuras, tenemos que el bot calcula previamente antes de simular unos rangos de probabilidad los cuales son utilizados para saber si tiene o no que continuar jugando. Estos rangos son dos y el primero es un valor calculado como la probabilidad promedio de ganar la mano más un porcentaje extra determinado por el usuario para controlar el riesgo (100/jugadores + gap) %. El segundo valor se calcula de igual manera, con la diferencia de que el valor añadido por el usuario es mayor por que este valor será usado para toma de decisiones en caso de apostarse todas las fichas disponibles.

De las figuras antes citadas, las primeras contienen la actuación de la inteligencia en función de la fase y son similares, pero tienen importantes pequeños cambios debido al comportamiento de la aplicación con la que vamos a interactuar. Por

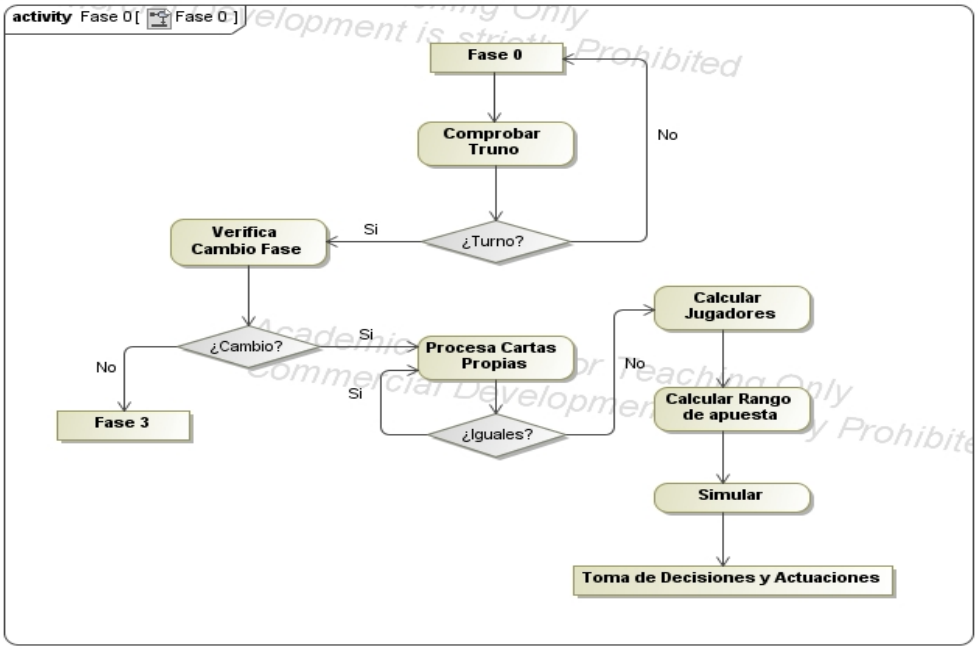

Figure 9: Diagrama de actividad de la Fase 0.

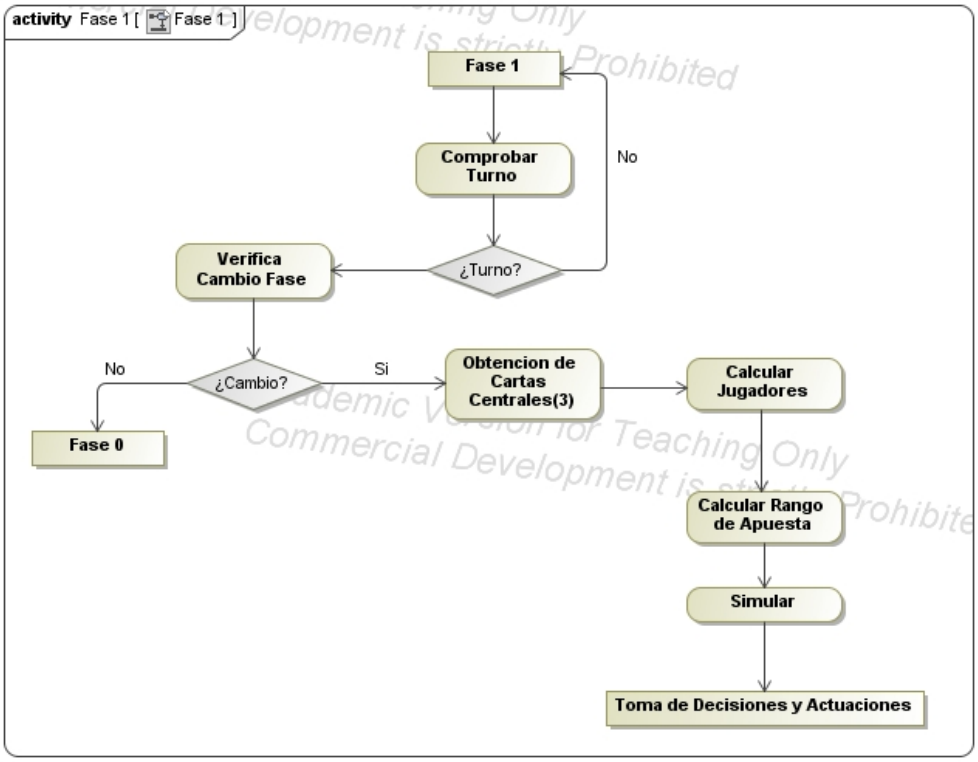

Figure 10: Diagrama de actividad de la Fase 1.

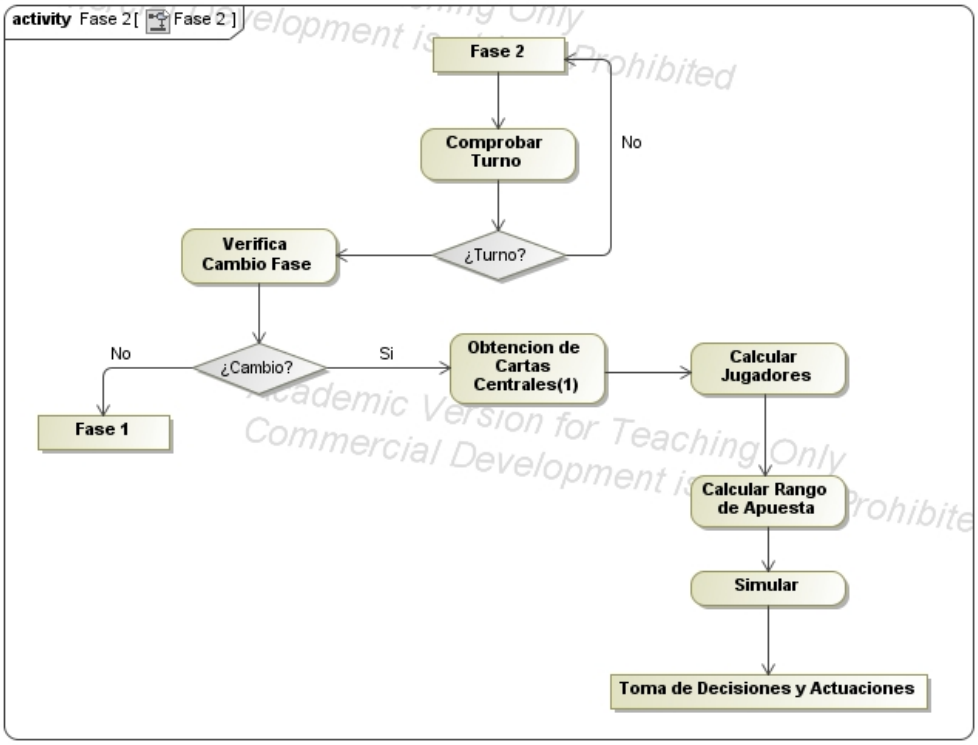

Figure 11: Diagrama de actividad de la Fase 2.

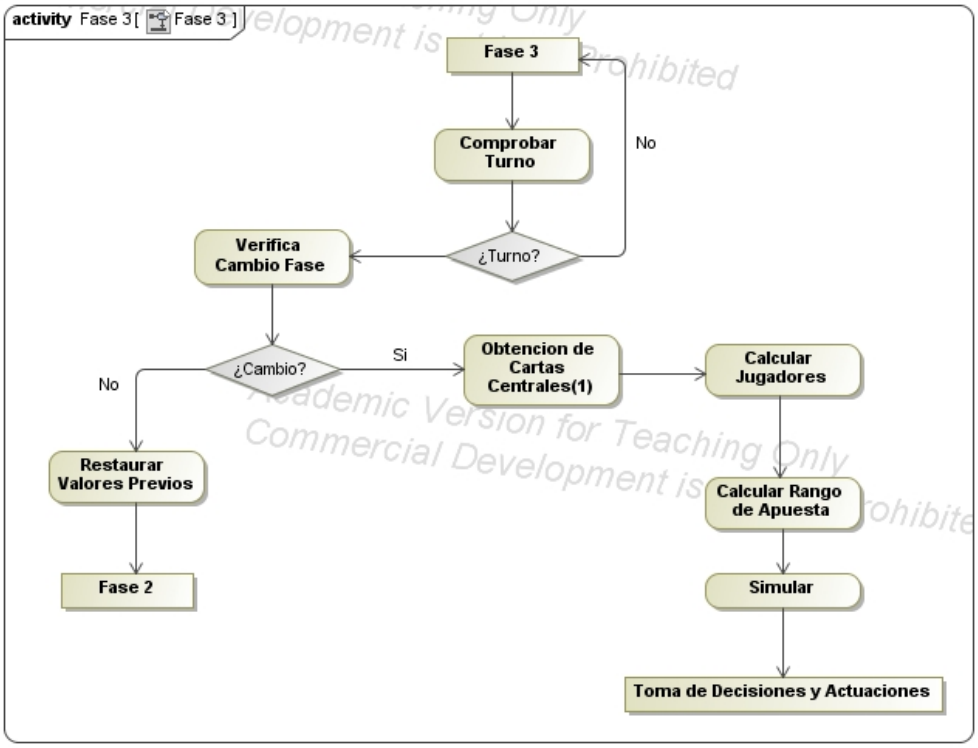

Figure 12: Diagrama de actividad de la Fase 3.

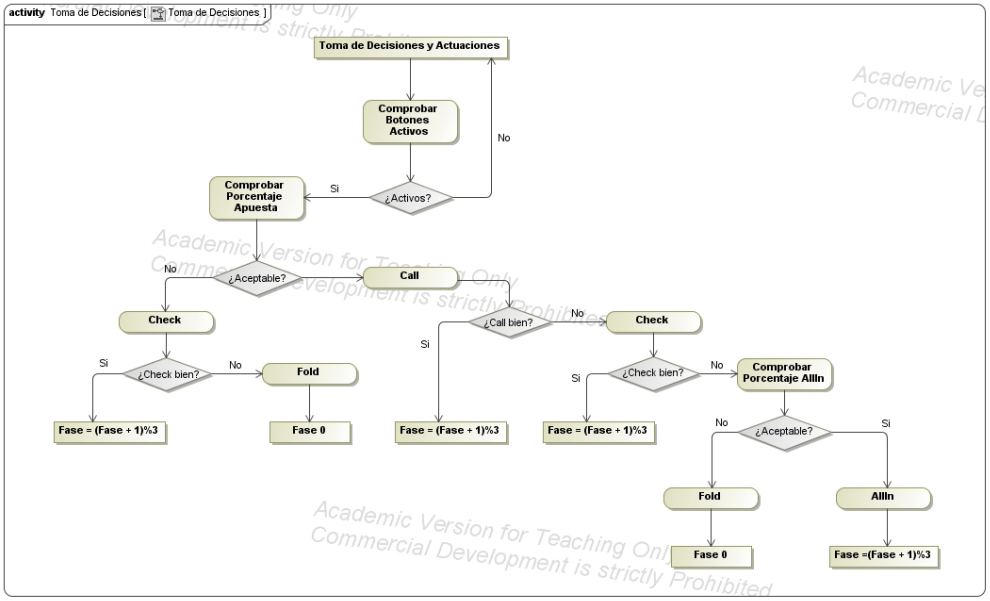

Figure 13: Diagrama de toma de decisiones del bot.

otro lado, el último diagrama que contiene la toma de decisiones del bot una vez analizado otros parámetros dependientes de la fase, es común para todas las etapas de nuestro bot debido a que el modo de juego es igual para todas las fases a pesar de que los datos a analizar sean diferentes. Esto facilita el poder cambiar únicamente dicho comportamiento, obteniendo una inteligencia mayor independientemente de la obtención de datos externa y viceversa, en caso de que lo que se quiera cambiar no sea la inteligencia si no el comportamiento del bot en una fase determinada debido posiblemente a una actualización de la aplicación externa.

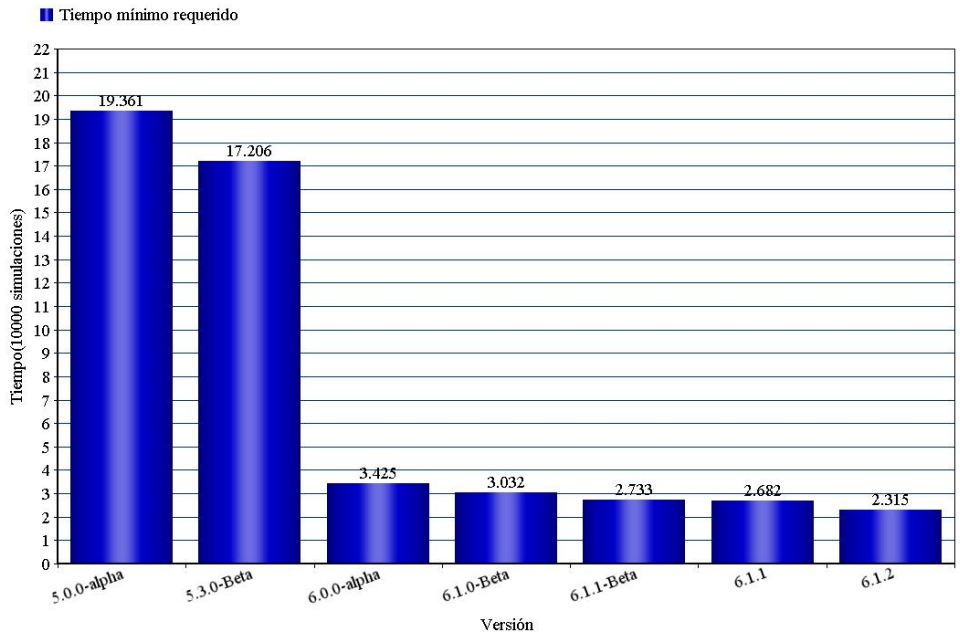

Figure 14: Gráfica de tiempos según versión.

5 Historial de optimizaciones

En este apartado se muestra de manera cronológica las diferentes revisiones de la aplicación completa, indicando la fecha de revisión, la versión asignada y los cambios, resolución de errores y optimizaciones realizadas.

También se muestran basándonos una serie de versiones clave de nuestra aplicación los tiempos de ejecución obtenidos de las diferentes versiones funcionales y su progresión a medida que se ha ido optimizando hasta obtener unos tiempos acorde con los requisitos de la aplicación antes mencionados. Este estudio de tiempos está calculado a partir del número máximo de jugadores con una cantidad de simulaciones igual a 10000. Al tratarse de un algoritmo lineal, el número de simulaciones y el tiempo crecen en igual manera por lo que dicho número no es significativo, frente al número de jugadores el cual si que influye en el crecimiento temporal, de ahi que se coja el máximo.

Índice	Fecha	Versión	Cambios representativos
1	18/02/2013	1.0.0-alpha	Versión inicial con las funciones básicas y sin inteligencia.
2	21/02/2013	1.1.0-alpha	Introducción de simulación aleatoria de cartas centrales. Mejora superficial en la interfaz de usuario. Corrección de errores en el cálculo del valor de jugadas.
3	23/02/2013	1.2.0-alpha	Implementación de función de obtención de resolución del monitor. Corrección de errores de la interfaz.
4	26/02/2013	2.0.0-alpha	Optimización de generación aleatoria de cartas. Creación de submenus en la interfaz.
5	04/03/2013	2.0.1-alpha	Resolución de errores de submenus de la interfaz.
6	06/03/2013	2.1.0-alpha	Introducción de simulación aleatoria de cartas de jugadores. Mejora de submenus para adaptarlos a cambios previos.
7	12/03/2013	2.2.0-alpha	Implementación de funciones de interactuación con el sistema. Solución de errores en el algoritmo de generación de cartas.
8	19/03/2013	3.0.0-alpha	Implementación de funciones de interactuación con el monitor. Adaptación de la interfaz para ejecución del bot. Implementación de algoritmo de comparación de imagenes.
9	22/03/2013	3.1.0-Beta	Mejora del algoritmo de comparación de imagenes. Inclusión de interactuación con ficheros del sistema. Mejora de métodos de análisis de imagenes.
10	25/03/2013	4.0.0	Implementación de interacción con aplicación externa. Añadido algoritmo para analizar las cartas. Mejora del submenu de función automática. Corrección de errores en módulo estadístico. Optimización del algoritmo de simulación. Inclusión de control de datos de entrada de interfaz de usuario.
11	23/04/2013	5.0.0-alpha	Implementación de la inteligencia del bot. Mejora del algoritmo de simulación. Inclusión de mejora de comprobación de cartas repetidas.
12	25/04/2013	5.1.0-alpha	Reconocimiento de cartas centrales añadido al bot. Mejora del comportamiento de la inteligencia. Corrección de errores del algoritmo de comportamiento.
13	30/04/2013	5.1.1-alpha	Recalculados límites estadísticos de riesgo. Corrección de errores en cálculo de estadísticas.
14	15/05/2013	5.1.2-Beta	Corrección de errores de algoritmos de inteligencia.
15	17/05/2013	5.2.0-Beta	Introducción de sistema automático avanzado en la interfaz. Redistribución de funciones en módulos para optimizar la modularidad. Optimización de tiempos de ejecución de algoritmo de simulación.
16	19/05/2013	5.3.0-Beta	Unificación de funciones de simulación para mejora de tiempos. Mejora del algoritmo de simulación de cartas. Optimización importante código Python para simular. Reducción significativa de tiempos de simulación.
17	21/05/2013	6.0.0-alpha	Rediseño del algoritmo cálculo de valores de jugadas. Mejora de tiempos basada en restricciones. Adaptación de la interfaz al nuevo diseño. Optimización importante de tiempos de ejecución.
18	23/05/2013	6.0.1-Beta	Optimización de código Python en cálculos básicos. Mejora superficial de interfaz de usuario.
19	31/05/2013	6.1.0-Beta	Optimización de tiempos de ejecución globales.
20	01/06/2013	6.1.1-Beta	Corrección de errores de interactuación con el sistema. Corrección de errores estéticos de la interfaz. Optimización de tiempos de ejecución.
21	08/06/2013	6.1.1	Generación de versión final probada y verificada.
22	12/06/2013	6.1.2	Optimización de tiempos no críticos.

Table 3: Control de cambios y versiones

Tiempo	Jugadores	Cifra Inicial	Cifra Final	Beneficios
10 min	5 - 9	160	0	-160
20 min	2 - 5	135	252	117
15 min	4 - 9	160	331	171
15 min	6 - 9	120	107	-13
25 min	4 - 6	160	0	-160
10 min	3 - 5	40	109	69
25 min	4 - 6	40	0	-40
35 min	4 - 8	200	0	-200
15 min	3 - 6	160	765	605
30 min	2 - 6	400	336	-64
20 min	6 - 9	160	0	-160
30 min	4 - 9	120	13	-107
4 h 10 min		1855	1913	58

Table 4: Pruebas y Análisis del bot

6 Pruebas y análisis de resultados

Una vez que hemos conseguido una versión estable de la aplicación con las funcionalidades que estamos buscando es hora de verificar el correcto funcionamiento de nuestra aplicación. Para ello basándonos en la version 6.1.2 de las antes mencionadas hemos puesto a prueba sus capacidades, por lo que hemos probado la aplicación en diferentes escenarios posibles, variando el número de jugadores, la dificultad de los mismos y el tiempo de ejecución que el bot ha estado jugando para concluir los resultados indicados en la Tabla 4.

Como podemos ver en dicha tabla, tenemos que el bot registra pérdidas sobre todo cuando se enfrenta a un mayor número de rivales. También influye bastante el modo de juego de los adversarios, debido a que pueden ser muy agresivos con lo que nuestro bot tiene problemas. También no solo influye el número de jugadores y su estrategia de juego, sino también el tiempo que juegue el bot de manera continuada, puesto que los rivales pueden obtener el patrón de juego del bot y así actuar en consecuencia. De todas formas como vemos, consigue jugar de manera automática frente a humanos durante varias horas consiguiendo unos beneficios mínimos. También hay que resaltar que dichas pruebas se han realizado en mesas de dinero ficticio en las que el nivel de los jugadores rivales es bajo.

7 Posibles mejoras y optimizaciones

En cuanto al funcionamiento de la aplicación así como de el bot de juego automático, hemos de decir que son totalmente funcionales y libres de errores, pero este solo es un primer paso para el desarrollo de una suite completa. En los siguientes apartados trataremos de forma más detallada las posibles mejoras que se pueden implementar sobre la aplicación para que esta proporcione más

funcionalidades o implemente una inteligencia más óptima, la cual basándose en diferentes cálculos mentados a continuación a demás de los estadísticos ya implementados, proporcione un modo de juego más seguro y basado no solo en el contexto si no también en el comportamiento de el resto de jugadores.

7.1 Mejoras del rendimiento de simulación

Como hemos ido mostrando en los apartados anteriores los tiempos de simulación conseguidos son suficientes para obtener una probabilidad fiable a partir de la cual poder tomar decisiones, pero estos tiempos pueden ser mucho más optimizados, obteniendo de esta manera unos cálculos más precisos y por lo tanto más fiables. La optimización de estos tiempos de simulación no solo nos proporciona una mayor precisión si no que también al requerir menos tiempo, estas optimizaciones permiten realizar otro tipo de cálculos que se muestran en la subsección siguiente.

Las posibles optimizaciones de tiempos pueden encontrarse en diferentes ámbitos de la aplicación. El primero de ellos lo encontramos de manera directa en el algoritmo de simulación aleatorio, el cual podría ser sustituido por otro más preciso basado en la teoría del juego que tratamos.

Por otro lado al tratarse de código Python el cual no es precompilado si no que se ejecuta directamente y el cual está preparado para cálculos matemáticos complejos, pero no tanto para acciones repetitivas, nos limita el rendimiento máximo de nuestra aplicación en cuanto a lo que optimización del código se trata. Dicho código puede ser precompilado y optimizado mediante interpretes como PyPy [14], el cual nos proporciona una reducción considerable en los tiempos de ejecución o incluso ser implementado en otro lenguaje como C.

Por último, en lo que optimizaciones de tiempos se refiere hemos de nombrar la posible aplicación de hilos para subdividir las tareas propias de la aplicación, paralelizando de esta manera el trabajo y obteniendo un bot más óptimo capaz de hacer diversas tareas simultáneamente.

7.2 Mejoras en la funcionalidad

Una vez visto la posibilidad de optimizar nuestra aplicación hasta el punto de que los requisitos de tiempo real de la aplicación objetivo no sean un problema, hemos de hablar sobre las posibles mejoras tanto en la funcionalidad de la aplicación, como en la aplicación de otras estrategias a la hora de establecer una inteligencia capaz de jugar de manera más óptima, analizando datos de los rivales.

En primer lugar, la aplicación podria ser ampliada para que pueda ser ejecutada en multiplataforma. También se podrían ofrecer otras funcionalidades directas al usuario como análisis de jugadores mediante el estudio de los mismos a medida

que van jugando. Por otro lado también se podría hacer que el usuario pudiera obtener una tabla de manos con las cuales apostar en determinadas situaciones.

Por otro lado en cuanto a lo que estrategias de juego del bot se refiere, este podría analizar a los diferentes jugadores con los que esta compitiendo a partir de una estrategia basada en minería de datos y de esta manera estimar el rango de cartas de los rivales analizando los datos de las manos previamente jugadas. De este modo calcularemos la probabilidad de ganar la mano de forma más realista que si le asignamos unas cartas aleatorias. También como hemos visto en el apartado de pruebas tenemos que el bot sería más óptimo si en vez de jugar continuamente, este parase y volviese a jugar con una cantidad de dinero ficticio inicial mínima en el momento que sus beneficios sean altos o lleve mucho tiempo en una misma mesa, dado que el juego con stacks grandes es más complejo.

Otra mejora, dentro de las muchas posibles sería obtener el valor esperado de la variable beneficio neto, utilizando las probabilidades descritas en el párrafo anterior para lo cual también deberíamos poder obtener de la aplicación externa otros datos como el pot(cantidad total en juego) y el stack(cantidad con la que contamos nosotros) para su uso en el cálculo del valor esperado. La obtención de dichos datos no solo requeriría un algoritmo más complejo de reconocimiento de imágenes del que nosotros hemos implementado sino que también requeriría de un algoritmo de reconocimiento de patrones, debido a que los números no son imágenes estáticas posicionadas en unas coordenadas concretas. La obtención de dichos datos no solo requeriría un algoritmo más complejo de reconocimiento de imágenes del que nosotros hemos implementado sino que también requeriría de un algoritmo de reconocimiento de patrones, debido a que los números no son imágenes estáticas posicionadas en unas coordenadas concretas.

Referencias

[1] Caroline Pantofaru y Martial Hebert, *A Comparison of Image Segmentation Algorithms*, Septiembre 2005.

[2] Licesio J. Rodríguez-Aragón. *Simulación, Método de Montecarlo*, Marzo 2011.

[3] Nicholas Abou Risk, *Using Counterfactual Regret Minimization to create a conpetitive multiplayer poker agent*, 2009, University of Alberta.

[4] Raul Gonzalez Duque, *Python para todos*, 2011.

[5] Jonathan Rubin e Ian Watson, *Computer Poker: A Review*, 21 Enero 2011, University of Auckland.

[6] Michael Bradley Johanson, *Robust Strategies and Counter Strategies: Building a Champion Level Computer Player*, 2007, University of Alberta.

[7] Darse Billings, *Computer Poker*, Agosto 1995, University of Alberta.

[8] Terence Conrad Schauenberg, *Opponent Modelling and Search in Poker*, 2006, University of Alberta.

[9] Fredrik Lundh, *Python Imaging Library Handbook*, Mayo 2005.

[10] Aaron Davidson, *Opponetn Modeling in Poker: Learning and Acting in a Hostile and Uncertain Environment* , 2002, University of Alberta.

[11] John W. Shipman, *Tkinter 8.5 reference: a GUI for Python*, 2013, New Mexico Tech Computer Center.

[12] Guillem Borrell Nogueras, *Python como entorno de desarrollo cientifico*, Julio 2008.

[13] Guido van Rossum, *Guía de aprendizaje de Python*, Octubre 2000, BeOpen PythonLabs.

[14] Carl Friedrich, Antonio Cuni, Maciej Fijalkowski, Michael Leuschel, Samuele Pedroni y Armin Rigo, *Runtime Feedback in a Meta-Tracing JIT for Efficient Dynamic Languages*, 2011, Heinrich-Heine-Universität Düsseldorf.