Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Implementación en Matlab del algoritmo *Minimax* en un juego simple mediante interfaces gráficas de usuario

Autor: Antonio Fernández Delgado

Tutores: Santiago Díaz Madrigal

Fernando Fernández Sánchez

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

DEPARTAMENTO DE
MATEMÁTICA APLICADA II

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Implementación en Matlab del algoritmo *Minimax* en un juego simple mediante interfaces gráficas de usuario

Autor:
Antonio Fernández Delgado

Tutores:
Santiago Díaz Madrigal
Catedrático de Universidad
Fernando Fernández Sánchez
Profesor Titular de Universidad

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Implementación en Matlab del algoritmo *Minimax* en un juego simple mediante interfaces gráficas de usuario

Autor: Antonio Fernández Delgado
Tutores: Santiago Díaz Madrigal
Fernando Fernández Sánchez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia, en especial a mis padres por apoyarme en los muchos momentos difíciles que he vivido desde que comencé la carrera, por celebrar mis aprobados incluso más que yo y por estar ahí siempre para ayudarme en lo que hiciera falta. A mi abuelo, que aunque no ha podido verme como ingeniero aeroespacial, seguro que estará orgulloso de mí.

A mis tutores, Santiago y Fernando. Por guiarme y aconsejarme a lo largo del trabajo, siempre dispuestos a echarme una mano y enseñándome todo el potencial de un programa tan importante en la ingeniería como es Matlab. En definitiva, por hacerme un poco más fácil el camino.

Al Club Natación Jerez, club al que entré hace ya 14 años y que me ha inculcado valores importantísimos como el esfuerzo, la disciplina y el trabajo diario. A los chicos y chicas que he entrenado estos años de carrera en los que no he tenido tiempo de jugar, por ser mi válvula de escape en los peores momentos, incluso sin ellos saberlo.

Gracias.

Antonio Fernández Delgado
Jerez de la Frontera, 2021

Resumen

Usando como ejemplo central el conocido juego de tres en raya y alguna de sus variantes, este trabajo aborda la implementación en Matlab del algoritmo *Minimax* en su forma canónica y con las mejoras proporcionadas por la *Poda Alfa–Beta*. Previamente será necesario hacer una breve introducción a la teoría de juegos y a los árboles de juego, tan importantes en la programación y comprensión del *Minimax*

El objetivo final es la creación en Matlab de una aplicación interactiva mediante una interfaz gráfica de usuario (GUI) avanzada, que incluya objetos gráficos, botones, menús desplegables, mensajes de error e informativos, imágenes tratadas digitalmente y sonidos. La aplicación desarrollada es totalmente personalizada y única, creada desde cero y con numerosas funcionalidades y opciones para el usuario.

Abstract

Using the example of the well-known game Tic-tac-toe (or Noughts and Crosses), this project broach the implementation in Matlab of the *Minimax* algorithm in its canonic form and with the improvements given by *Alpha–Beta Pruning*. Previously it will be necessary a brief introduction to game theory and game trees, so importants in coding and understanding of *Minimax*

The final aim is the creation in Matlab of an interactive application by means of an advanced graphic user interface (GUI), which includes graphic objects, buttons, drop–down menus, error and informative messages, digitally processed images and sounds. The developed application is completely custom–made and unique, created from scratch and with a large number of funcionalities and usefulnesses for the user.

Índice

1 Introducción

En este trabajo se pueden diferenciar claramente dos bloques: la parte matemática, en la que se desarrolla e implementa el algoritmo *Minimax*, y la parte de programación o programática, donde se crea la aplicación desde cero usando objetos gráficos y objetos propios de interfaces gráficas de usuario (GUI) de Matlab.

Sin embargo, es interesante dedicar el capítulo 2 a introducir brevemente el juego del tres en raya. Se resume primero la historia y procedencia del juego, para luego pasar a describir las estrategias de juego y las opciones de ambos jugadores. Para acabar el capítulo, se tratan las algunas de las variantes del tres en raya, generalmente más complejas y de estrategias más difíciles.

El capítulo 3 cubre la parte matemática. Antes de exponer el algoritmo *Minimax*, es necesario hablar de la teoría de juegos y de la forma que tiene esta rama de las matemáticas de representar los movimientos del juego: los árboles de juego. Tras un ejemplo del funcionamiento del algoritmo en un árbol de un juego genérico, se procede a la implementación y explicación del código del *Minimax*. De igual manera, un ejemplo específico de una partida de tres en raya ayudará a la comprensión del algoritmo. Una vez descrito el algoritmo *Minimax*, se implementa una mejora muy usada y eficiente, si bien en el tres en raya no se notará mucho su uso. La *Poda Alfa–Beta* tiene como objetivo principal no analizar ramas que no sean necesarias, es decir, que no vayan a influir en el resultado final. Esto supondrá un ahorro de tiempo considerable (dependerá de cuantas ramas se puedan podar) o bien un aumento de la profundidad con la que se analiza el árbol de juego.

Toda la programación de la interfaz gráfica de la aplicación se lleva a cabo en el capítulo 4. Primero se detallan los objetos multimedia usados, tanto imágenes como sonidos digitales, que serán de utilidad en la creación de algunos de los numerosos objetos gráficos y objetos propios de interfaces gráficas que se plantean a continuación. Para finalizar, se desglosan y explican las funciones auxiliares que permiten el desarrollo de las partidas.

Por último, en el capítulo 5 se presentan las conclusiones más relevantes que se derivan de la realización de este trabajo. Se incluyen algunas imágenes de la aplicación en funcionamiento y se comentan posibles futuras mejoras de la aplicación.

2 Funcionamiento del juego. Variantes y estrategias

El tres en raya o tres en línea es un juego de lápiz y papel en el que dos jugadores, **O** y **X**, marcan alternativamente las casillas de un tablero 3×3. Para ganar la partida se deben situar tres marcas en fila, ya sea en horizontal, en vertical o en diagonal. El tres en raya es un juego resuelto, es decir, su resultado (victoria, derrota o empate) se puede predecir desde cualquier posición asumiendo un juego perfecto por parte de ambos jugadores. El juego perfecto hace referencia a la estrategia de un jugador que desemboca en el mejor resultado posible independientemente de las jugadas de su oponente. Se hablará de esto más en detalle posteriormente.

Es conocido por todo el mundo que en el caso de juego perfecto de ambos jugadores se llega inevitablemente a un empate. Es precisamente esta simplicidad del tres en raya la que lo convierte en ideal para iniciarse en la *teoría de juegos* y en la rama de la inteligencia artificial que estudia los *árboles de juego*.

2.1 Historia

El tres en raya es posiblemente el juego de estrategia más conocido a nivel mundial; millones de personas han jugado a lo largo de su amplia historia.

Aunque algunos expertos fechan su origen en el antiguo Egipto, la primera prueba de algo parecido al tres en raya se encuentra en la época romana. El *terni lapilli*, por su traducción "tres piedras" o "piedras de tres en tres", no es más que una variante del tres en raya. La única diferencia es que en el *terni lapilli* cada jugador tiene solo tres fichas y, si cuando colocan todas sus fichas ninguno ha conseguido encadenar las tres en una fila (lo más común como se puede intuir), el juego continúa con el movimiento de las fichas. Cada jugador puede mover una de sus fichas en horizontal o en vertical sin pasar por encima de otra ficha, hasta que uno consiga tener sus tres fichas en fila.

La historia del tres en raya tal y como se conoce ahora empieza en Persia. Se trataba de un juego tan simple como popular, que se extendió rápidamente por Europa. Sin embargo, tuvo un periodo negro entre los siglos IX y X debido a su asociación con rituales paganos macabros y considerados demoníacos. Tan grave fue la situación que incluso se creó una bula papal que impedía a los cristianos jugar al tres en raya, dejando el juego como un elemento de magia negra. Esta medida, ademas de impopular, fue cayendo en el olvido hasta que el tres en raya resurgió sobre todo en las universidades. Durante el Renacimiento el juego volvió a formar parte de la cultura popular y pasó a ser universalmente conocido.

Finalmente, dando un salto hasta 1952, A.S. Douglas crea *OXO*, uno de los primeros videojuegos de la historia, como parte de su tesis en HCI (interacción humano–máquina) en Cambridge. Fue programado en el EDSAC (Electronic Delay Storage Automatic Calculator) y permitía jugar contra una inteligencia artificial que podía hacer un juego perfecto. El usuario debía elegir en cuál de las nueve casillas poner su ficha mediante un disco de marcar o dial rotatorio. *OXO* no estaba disponible al público, pues solo se podía jugar en el EDSAC del laboratorio de matemáticas de la Universidad de Cambridge, que por supuesto no podía moverse.

2.2 Estrategias

En este capítulo ya se ha mencionado el concepto de juego perfecto y lo que ello conlleva. Con juego perfecto por parte de los dos jugadores, una partida de tres en raya acaba siempre en empate. Teniendo en cuenta el tablero y las posibles simetrías, hay 138 configuraciones del tablero final posibles, siendo solamente tres de ellas un empate (91 victorias del primer jugador y 44 victorias del segundo jugador).

Cuando se habla de juego perfecto en tres en raya, se hace referencia a la estrategia a seguir para ganar o al menos no perder (empatar). Esta estrategia se puede resumir en los siguientes puntos:

1. Ganar. Si el jugador tiene dos casillas seguidas marcadas, debe marcar la tercera.

2. Bloquear. Si el oponente tiene dos casillas seguidas marcadas, el jugador debe marcar la tercera casilla de esa fila para evitar la victoria rival.

3. Bifurcar. Crear una oportunidad donde el jugador tenga dos opciones de ganar, es decir, dos líneas no bloqueadas de dos casillas seguidas.

4. Bloquear una bifurcación. Si hay posibilidad de que el oponente cree una bifurcación, el jugador debe bloquearla.

5. Centro. Marcar el centro en la primera jugada no es la mejor opción, pero en el caso de juego perfecto se llega a empate igualmente.

6. Esquina opuesta. Ocupar la esquina opuesta a una casilla ocupada por el oponente.

7. Esquina libre. Marcar una de las esquinas libres que queden.

8. Borde o casilla vacía. Marcar una de las 4 casillas medias.

Enfocándolo de otra manera, el primer jugador tiene tres posibles movimientos en la primera jugada: esquina, centro o borde. En realidad, son 9 posibilidades al haber 9 casillas, pero la simetría del tablero permite reducirlas, siendo equivalente por ejemplo marcar cualquiera de las cuatro esquinas. El primer jugador puede forzar al menos el empate en cualquiera de los tres casos, pero si comienza jugando en una esquina minimiza las posibilidades del oponente, que se ve obligado a jugar a no perder. Por tanto, suponiendo juego perfecto, la mejor apertura es marcar una esquina. Considerando juego no perfecto, el oponente puede responder a una apertura en la esquina jugando en el centro (movimiento óptimo), pero si su siguiente jugada es en una esquina y no en un borde, lo que llevaría a un empate inevitable, permitiría al primer jugador jugar en la esquina restante evitando la victoria del oponente al mismo tiempo que crea una bifurcación que le llevará a ganar la partida. Si el rival responde a la apertura jugando en un borde o una esquina, el primer jugador tiene garantizada la victoria (si juega perfecto).

Desde el punto de vista del segundo jugador, la mejor respuesta a una apertura en la esquina es jugar en la casilla central y luego en un borde, obligando así al oponente a defenderse e impidiendo posibles bifurcaciones. Si el primer jugador juega en un borde su apertura, el segundo jugador debe jugar en el centro o en alguna de las dos esquinas adyacentes, y seguir con la lista de prioridades arriba descrita. Por último, si la apertura es en una esquina, el segundo jugador debe jugar en la casilla central.

2.3 Juegos m,n,k

El tres en raya puede generalizarse a un juego m,n,k (m,n,$k-$games), en el que dos jugadores marcan alternativamente con fichas, símbolos, colores, etc. las casillas de un tablero $m \times n$, siendo el ganador el que consiga k de sus marcas en fila. Así, el tres en raya es el juego $(3,3,3)$, en el que un juego perfecto lleva siempre a un empate. Otras variantes pueden ser:

- El juego $(m,n,1)$ tiene solución trivial si $m > 1$ y $n > 1$, la victoria del que comienza jugando.
- El juego $(m,n,4)$ lleva a empate si $m \leq 5$ y $n \leq 5$, o bien a una victoria del primer jugador si $m \geq 6$ y $n \geq 5$ o $m \geq 5$ y $n \geq 6$. Es curioso el juego $(m,4,4)$, que resulta en victoria si $m \geq 30$ y en empate si $m \leq 8$, pero se desconoce la solución para $9 \leq m \leq 29$.
- La solución del juego $(m,n,5)$ es empate para $m \leq 8$ y $m \leq 8$.

Al aumentar el tamaño del tablero se complica considerablemente el análisis de soluciones, y las variantes de los juegos (m,n,k) son infinitas. A continuación se comentan tres de las más famosas variaciones del tres en raya:

- Gomoku. Es un "cinco en línea" que se juega en un tablero de Go 15×15 (o 19×19) con las piezas blancas y negras propias del Go. El objetivo, al igual que en el tres en raya, es encadenar cinco piezas seguidas horizontalmente, verticalmente o en diagonal. Es evidentemente más complejo que el tres en raya, pero se sabe que, sin restricciones en la primera jugada, las negras (que siempre empiezan jugando) pueden forzar la victoria relativamente fácil. Por ello se creó el Swap2, una regla que obliga al primer jugador a colocar dos piezas negras y una blanca, y posteriormente el segundo jugador puede elegir si juega con negras o blancas. Esta regla iguala mucho la partida y de hecho, a día de hoy, la inteligencia artificial alrededor del Gomoku sigue siendo tema de investigación, buscando cómo mejorar los algoritmos para hacerlos más competitivos. A pesar de ello, en 2017 el programa Yixin ganó al campeón mundial humano.
- Cuatro en raya 3D. El concepto es similar al tres en raya, pero se juega usando un tablero tridimensional $4 \times 4 \times 4$. El objetivo es encadenar una de las 76 posibles filas de cuatro marcas seguidas en horizontal, vertical, diagonal (en cualquiera de los planos). Tras analizarse matemáticamente, se sabe que el primer jugador puede forzar la victoria independientemente de que su rival juegue perfecto. La versión $3 \times 3 \times 3$ del juego no puede acabar en empate y puede ser fácilmente ganada por el primer jugador si marca la casilla central, por lo que no es un caso tan interesante de estudiar.
- Conecta 4. En este juego los jugadores dejan caer las fichas en un tablero vertical 7×6. Las fichas caen hasta la casilla vacía mas baja, siendo el objetivo alinear cuatro fichas en horizontal, vertical o diagonal. Se trata de un juego resuelto, en el que el primer jugador puede forzar la victoria poniendo su primera ficha en la columna central. Si comienza en una de las dos columnas adyacentes, y suponiendo juego perfecto por parte de ambos jugadores, se llegará a un empate, y si empieza colocando su ficha en las cuatro columnas exteriores (dos por cada lado) permitirá al segundo jugador ganar la partida.

3 Implementación del algoritmo *Minimax*

La teoría de juegos combinatoria es la rama de las matemáticas que estudia los juegos secuenciales, es decir, por turnos, y de información perfecta, ya que los jugadores saben todo lo que ha ocurrido desde el inicio del juego. En teoría de juegos, los movimientos de los jugadores se representan mediante árboles de juego. El tipo de juegos que estudia la teoría de juegos también es interesante desde el punto de vista de la inteligencia artificial, contribuyendo al desarrollo de métodos para analizar los árboles de juego. Así, en este trabajo se desarrollarán e implementarán dos de ellos: el algoritmo *Minimax* y la *Poda Alfa–Beta*. Algunos ejemplos clásicos de estos juegos son el ajedrez, el Go, las damas y, por supuesto, el **tres en raya**.

El motivo principal de la elección de este juego es su simplicidad, es fácil representar su árbol de juego y la ejecución del algoritmo no requiere excesivo tiempo. El tres en raya es comúnmente usado en la enseñanza de los principios básicos del diseño de inteligencia artificial. Además, es un juego resuelto, es decir, se puede predecir el resultado dada cualquier distribución del tablero. En el caso del tablero vacío, el resultado es un empate siempre que haya juego perfecto por parte de los dos jugadores.

El *Minimax* es un tipo de algoritmo de *backtracking* que se usa en la toma de decisiones y teoría de juegos para encontrar el movimiento óptimo de un jugador, asumiendo que su oponente también juega de forma óptima.

En el *Minimax* hay un jugador llamado *maximizador*, que intenta conseguir la mayor puntuación posible, y un *minimizador*, que trata de obtener la menor puntuación posible. Cada estado del tablero, es decir, cada posible distribución de las fichas en él, tiene un valor asociado que se calcula de forma diferente según el juego. Este valor se suele obtener mediante una función de evaluación estática (o heurística) que considera todas las posibles secuencias posteriores e indica cómo de bueno sería para el jugador que usa el algoritmo llegar a ese estado del tablero.

Esto lleva de manera inmediata a visualizar el algoritmo como un árbol de juego en el que se evalúa cada estado del tablero buscando el mejor movimiento posible.

3.1 Árbol de juego

En teoría de juegos, un árbol de juego es un gráfico que representa todos los posibles estados del juego, por lo que es de utilidad para medir la complejidad del mismo. Cada nodo se corresponde con un posible estado del juego y cada rama con uno de los posibles movimientos de los jugadores.

En el siguiente ejemplo sencillo (figura 3.1), se parte de un estado del tablero cualquiera en el turno del jugador azul (el que usa el *Minimax*) y se supone que en cada nodo solo hay dos posibles movimientos. Estos posibles movimientos serán dos ramas separadas del árbol que llegaran a dos nuevos estados del tablero diferentes, pero ahora será el turno del jugador rojo. Podemos repetir el procedimiento hasta que estén representados todos los posibles estados o hasta que se decida parar.

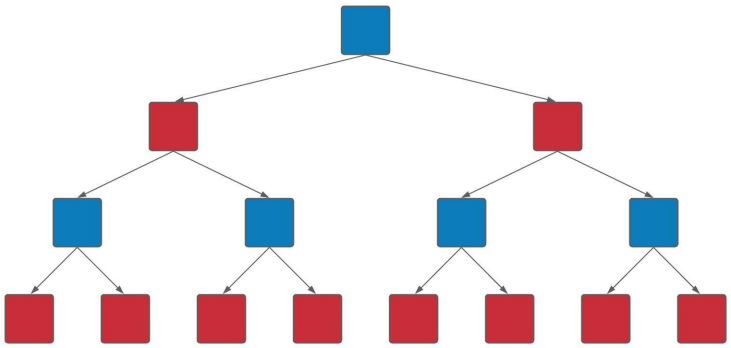

Figura 3.1 Generación del árbol de juego.

Ahora es el momento de usar la función de evaluación en las posiciones finales y obtener un valor para cada una de ellas (en este ejemplo son valores aleatorios). También es importante recordar que en el turno del jugador azul se intentará maximizar el valor y en el tuno del rojo se buscará minimizar.

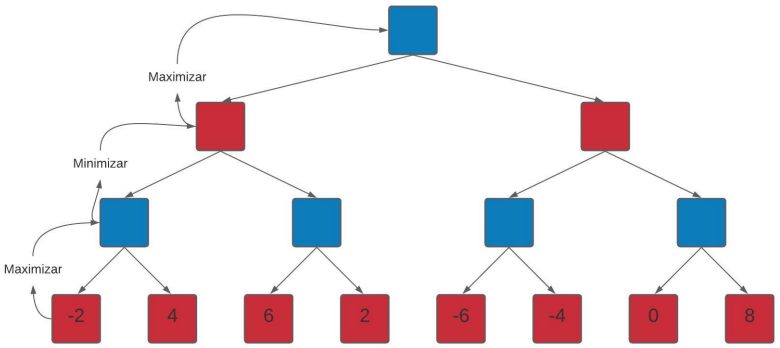

Figura 3.2 Asignación de puntuaciones de los nodos finales.

Si se observan las dos ramas más a la izquierda se tienen los valores -2 y 4. Como en la posición previa es el turno del jugador azul, éste escogerá la rama que lleve a un valor más alto. Por tanto, se puede asignar un valor de 4 a dicho nodo. Procediendo igual para el resto de ramas resulta:

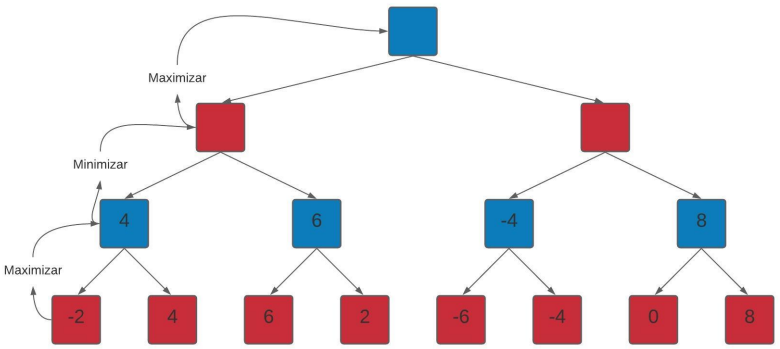

Figura 3.3 Asignación de puntuaciones a cada nodo (I).

Analizando de nuevo las dos ramas de la izquierda, ahora de valores 4 y 6, se puede hacer el mismo razonamiento pero ahora siendo el turno del jugador rojo, que elegirá el menor valor (4). Completando la

otra rama del árbol se llega la figura 3.4.

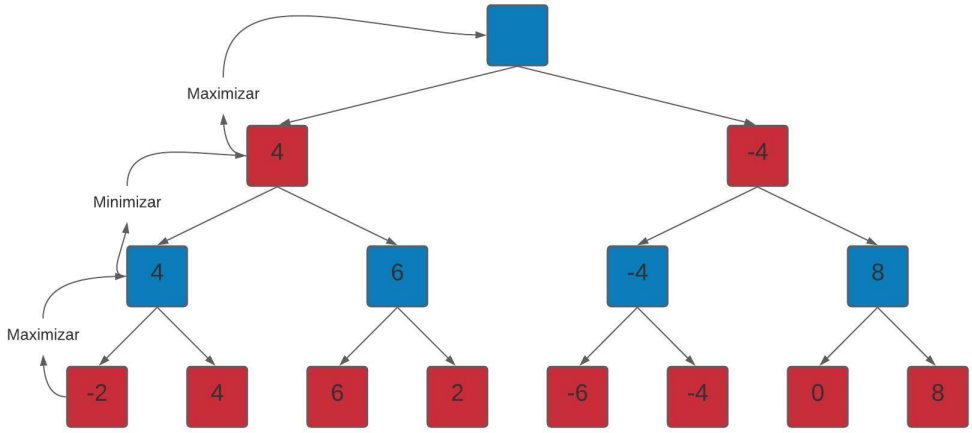

Figura 3.4 Asignación de puntuaciones a cada nodo (II).

Por último, sabiendo que de nuevo es el turno del jugador azul se puede asignar el valor de 4 (es mayor que el valor de la otra rama, -4) al nodo inicial y ya estaría completado el árbol de juego.

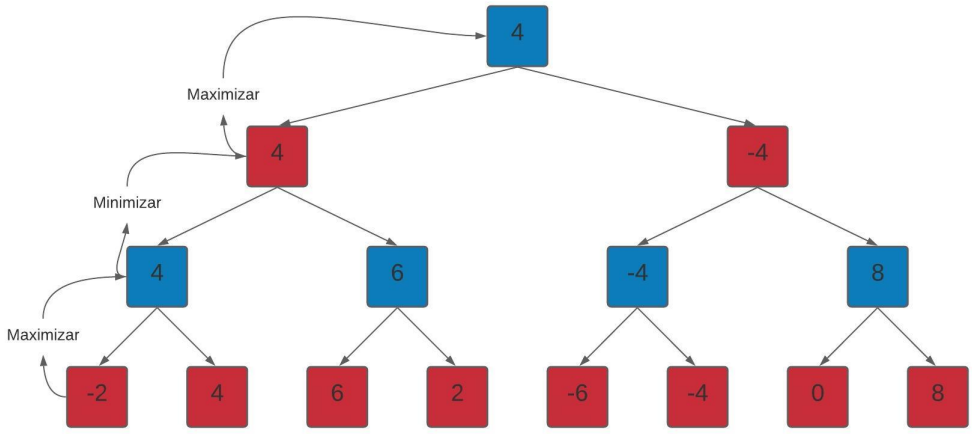

Figura 3.5 Árbol de juego completado.

Se realizan a continuación algunas puntualizaciones importantes:

- Aunque el nodo de mayor puntuación del árbol es 8, el jugador rojo nunca dejará que el juego llegue a ese nodo. Siempre se debe suponer que el oponente juega de forma óptima (juego perfecto).
- En el ejemplo solo hay dos posibles movimientos para cada jugador pero, en general, pueden haber más opciones. En este caso se deben analizar todas las posibilidades, aumentando la complejidad del árbol. En el tres en raya, por ejemplo, el primer jugador tiene 9 posibles movimientos.
- Se han utilizado en el ejemplo puntuaciones aleatorias dadas. En un juego real estas puntuaciones se deben calcular.

3.2 Función *Minimax*

La idea básica del algoritmo *Minimax* consiste en generar todas las posibles situaciones sucesivas hasta un cierto límite (o no, si la complejidad del juego lo permite) partiendo de la situación del tablero actual. A

continuación, se evalúa lo bueno que es el estado al que se ha llegado, y se elige el mejor movimiento para el jugador que está usando el *Minimax*, llevando los valores hacia arriba en el árbol de juego hasta llegar a la situación actual. En la evaluación normalmente se suelen asignar valores positivos a situaciones ventajosas para el jugador que usa el algoritmo (*maximizador*) y valores negativos para su oponente (*minimizador*).

La función *Minimax*, como se ve en el código 3.1, recibe como argumentos de entrada la situación actual del tablero, la profundidad, es decir, el número de movimientos hacia adelante que se están analizando, la variable que indica si es el turno del jugador *maximizador* o del *minimizador* y la lista de posibles movimientos (casillas libres). Primero se comprueba si la partida ha acabado, ya sea en victoria, derrota o empate, y caso afirmativo se asigna una puntuación de +100, -100 o 0, respectivamente. A esta puntuación se le ha hecho una corrección gracias al parámetro de la profundidad. Si se resta dicha profundidad a +100, las victorias en menos movimientos tienen más valor, y por tanto serán escogidas antes que las victorias en partidas más largas. En el caso de las derrotas, si se puntúan como -100 más la profundidad, se da preferencia a las derrotas en partidas largas frente a derrotas rápidas.

Si no es posible analizar el árbol completo, se puede evaluar también lo buena que es una situación intermedia. Esto ocurre al aumentar de tamaño el tablero, por ejemplo a 4×4, lo que eleva el numero de posibles situaciones a analizar de $9! \sim 10^5$ a $16! \sim 10^{13}$ en el caso mas desfavorable (el primer movimiento, cuando el tablero está vacío), haciendo imposible su ejecución en un tiempo razonable para una aplicación.

En los casos de cuatro y cinco en raya, si se ha superado el valor límite de la profundidad y la partida no ha acabado, se asigna una puntuación de 0 y no se analizan más movimientos.

Código 3.1 Función *Minimax* (I).

```
function mejorPuntuacion = minimax(tableroaux,prof,isMax,disponibleaux)
    [ganador,a]=compruebaGanador(tableroaux);
    if ganador==1
        mejorPuntuacion=-100+prof;
    elseif ganador==-1
        mejorPuntuacion=100-prof;
    elseif a==2 && ganador==0
        mejorPuntuacion=0;
    elseif n==4 && prof>4 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    elseif n==5 && prof>3 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    end
```

Si la partida no ha acabado y es el turno del jugador *maximizador* (código 3.2), se busca la puntuación más alta que se puede obtener. Para ello se inicializa la mejor puntuación a menos infinito y para obtener la puntuación de cada posible movimiento se realiza una llamada recursiva a la función *Minimax*, pasándole como argumentos el nuevo estado del tablero, la profundidad más uno, la nueva lista de casillas libres y "false" ya que el siguiente movimiento será del oponente. El hecho de que la función *Minimax* sea recursiva es clave en el desarrollo y comprensión del programa, y es lo que permite llevar hacia arriba en el árbol de juego las puntuaciones de los estados finales por los estados intermedios hasta llegar al estado actual para poder tomar una decisión. Una vez se calcula la puntuación del movimiento, si es mayor que la más alta hasta el momento se actualiza la mejor puntuación y se guarda también qué movimiento es.

Código 3.2 Función *Minimax* (II).

```
    if isMax==true && a==1 %si le toca mover a la maquina
        mejorPuntuacion=-Inf;
        for k=1:length(disponibleaux)
            tableroaux(disponibleaux(k))=-1;
            sig_disponible =find(tableroaux==0);
```

```
            puntuacion=minimax(tableroaux,prof+1, false , sig_disponible );
            tableroaux ( disponibleaux (k))=0;
            if puntuacion>mejorPuntuacion
                mejorPuntuacion=puntuacion;
                mejorMov=disponibleaux(k);
            end
        end
```

Si es el turno del jugador *minimizador* se procede esencialmente de la misma manera, como se puede ver en el código 3.3. Se inicializa la puntuación mínima a más infinito, se calcula la puntuación de los posibles movimientos y se actualiza en caso de ser más baja.

Código 3.3 Función *Minimax* (III).

```
    elseif isMax==false && a==1 %si le toca mover al usuario
        mejorPuntuacion=Inf;
        for k=1:length( disponibleaux )
            tableroaux ( disponibleaux (k))=1;
            sig_disponible =find( tableroaux ==0);
            puntuacion=minimax(tableroaux,prof+1,true , sig_disponible );
            tableroaux ( disponibleaux (k))=0;
            if puntuacion<mejorPuntuacion
                mejorPuntuacion=puntuacion;
                mejorMov=disponibleaux(k);
            end
        end
    end
end
```

Sin embargo, la finalidad del algoritmo es encontrar el movimiento óptimo, no la puntuación óptima. Por ello, la primera llamada a la función *Minimax* se debe hacer desde una función similar a ella pero con argumento de salida una posición, la óptima, en vez de una puntuación. Así, en el código 3.4 se observa una estructura similar a la ya descrita, teniendo en cuenta que la función se llamará en el turno del jugador *maximizador*.

Código 3.4 Función Movimiento Óptimo.

```
function casilla = movOptimo
    mejorPuntuacion=-Inf;
    for k=1:length( disponibleaux )
        tableroaux ( disponibleaux (k))=-1;
        sig_disponible =find( tableroaux ==0);
        puntuacion=minimax(tableroaux,0, false , sig_disponible );
        tableroaux ( disponibleaux (k))=0;
        if puntuacion>mejorPuntuacion
            mejorPuntuacion=puntuacion;
            mejorMov=disponibleaux(k);
        end
    end
    casilla =mejorMov;
    pulsa2=1;
end
```

Para comprender mejor el funcionamiento del algoritmo, se muestra a continuación un árbol de juego del tres en raya. Se parte de un estado intermedio en la partida (figura 3.6) para no complicar innecesariamente su explicación, y se supone que le toca mover al jugador *maximizador*.

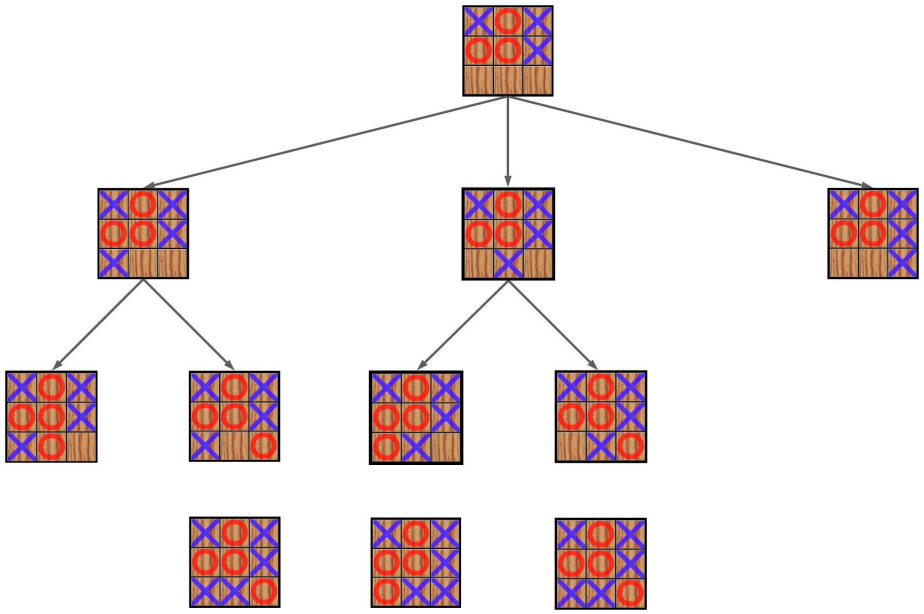

Figura 3.6 Generación del árbol de juego en el tres en raya.

Si se observa la rama derecha se consigue una victoria con un solo movimiento del jugador, por tanto la puntuación es +100 y no se sigue analizando al haber acabado la partida. La rama izquierda tiene a su vez dos ramas, la primera termina con una victoria del oponente de puntuación -99 y la otra en un empate (puntuación 0). Por tanto, el jugador *minimizador* escogerá la rama de puntuación menor, -99. La rama central también se subdivide en otras dos. La primera una victoria en el último movimiento (+98) y la segunda en un empate(0). El jugador *minimizador*, de nuevo, elegirá la menor puntuación de ambas, es decir, el 0. Subiendo un nivel más en el árbol, las tres ramas iniciales tiene unas puntuaciones de -99, 0 y +100. Como el jugador *maximizador* busca la mayor puntuación se escogerá la rama derecha, con una puntuación de +100.

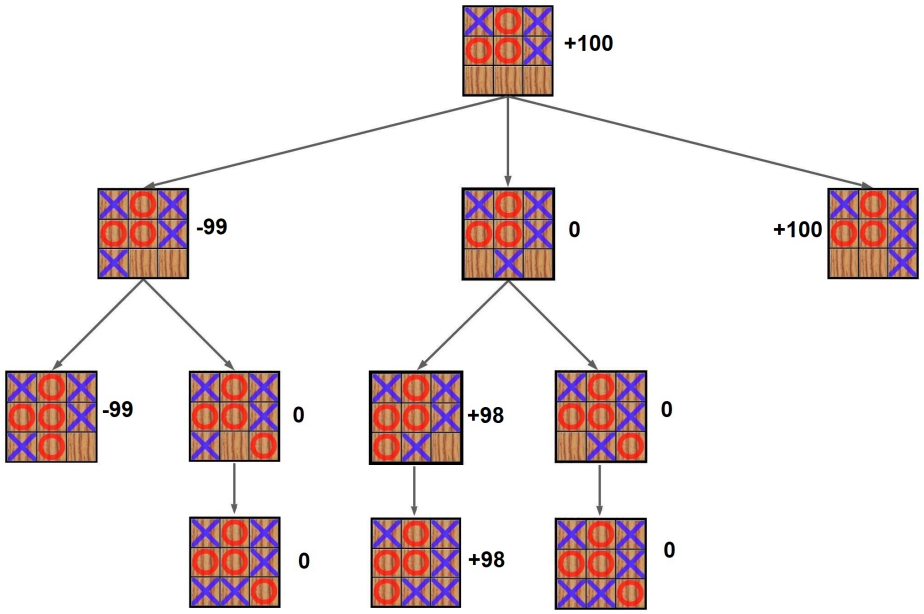

Figura 3.7 Asignación de puntuaciones en el tres en raya.

Cabe destacar que el jugador *maximizador* también podría haber ganado por el camino de la rama central, pero el algoritmo *Minimax* supone que el oponente realiza un juego perfecto, y por tanto no dejará que la partida llegue a ese estado. En cualquier caso, si el oponente no es un jugador perfecto solo beneficia al jugador que usa *Minimax*.

3.3 Función *Poda Alfa–Beta*

La *Poda Alfa–Beta* (en inglés *Alpha–Beta Pruning*) no es en realidad un nuevo algoritmo, sino más bien una técnica que reduce el número de nodos a evaluar por el algoritmo *Minimax*, y por tanto el tiempo de ejecución. Esto permite analizar el árbol de juego mucho más rápido en el caso del tres en raya y en el caso del cuatro y cinco en raya aumentar considerablemente la profundidad con la que se trabaja en cada movimiento.

La idea detrás de la *Poda Alfa–Beta* es no evaluar las ramas del árbol de juego que no se necesiten porque ya exista un movimiento mejor. *Alfa* y *Beta* son los nombres que usualmente se le dan a las dos variables que se introducen respecto al *Minimax*. *Alfa* es el mejor valor que el jugador *maximizador* puede garantizar en un nivel dado o superior y *Beta* es el mejor valor que el jugador *minimizador* puede garantizar en un nivel dado o superior. Ambos valores se van actualizando conforme se evalúa el árbol, de forma que se puedan usar para tomar la decisión de podar una rama del mismo.

Se presenta a continuación un ejemplo de la *Poda Alfa–Beta* para clarificar su funcionamiento.

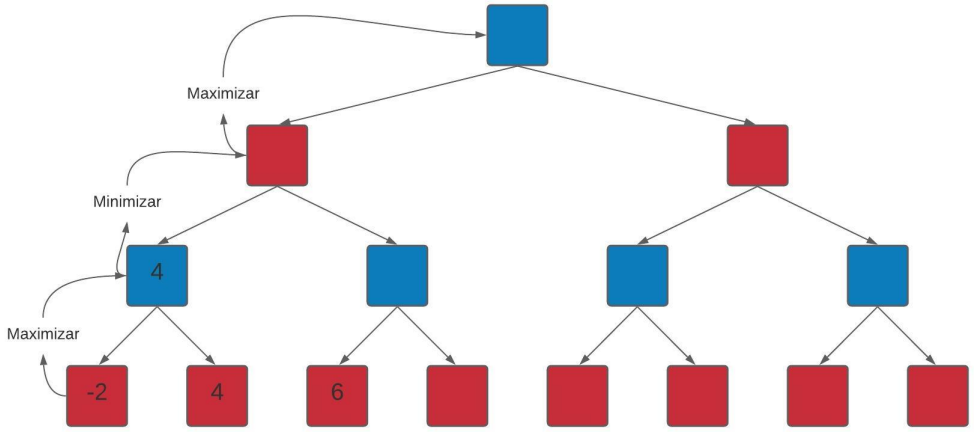

Figura 3.8 Poda de un estado final (I).

En la figura 3.8 se evalúa la rama izquierda, escogiendo el jugador *maximizador* el valor de 4 ya que es mayor que -2, y el primer estado final de la segunda rama, que resulta en un valor de 6. Sabiendo que se elegirá el mayor valor entre 6 y el estado final que falta por analizar, el jugador *maximizador* tiene garantizado un 6 o mayor. Subiendo un nivel más en el árbol (figura 3.9), ahora el jugador *minimizador* tendrá que elegir entre 4 o un valor mayor o igual que 6, escogiendo el 4 sin importar el otro valor. Este razonamiento lleva a pensar que no es necesario evaluar el estado final restante, pudiéndose ahorrar ese tiempo de computación. Dicho de otro modo, es como si ese estado final no existiera, ha sido *podado* del árbol de juego.

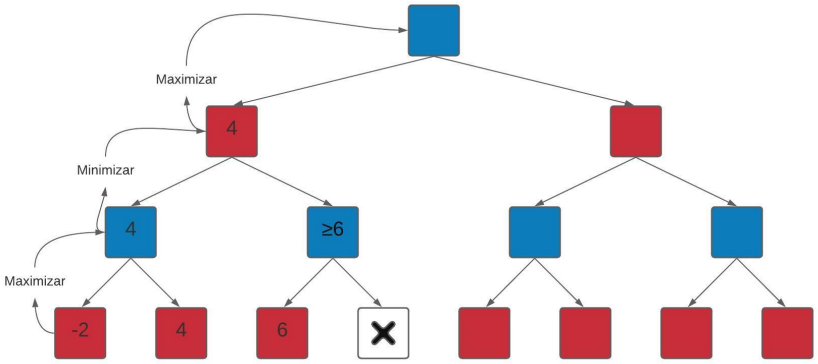

Figura 3.9 Poda de un estado final (II).

Se puede continuar por la otra rama del árbol, como se ve en la figura 3.10. Se tienen los valores de -6 y -4, eligiendo el jugador *maximizador* el -4. De forma análoga a lo explicado anteriormente, el jugador *minimizador* deberá escoger entre -4 u otro valor, siendo en cualquier caso el resultado menor o igual a -4.

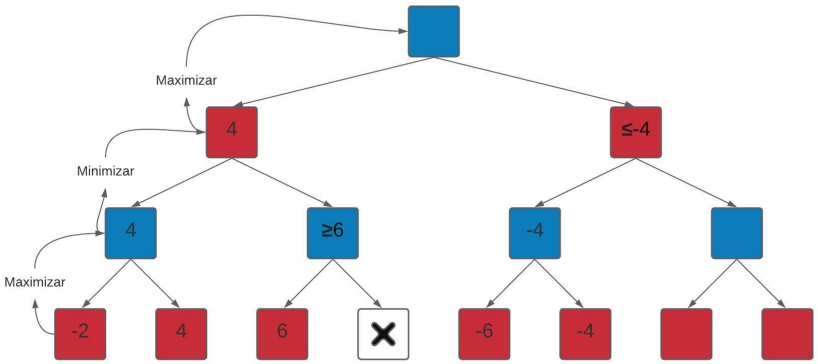

Figura 3.10 Poda de un estado intermedio y de los dos estados finales asociados (I).

Si se observa ahora las dos ramas que parten del estado actual, el jugador *maximizador* tiene que elegir entre 4 o un valor menor o igual que -4. Evidentemente escogerá el 4, por lo que no es necesario evaluar el resto del árbol, que también puede ser *podado* (figura 3.11).

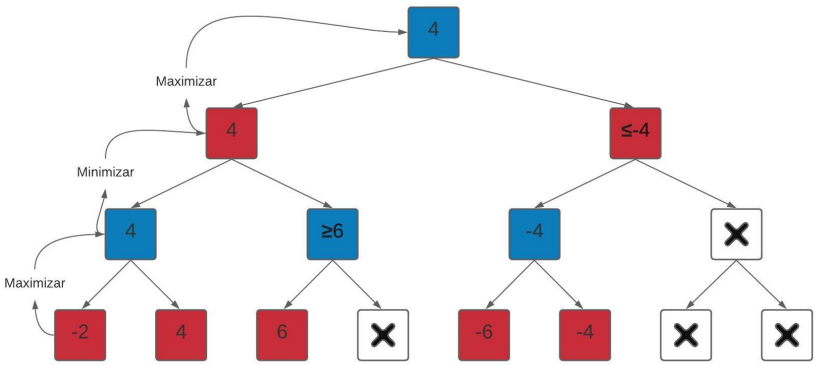

Figura 3.11 Poda de un estado intermedio y de los dos estados finales asociados (II).

Explicado ya el funcionamiento básico de la *Poda Alfa–Beta*, se expone ahora su implementación en el código 3.5.

Código 3.5 Función *Poda Alfa–Beta*.

```
function  casilla  = movOptimo_poda
    mejorPuntuacion=-Inf;
    for k=1:length( disponibleaux )
        tableroaux ( disponibleaux (k))=-1;
        sig_disponible =find( tableroaux ==0);
        puntuacion=minimax_poda(tableroaux,0,-Inf, Inf , false , sig_disponible ) ;
        tableroaux ( disponibleaux (k))=0;
        if  puntuacion>mejorPuntuacion
            mejorPuntuacion=puntuacion;
            mejorMov=disponibleaux(k);
        end
    end
    casilla =mejorMov;
    pulsa2=1;
end

function  mejorPuntuacion = minimax_poda(tableroaux,prof ,alpha ,beta ,isMax,disponibleaux )
    [ganador,a]=compruebaGanador(tableroaux);
    if  ganador==1
        mejorPuntuacion=-100+prof;
    elseif  ganador==-1
        mejorPuntuacion=100-prof;
    elseif  a==2 && ganador==0
        mejorPuntuacion=0;
    elseif  n==4 && prof>8 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    elseif  n==5 && prof>6 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    end
    if  isMax==true && a==1 %si le toca mover a  la  maquina
        mejorPuntuacion=-Inf;
        for k=1:length( disponibleaux )
            tableroaux ( disponibleaux (k))=-1;
            sig_disponible =find( tableroaux ==0);
            puntuacion=minimax_poda(tableroaux,prof+1,alpha ,beta, false , sig_disponible ) ;
            tableroaux ( disponibleaux (k))=0;
            if  puntuacion>mejorPuntuacion
                mejorPuntuacion=puntuacion;
                mejorMov=disponibleaux(k);
            end
            alpha=max(alpha,puntuacion);
            if  beta<=alpha
                break
            end
        end
    elseif  isMax==false  && a==1 %si le toca  mover al  usuario
        mejorPuntuacion=Inf;
        for k=1:length( disponibleaux )
            tableroaux ( disponibleaux (k))=1;
            sig_disponible =find( tableroaux ==0);
            puntuacion=minimax_poda(tableroaux,prof+1,alpha ,beta, true , sig_disponible ) ;
            tableroaux ( disponibleaux (k))=0;
            if  puntuacion<mejorPuntuacion
                mejorPuntuacion=puntuacion;
```

```
                    mejorMov=disponibleaux(k);
                end
                beta=min(beta,puntuacion);
                if beta<=alpha
                    break
                end
            end
        end
```

Las diferencias con la función *Minimax* son mínimas, únicamente las relacionadas a los nuevos parámetros *Alfa* y *Beta*. Se les da un valor inicial de menos infinito e infinito respectivamente. En el caso del jugador *maximizador*, después de evaluar cada estado del tablero se actualiza *Alfa* si la puntuación es mayor, y si *Beta* es menor o igual que *Alfa* se rompe el bucle para dejar de evaluar estados. Si se trata de un movimiento del jugador *minimizador* se procede de forma similar, actualizando *Beta* si es menor que la puntuación del movimiento y rompiendo el bucle si *Beta* es menor o igual que *Alfa*.

4 Programación y resultados

En este capítulo se expone la programación de la aplicación desarrollada. Se tratará de una aplicación totalmente personalizada hecha desde cero escribiendo todas las líneas del código final. Esto permite un mayor control en el diseño y desarrollo de la interfaz que se detallará en la sección 4.2. Otra opción podría haber sido el uso de la herramienta *GUIDE*, que puede generar códigos excesivamente largos y difíciles de leer, o *App Designer*, pero requiere un cierto dominio de la programación orientada a objetos (OOP).

En casos de aplicaciones largas con interfaces complejas con un elevado número de objetos puede ser complicada la gestión de las variables. En esta aplicación se han decidido usar variables anidadas para todos los datos que se vayan actualizando durante la ejecución del programa, asignando nombres descriptivos e intuitivos para facilitar la comprensión del mismo.

Otro problema que suele surgir en aplicaciones de tamaño considerable es pasar variables como argumentos de entrada de las funciones auxiliares. Normalmente se recomienda, y así se ha hecho en este trabajo, usar funciones anidadas que tengan acceso a todas las variables ya definidas por la función que está por encima de ellas. Para ello será necesario inicializar dichas variables en la función principal, como se ve en el código A.1, y en cada llamada a una función auxiliar se podrán actualizar los valores que se deseen y compartirlos con las demás funciones anidadas.

4.1 Objetos multimedia

Matlab permite el tratamiento de imágenes y sonidos digitales, modificándolos en combinaciones de vectores, matrices e hipermatrices. Gracias a esto se han podido crear un fondo para la aplicación, unas casillas vacías y con fichas, y objetos de sonido que se reproducen en determinados momentos.

4.1.1 Imágenes matriciales

Una imagen matricial o digital no es más que una representación bidimensional de una imagen a partir de matrices numéricas, en las que se recoge la información necesaria en forma de píxeles (cada píxel tiene asignado un color) para realizar dicha representación.

Una de las formas de manejar imágenes matriciales en Matlab son las imágenes RGB o *truecolor*, que consisten en una hipermatriz de profundidad tres, donde cada matriz indica la intensidad del color rojo, verde o azul. El hecho de darle este carácter matricial a las imágenes permite un sinfín de operaciones algebraicas que se traducirán en efectos visuales.

En el programa de la aplicación (código A.1) se ha importado la imagen que servirá de fondo de la interfaz gráfica con la orden `imread` para posteriormente visualizarla con `image`, que en realidad está creando un objeto con diversas propiedades (color, transparencias, suavidad, posición, etc.). Cabe destacar que la imagen de fondo de las casillas vacías se obtiene recortando la imagen del fondo, al reducir el numero de filas y de columnas de las tres matrices antes mencionadas para que coincidan con el tamaño de la casilla.

El caso de la imagen de la casillas ocupadas por alguna de las dos fichas es más complejo. Como se ve en el código 4.1, se comienza cargando la imagen y se recorta a 100×100 píxeles. A continuación, se guardan cada una de las tres matrices que forman la imagen en variables separadas para poder operar con ellas. En concreto, en el caso de la ficha **X**, que será azul, el procedimiento es el siguiente:

- Se refuerza el contenido en azul, transformando los píxeles 12 filas por encima y por debajo de las diagonales principal y secundaria a un nivel alto de azul: 250 (el máximo es 255).
- Se suavizan los contenidos rojo y verde en la misma región, dividiendo el valor por tres.
- Se crean unos ejes que después se ocultan en los que poner la nueva imagen ya modificada, que se guarda y se carga con la orden `imread` desde el código principal.

Código 4.1 Generación de las fichas **X**.

```
fondo=imread('madera.jpg');

jj=1:100;
J=jj(ones(1,100),:);
I=J';
fondo2=fondo(1:100,1:100,:);
roja=fondo2(:,:,1);
verde=fondo2(:,:,2);
azul=fondo2(:,:,3);
azul((J-I>-12 & J-I<12)|(J-(100-I+1)>-12 & J-(100-I+1)<12))=250;
verde((J-I>-12 & J-I<12)|(J-(100-I+1)>-12 & J-(100-I+1)<12))=verde((J-I>-12 & J-I<12)|(J-(100-I+1)
    >-12 & J-(100-I+1)<12))/3;
roja((J-I>-12 & J-I<12)|(J-(100-I+1)>-12 & J-(100-I+1)<12))=roja((J-I>-12 & J-I<12)|(J-(100-I+1)>-12
    & J-(100-I+1)<12))/3;
fondo2(:,:,1)=roja;
fondo2(:,:,2)=verde;
fondo2(:,:,3)=azul;
figure
a1=axes('position',[0 0 1 1]);
X=image(fondo2);
axis off
```

Para generar la ficha **O** se ha seguido un proceso completamente análogo, con la excepción de que ahora la ficha será roja. Esto significa que la matriz a reforzar sera la roja y se suavizarán la verde y la azul.

Código 4.2 Generación de las fichas **O**.

```
jj=1:100;
J=jj(ones(1,100),:);
I=J';
fondo2=fondo(1:100,1:100,:);
roja=fondo2(:,:,1);
verde=fondo2(:,:,2);
azul=fondo2(:,:,3);
roja((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I-50).^2<47^2)=250;
verde((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I-50).^2<47^2)=verde((J-50).^2+(I-50).^2>30^2 & (J-50)
    .^2+(I-50).^2<47^2)/3;
azul((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I-50).^2<47^2)=azul((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I
    -50).^2<47^2)/3;
fondo2(:,:,1)=roja;
fondo2(:,:,2)=verde;
fondo2(:,:,3)=azul;
figure
a1=axes('position',[0 0 1 1]);
X=image(fondo2);
axis off
```

4.1.2 Sonido

Matlab también permite el tratamiento del audio digital, es decir, editar, reproducir, grabar e incluso componer sonidos digitalizados. El audio digital se consigue mediante la codificación de una onda de sonido combinando dos procesos: el muestreo y la cuantificación. El muestreo fija la amplitud de la señal a intervalos regulares de tiempo, llamado tasa de muestreo. La cuantificación, por su parte, convierte dichas amplitudes en valores numéricos dentro del conjunto de niveles preestablecidos.

La orden `audioread` permite leer archivos de sonido y generar el vector (son dos si el sonido es estereo) de la señal digital de audio y la frecuencia de muestreo. Puesto que los sonidos de la aplicación deben ser reproducidos más de una vez, se ha decidido crear objetos de sonido haciendo uso de la instrucción `audioplayer`, para posteriormente reproducirlos en su momento con la orden `play`. En total son cuatro los sonidos que se tienen, uno para la colocación de las fichas, similar a un pequeño golpe sobre madera, y una música de victoria, de derrota o de empate según acabe el juego.

Código 4.3 Generación de los objetos de sonido.

```
%Objetos de Sonido
[y1,Fs1]=audioread('poneficha.mp3',[1,22050/3]);
poneficha=audioplayer(y1,Fs1);

[y2,Fs2]=audioread('victoria.mp3');
victoria=audioplayer(y2,Fs2);

[y3,Fs3]=audioread('derrota.wav');
derrota=audioplayer(y3,Fs3);

[y4,Fs4]=audioread('empate.wav');
empate=audioplayer(y4,Fs4);
```

Es importante resaltar que Matlab permite grabar sonidos directamente desde el micrófono del ordenador, pero esta opción se descartó por la baja calidad del mismo, que producía un sonido peor a los que se pueden encontrar fácilmente en Internet y son de uso libre.

4.2 Interfaz gráfica de usuario

Una interfaz gráfica de usuario se puede definir como un programa que actúa de intermediario entre la máquina y el usuario mediante un conjunto de objetos que facilitan la información y las posibles acciones disponibles. El objetivo en esta sección será crear un entorno visual sencillo y atractivo para poder comunicarse con la aplicación, para lo cual se recurre a un conjunto de eventos y propiedades evento–funcionales. Un evento respecto a un objeto gráfico es un aviso que se envía a las propiedades evento–funcionales cuando hay algún cambio en el objeto, y dichas propiedades ejecutan la función que tengan asociada. Un claro ejemplo de esto puede ser devolver las coordenadas de un punto al hacer click con el ratón sobre una figura, o bien mostrar un mensaje cuando se cierre una ventana de figura.

Lo primero será crear la ventana de la aplicación, donde aparecerán todos los aspectos gráficos de la misma. El objeto *Figure* creado tiene una gran cantidad de propiedades especificas, algunas de las cuales se han modificado como se ve en el código 4.5. Se ha cambiado el nombre de la ventana y su posición (siguiendo el formato LBHW: Left, Bottom, Width, Height), y se han quitado tanto la barra de tareas como el número de figura para mejorar la presentación. La visibilidad se mantiene desactivada hasta que se crean todos los objetos y se posiciona la ventana (una práctica muy común) y también se impide cambiar el tamaño de la figura, con el objetivo de que la aplicación se vea igual en cualquier ordenador. Por último, se decide añadir una propiedad evento–funcional para el evento de cerrar la ventana. Esta propiedad llamará a una función (código 4.4) que borra todos los objetos *Timer* (más adelante en esta sección se explican estos objetos y su utilidad) que pudieran permanecer activos cuando se cierre la figura, evitando posibles mensajes de error.

Código 4.4 Función llamada al cerrar la aplicación.

```
function cierra (~,~)
    delete( timerfindall )
end
```

Para poner de fondo la imagen que se ha importado con `imread`, se necesitan crear unos ejes con el objeto *Axes* que ocupen toda la ventana, dibujar en ellos la imagen y hacerlos invisibles para una mejor estética. Para acabar este bloque, se ha utilizado la orden `movegui` para centrar la ventana en la pantalla.

Código 4.5 Creación de los objetos *Figure* y *Axes*.

```
%Figura Principal
margenlat=30;
margeninf=65;
borde=14;
raya=6-(n-3)*2;

fondo=imread('madera.jpg');

hfig=figure('Name','Tres en raya' ,...
    'Position',[1 1 1200 640],...
    'Color',[0.65 0.65 0.65],...
    'menubar','none' ,...
    'numbertitle','off' ,...
    'Visible','off' ,...
    'Resize','off' ,...
    'DeleteFcn',@cierra);
ejes=axes('position',[0 0 1 1]);
X=image(fondo);
axis off
movegui(hfig,'center');
```

Existe un tipo de objetos típico de las interfaces gráficas de usuario, los *UIObjects*. Estos objetos son de vital importancia en el desarrollo de aplicaciones ya que incluyen cuadros de texto, botones, menús desplegables, etc. Aunque algunos de estos objetos se crean con una instrucción propia, en este trabajo se han usado mayoritariamente los generados con `uicontrol` y determinados gracias a su propiedad "Style".

Por ejemplo, en la aplicación debe haber un texto que indique en qué partida del juego se está en un momento dado. Para ello, se han creado cuatro *UIControl* de tipo "text" configurando el color de fondo, el tamaño y tipo de letra, la alineación y ajustando la posición dentro de la ventana de figura. Por el momento no se muestran ya que el número de partidas por juego es un dato que puede introducir el usuario, y hasta que no lo haga se mantienen invisibles al igual que otros objetos que se verán más adelante.

Código 4.6 Creación de los objetos de texto del contador de partidas.

```
fondoopciones=[215 164 108]/255;
fondotexto=[235 190 120]/255;
h=1;
%Texto contador partidas
textopartidas =uicontrol('Style','text' ,...
    'BackgroundColor',fondotexto ,...
    'String','PARTIDA',...
    'Position' ,[150,600,150,35],...
    'HorizontalAlignment','center' ,...
    'FontSize' ,20,...
    'Visible','off');
textonum1=uicontrol('Style','text' ,...
    'BackgroundColor',fondotexto ,...
    'Position' ,[285,600,40,35],...
```

```
      'HorizontalAlignment','center',...
      'FontSize',20,...
      'FontWeight','bold',...
      'Visible','off');
textode=uicontrol('Style','text',...
      'BackgroundColor',fondotexto,...
      'String','DE',...
      'Position',[325,600,50,35],...
      'HorizontalAlignment','center',...
      'FontSize',20,...
      'Visible','off');
textonum2=uicontrol('Style','text',...
      'BackgroundColor',fondotexto,...
      'Position',[375,600,40,35],...
      'HorizontalAlignment','center',...
      'FontSize',20,...
      'FontWeight','bold',...
      'Visible','off');
```

Figura 4.1 Aspecto gráfico de los objetos de texto del contador de partidas.

Otra opción importante a añadir es la elección de la estrategia de respuesta del programa. Como ya se vio en el capítulo 3, se han implementado el algoritmo *Minimax* básico y con la modificación de la *Poda Alfa–Beta*. Además de esto, se decide incluir una opción de movimiento aleatorio, en el que el programa elige una de las casillas libres al azar para realizar su movimiento. Estas tres opciones están recogidas en un menú desplegable tipo "popupmenu" en el que se puede seleccionar cualquiera de ellas, acompañado de otro *UIControl* tipo "text" como titulo del menú o a modo aclaratorio. Aquí lo más interesante, además de modificar propiedades ya conocidas como el color, el tamaño o la posición, es que se ha incluido una propiedad evento–funcional que es simplemente una función que guarda la opción elegida en una variable.

De manera similar se crea otro menú que permite elegir el jugador inicial, existiendo también tres opciones: el usuario, la máquina o aleatorio. La programación es análoga al primer menú, y se pueden ver ambos en el código 4.7

Código 4.7 Creación de los menú de estrategia y de elección de jugador inicial.

```
%Panel derecha
margensup=20;
margendcha=20;
anchopanel=580;
altopanel =440;
xpanel=1200-margendcha-anchopanel;
ypanel=640-margensup-altopanel;

%Texto estrategia
altopanel = altopanel -45;
textoestrategia =uicontrol('Style','text',...
      'BackgroundColor',fondotexto,...
      'String','ESTRATEGIA DE RESPUESTA',...
      'Position',[ xpanel+40,ypanel+ altopanel ,500,35],...
      'HorizontalAlignment','center',...
      'FontSize',20,...
      'FontWeight','bold');
lista1 =uicontrol('style','popupmenu',...
      'position',[ xpanel+40,ypanel+ altopanel -50,500,35],...
```

```
        'FontSize' ,16,...
        ' string ' ,{ ' Aleatorio ' , 'Minimax','Poda' },...
        'BackgroundColor',fondoopciones ,...
        ' callback ' , @elige1);

 %Texto jugador  inicial
textocomienzo=uicontrol(' Style ' , ' text ' ,...
        'BackgroundColor',fondotexto ,...
        ' String ' , 'ELECCIÓN JUGADOR INICIAL',...
        ' Position ' ,[ xpanel+40,ypanel+3*altopanel /4,500,35],...
        'HorizontalAlignment' , ' center ' ,...
        'FontSize' ,20,...
        'FontWeight','bold');
 lista2 =uicontrol(' style ' , 'popupmenu',...
        ' position ' ,[ xpanel+40,ypanel+3*altopanel /4-50,500,35],...
        'FontSize' ,16,...
        ' string ' ,{ ' Aleatorio ' , 'Usuario' , 'Máquina' },...
        'BackgroundColor',fondoopciones ,...
        ' callback ' , @elige2);
```

Figura 4.2 Aspecto gráfico de los menús de estrategia y de elección de jugador inicial.

Siguiendo con las opciones que puede elegir el usuario previamente a empezar la partida, se encuentran la elección del número de partidas, como ya se comentó antes, y el tiempo de respuesta del usuario. En este caso se añaden dos *UIControl* tipo "edit" en los que se introducen ambos valores, acompañados de otros dos objetos *UIControl* de tipo "text" como títulos.

Código 4.8 Creación de los objetos de texto editables.

```
 %Texto eleccion  tiempo  de  respuesta
textocomienzo=uicontrol(' Style ' , ' text ' ,...
        'BackgroundColor',fondotexto ,...
        ' String ' , 'TIEMPO DE RESPUESTA',...
        ' Position ' ,[ xpanel+40,ypanel+ altopanel /2-20,420,35],...
        'HorizontalAlignment' , ' center ' ,...
        'FontSize' ,20,...
        'FontWeight','bold');
entradatiempo=uicontrol(' Style ' , ' edit ' ,...
        ' Position ' ,[ xpanel+480,ypanel+ altopanel /2-20,60,35],...
        'BackgroundColor',fondoopciones ,...
        'FontSize' ,20,...
```

```
      'FontWeight','bold' ,...
      ' String ',num2str(T));

%Texto eleccion numero partidas
textocomienzo=uicontrol(' Style ',' text ' ,...
      'BackgroundColor',fondotexto ,...
      ' String ','Nº DE PARTIDAS POR JUEGO',...
      ' Position ',[ xpanel+40,ypanel+ altopanel /2-80,420,35],...
      'HorizontalAlignment',' center ' ,...
      'FontSize' ,20,...
      'FontWeight','bold');
 entradapartidas =uicontrol(' Style ',' edit ' ,...
      ' Position ',[ xpanel+480,ypanel+ altopanel /2-80,60,35],...
      'BackgroundColor',fondoopciones ,...
      'FontSize' ,20,...
      'FontWeight','bold' ,...
      ' String ',num2str( totalpartidas ));
```

Figura 4.3 Aspecto gráfico de los objetos de texto editables.

Otra elección clave en el juego serán las dimensiones del tablero. Haciendo uso de un menú desplegable de tipo "popupmenu" como los ya usados en el código 4.7 se pueden elegir las dimensiones 3×3, 4×4 o 5×5. En este caso la función asociada a la propiedad evento–funcional cambia el valor del parámetro que define el tamaño del tablero dependiendo de la opción seleccionada.

Código 4.9 Creación del menú para elegir el tamaño del tablero.

```
%Texto eleccion tamaño tablero
 textotablero =uicontrol(' Style ',' text ' ,...
      'BackgroundColor',fondotexto ,...
      ' String ','TAMAÑO DEL TABLERO',...
      ' Position ',[ xpanel+40,ypanel+ altopanel /2-140,420,35],...
      'HorizontalAlignment',' center ' ,...
      'FontSize' ,20,...
      'FontWeight','bold');
 lista3 =uicontrol(' style ','popupmenu',...
      ' position ',[ xpanel+480,ypanel+ altopanel /2-140,60,35],...
      'FontSize' ,16,...
      ' string ',{ '3x3','4x4','5x5' },...
      'BackgroundColor',fondoopciones ,...
      ' callback ',@elige3);
```

Figura 4.4 Aspecto gráfico del menú para elegir tamaño del tablero.

El siguiente elemento de la interfaz es realmente importante: el botón de comienzo de partida. Se trata de un objeto *UIControl* de tipo "pushbutton" que tiene asociado una función que no empieza directamente la partida, sino que realiza unas comprobaciones previas. Si no detecta ningún error sí comienza la partida.

Código 4.10 Creación del botón de comienzo de partida.

```
%Texto empezar partida
textoempezarpartida =uicontrol(' Style ' ,' text ' ,...
    'BackgroundColor',fondotexto ,...
    ' String ' ,'COMIENZO PARTIDA',...
    ' Position ' ,[ xpanel+40,ypanel -50,400,35],...
    'HorizontalAlignment' ,' center ' ,...
    'FontSize' ,20,...
    'FontWeight','bold');
botoncomienzo=uicontrol(' Style ' ,' pushbutton ' ,...
    'Callback' ,@comprobaciones_previas ,...
    ' position ' ,[ xpanel+460,ypanel -63,61,61],...
    'BackgroundColor',fondoopciones ,...
    ' String ' ,'START',...
    'FontWeight','bold' ,...
    'FontSize' ,12);
```

Figura 4.5 Aspecto gráfico del botón de comienzo de partida.

Para llevar la cuenta de las partidas ganadas por cada jugador se decide incluir un marcador que consta de cinco objetos *UIControl* de tipo "text", dos de los cuales empiezan vacíos para posteriormente cambiar en caso de que el jugador correspondiente gane una partida. Resaltar que el número de victorias del usuario se muestra en color azul y el número de victorias de la máquina en rojo, para que sean más fácilmente distinguibles.

Código 4.11 Creación de los objetos que forman el marcador del juego.

```
%Texto puntuacion
textopuntuacion =uicontrol(' Style ' ,' text ' ,...
    'BackgroundColor',fondotexto ,...
    ' String ' ,'PUNTUACIÓN',...
    ' Position ' ,[ xpanel+anchopanel/2-100,ypanel -100,200,35],...
    'HorizontalAlignment' ,' center ' ,...
    'FontSize' ,20,...
    'FontWeight','bold');
textousuario =uicontrol(' Style ' ,' text ' ,...
    'BackgroundColor',fondotexto ,...
    ' String ' ,'USUARIO',...
    ' Position ' ,[ xpanel+40,ypanel -150,150,35],...
    'HorizontalAlignment' ,' center ' ,...
    'FontSize' ,20,...
    'FontWeight','bold');
textopuntosusuario =uicontrol(' Style ' ,' text ' ,...
    ' Position ' ,[ xpanel+210,ypanel -150,40,35],...
    'HorizontalAlignment' ,' center ' ,...
    'FontSize' ,20,...
    'FontWeight','bold' ,...
    'BackgroundColor',fondoopciones ,...
    'ForegroundColor','blue');
```

```
textomaquina=uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'String','MÁQUINA',...
    'Position',[xpanel+330,ypanel-150,150,35],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'FontWeight','bold');
textopuntosmaquina=uicontrol('Style','text',...
    'Position',[xpanel+anchopanel-80,ypanel-150,40,35],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'FontWeight','bold',...
    'BackgroundColor',fondoopciones,...
    'ForegroundColor','red');
```

Figura 4.6 Aspecto gráfico de los objetos que forman el marcador del juego.

Por último, se necesita indicar en cada momento a quién le toca jugar y en el caso de que sea el turno del usuario cuánto tiempo le queda para realizar el movimiento. Tras varias pruebas en las que no quedaba muy claro a quién le tocaba mover, se decide poner un objeto *UIControl* tipo "text" con el mensaje *TURNO DE LA MÁQUINA* o bien *ES TU TURNO* y el tiempo restante, en función de si le toca a un jugador u otro. Ahora también se encuentra desactivada la opción de visibilidad, ya que el usuario puede variar el tiempo restante y el jugador que inicia la partida, y por tanto hasta el comienzo de la misma se ha decidido no mostrar el objeto.

Código 4.12 Creación del texto de cuenta atrás.

```
%Texto cuenta atras
textotiemporest =uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'String','TIEMPO RESTANTE',...
    'Position',[130,10,260,40],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'Visible','off');
textotimer =uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'Position',[390,10,50,40],...
    'String',num2str(t),...
    'FontSize',20,...
    'FontWeight','bold',...
    'HorizontalAlignment','center',...
    'Visible','off');
```

Figura 4.7 Aspecto gráfico del texto de cuenta atrás.

Se adjunta para acabar esta sección una imagen de la ventana de la aplicación al abrirla. Cabe recordar que no todos los objetos son visibles desde el principio, como ya se ha mencionado, y que se debe esperar al comienzo de la partida para mostrarlos.

Figura 4.8 Aspecto gráfico de aplicación antes de comenzar la partida.

4.3 Programación de la partida

Una vez se han detallado los objetos que forman la interfaz gráfica de la aplicación, es momento ahora de desarrollar la propia aplicación. Como ya se comentó al principio del capítulo, se han usado funciones anidadas a la función principal, cada una con una finalidad distinta. A lo largo de esta sección se expondrán y explicarán cada una de ellas.

La función *creatablero* (código 4.13) emplea una matriz de celdas (de las dimensiones del tablero) y coloca en cada una un objeto *UIControl* de tipo "pushbutton" con la imagen del fondo de la aplicación recortada al mismo tamaño que la propia casilla. En la creación de los botones no se asigna ninguna función de llamada (propiedad evento–funcional), ya que hasta que el juego empiece no se deben poner fichas. En la misma función se genera un panel de color negro con la orden UIPANEL, que servirá como borde del tablero y separación de las casillas, ya que se sitúa detrás de los botones. Es de importancia mencionar que el tablero actualiza de forma instantánea sus dimensiones al seleccionarlas en el menú correspondiente.

Código 4.13 Creación del tablero.

```
function creatablero

    ladocasilla =480/n;
    %Panel (tablero)
    p=uipanel('Units','pixels',...
        'Position',[margenlat-borde/2,...
        margeninf-borde/2,...
        ladocasilla*n+2*borde+(n-1)*raya,...
        ladocasilla*n+2*borde+(n-1)*raya],...
        'BorderType','none',...
        'BackgroundColor','black');
    %Botones (casillas tablero)
    for ii=1:n
```

```
        for jj=1:n
            b{ii,jj}=uicontrol('Style','pushbutton',...
                'Callback',[],...
                'position',[margenlat+borde/2+(jj-1)*ladocasilla+(jj-1)*raya,...
                margeninf+borde/2+(n-ii)*ladocasilla+(n-ii)*raya,...
                ladocasilla, ladocasilla],...
                'BackgroundColor','white',...
                'CData',fondo(1:ladocasilla,1:ladocasilla,:));
        end
    end
end
```

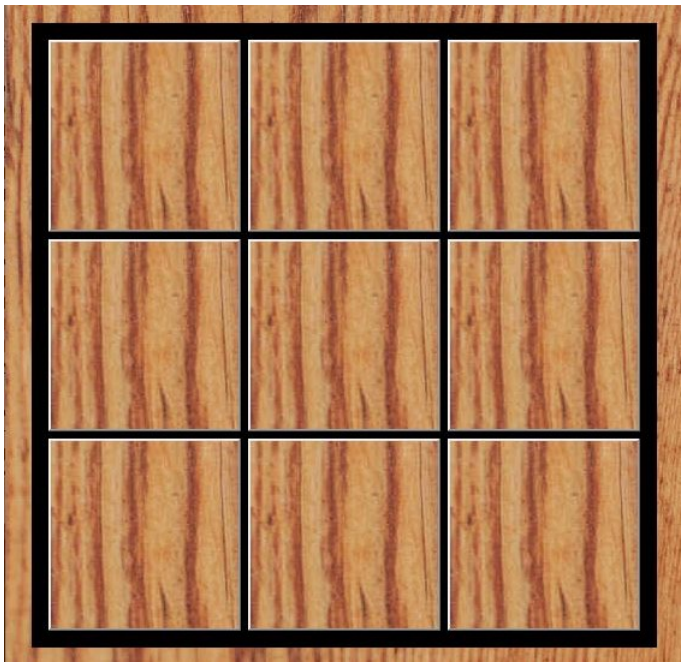

Figura 4.9 Botones y panel de fondo para el caso del tablero 3×3.

El código 4.14 es la función de llamada del botón de comienzo de partida. En dicha función se controla que no haya errores en los datos que puede introducir el usuario, es decir, el número de partidas por juego y el tiempo restante. Con errores se hace referencia a que la cadena introducida no sea un número positivo o que simplemente esté vacía. En caso de detectar algún error al presionar el botón de comienzo se muestra un mensaje de error creado con la orden `errordlg`, que crea un cuadro diálogo personalizable. Una vez se cierra manualmente el mensaje habrá que corregir el error y presionar de nuevo el botón de comienzo. Resaltar por último el uso de `waitfor`, que paraliza la ejecución de todo el programa hasta que se cierra el mensaje de error.

Código 4.14 Comprobaciones previas.

```
function comprobaciones_previas(~,~)
    e=[];
    v3=isempty(entradatiempo.String);
    v4=isempty(entradapartidas.String);
    v5=str2double(entradatiempo.String);
    v6=str2double(entradapartidas.String);
    if v3==1
        e=errordlg('No has puesto tiempo de respuesta','Error');
        waitfor(e)
    end
```

```
    if v4==1
        e=errordlg('No has puesto numero de partidas','Error');
        waitfor(e)
    end
    if isnan(v5) && v3==0
        e=errordlg('Tiempo de respuesta no valido','Error');
        waitfor(e)
    end
    if isnan(v6) && v4==0
        e=errordlg('Numero de partidas no valido','Error');
        waitfor(e)
    end
    if isempty(e)
        comienzo
    end
end
```

En caso de que no haya errores se llama a la función *comienzo*, que como se puede ver en el código 4.15 es una puesta a punto general para que pueda comenzar la partida sin contratiempos. Algunos de los aspectos a destacar son:

- Se desactivan las propiedades evento–funcionales del botón de comienzo y de los menús desplegables, porque el juego ya ha comenzado y no se pueden cambiar las opciones en mitad de un juego.
- Se crea la matriz numérica *tablero* y se inicializa a la matriz nula, ya que todas las casillas están vacías. Se trata de una variable anidada que se va actualizando con 1 (ficha **X**) y -1 (ficha **O**). Esta variable es clave en muchas otras funciones, ya que trabajar con la matriz de celdas puede llegar a ser bastante complicado.
- Se extrae de los objetos de texto el número de partidas y el tiempo de respuesta elegidos por el usuario. Por defecto, estas variables están inicializadas a 3 partidas y 20 segundos.
- Se hacen visibles los objetos de texto que indican el numero actual y total de partidas, así como el marcador con las puntuaciones (ceros inicialmente) de cada jugador.
- Se adaptan mediante la orden `imresize` las imágenes matriciales de las casillas con ambas fichas.
- Se toma un camino u otro en función del cuál sea el primer jugador. Básicamente consiste en mostrar el mensaje que indica el turno y el tiempo restante si procede con el color correspondiente y llamar a una de las dos siguientes funciones (códigos 4.16 y 4.20).

Código 4.15 Comienzo de la partida.

```
function comienzo
    botoncomienzo.Callback=[];

    tablero=zeros(n);

    str1=get(entradatiempo,'string');
    T=str2double(str1);
    t=T;
    entradatiempo.Style='text';
    str2=get(entradapartidas,'string');
    entradapartidas.Style='text';
    totalpartidas=str2double(str2);
    numeropartida=1;
    textonum1.String=num2str(numeropartida);
    textonum2.String=str2;

    lista1.Callback=[];
    lista2.Callback=[];
    lista3.Callback=[];
```

```
    textopartidas . Visible='on';
    textode . Visible='on';
    textonum1. Visible='on';
    textonum2. Visible='on';

    textopuntosusuario . String=num2str(puntosusuario);
    textopuntosmaquina . String=num2str(puntosmaquina);


    xfoto=imread('xficha2 .jpg');
    ximagen=imresize(xfoto ,[ ladocasilla  ladocasilla ]);
    ofoto=imread('oficha2 .jpg');
    oimagen=imresize(ofoto ,[ ladocasilla  ladocasilla ]);

    jugadorActual=primerJugador;
    if jugadorActual==1
        textotiemporest . Visible='on';
        textotiemporest . String='ES TU TURNO';
        textotiemporest .ForegroundColor='blue';
        textotimer . Visible='on';
        textotimer . String=num2str(t);
        textotimer .ForegroundColor='blue';
        turno_X
    else
        textotiemporest . Visible='on';
        textotiemporest . Position =[100,10,350,40];
        textotiemporest . String='TURNO DE LA MÁQUINA';
        textotiemporest .ForegroundColor='red';
        textotimer . Visible='off';
        pause(1)
        turno_O
    end
end
```

En el código 4.16 se gestiona todo lo relacionado con el turno del usuario. Quizás lo mas interesante es la utilización de un objeto *Timer*, que permite planificar la ejecución de una serie de tareas un número cualquiera de veces. En concreto, la tarea que se lleva a cabo es restar uno al tiempo restante del temporizador. Las propiedades más importantes del objeto *Timer* son el periodo (tiempo entre una ejecución y la siguiente, es un segundo), el número de veces que se ejecuta la tarea (hasta llegar a cero, es decir, el mismo número de segundos del tiempo total de respuesta), y sobre todo las dos propiedades evento–funcionales:

- "TimerFcn". Se ejecuta cada segundo y consiste simplemente en restar uno al número de segundos restantes y actualizar la propiedad "String" del texto de la cuenta atrás.
- "StopFcn". Se ejecuta al parar el *Timer* y puede ser alcanzada de dos formas: cuando el usuario marca una casilla o si se ha acabado el tiempo, en cuyo caso se coloca la ficha automáticamente de forma aleatoria llamando a la función *movAleatorio* (código 4.17). En ambos casos, lo primero es desactivar la función de llamada de los botones, para posteriormente realizar el movimiento en sí actualizando la variable tablero y la propiedad "CData" del botón de la casilla elegida. Este es el momento ideal para reproducir el objeto de sonido creado en la subsección 4.1.2 mediante la orden `play`. Viendo que en cada turno se crea un objeto *Timer*, será necesario borrarlo al acabar con una llamada a la función *matatimer* (código 4.18). Por último, se llama a la función *finpartida* que se explica más adelante.

Código 4.16 Turno del usuario.

```
function turno_X(~,~)
    botoncomienzo.Callback=[];
    jugadorActual=1;
    disponible=find( tablero ==0);
    tiempo_X=timer;
```

```
    tiempo_X.Period=1;
    tiempo_X.StartDelay=1;
    tiempo_X.TasksToExecute=T;
    tiempo_X.ExecutionMode='fixedRate';
    tiempo_X.TimerFcn=@xtimer;
    tiempo_X.StopFcn=@xtimer_stop;
    for ii=1:n
        for jj=1:n
            if tablero(ii,jj)==0
                b{ii,jj}.Callback=@movUsuario;
            end
        end
    end

    start(tiempo_X)

    function xtimer(~,~)
        t=t-1;
        textotimer.String=num2str(t);
    end

    function xtimer_stop(~,~)
        for ii=1:n
            for jj=1:n
                b{ii,jj}.Callback=[];
            end
        end
        if pulsa==1
            pulsa=0;
            tablero(casilla)=1;
            b{casilla}.CData=ximagen;
        else
            casilla=movAleatorio;
            tablero(casilla)=1;
            b{casilla}.CData=ximagen;
        end
        matatimer(tiempo_X)
        play(poneficha)
        finpartida
    end

end
```

Código 4.17 Movimiento aleatorio.

```
function casilla = movAleatorio
    %Funcion que escoge al azar una casilla libre
    x=randi(length(disponible));
    casilla=disponible(x);
end

function casilla = movOptimo
    mejorPuntuacion=-Inf;
    for k=1:length(disponibleaux)
        tableroaux(disponibleaux(k))=-1;
        sig_disponible=find(tableroaux==0);
        puntuacion=minimax(tableroaux,0,false,sig_disponible);
        tableroaux(disponibleaux(k))=0;
        if puntuacion>mejorPuntuacion
            mejorPuntuacion=puntuacion;
```

Código 4.18 Borrado del *Timer*.

```
function matatimer(tiempo)
    delete(tiempo)
end
```

Previamente al inicio del temporizador, se debe asignar la propiedad evento–funcional de los botones correspondientes a las casillas que estén libres. La función de llamada, *movUsuario*, se puede ver en el código 4.19. La función utiliza las ordenes `get` y `floor` para detectar la casilla que se ha pulsado y para la ejecución del objeto *Timer*.

Código 4.19 Función mov usuario.

```
function movUsuario(b,~)
    v=get(b,'Position');
    x=v(1); y=v(2);
    ii=floor(n-(x-margeninf-borde/2)/(ladocasilla+raya));
    jj=floor((y-margenlat-borde/2)/(ladocasilla+raya)+1);
    casilla=(n^2+1)-((ii-1)*n+jj);
    pulsa=1;
    stop(tiempo_X)
end
```

El código 4.20 se encarga del turno de la máquina. Es semejante al código 4.16, con la principal diferencia de la obtención de la casilla a marcar. Ahora se tienen las tres posibles estrategias: escoger un movimiento aleatorio con la función *movAleatorio* (código 4.17), usar el algoritmo *Minimax* o la variante de la *Poda Alfa–Beta*.

Código 4.20 Turno de la máquina.

```
function turno_O(~,~)
    botoncomienzo.Callback=[];
    jugadorActual=-1;
    disponible=find(tablero==0);
    tiempo_O=timer;
    tiempo_O.Period=1;
    tiempo_O.StartDelay=1;
    tiempo_O.TasksToExecute=T;
    tiempo_O.ExecutionMode='fixedRate';
    tiempo_O.TimerFcn=@otimer;
    tiempo_O.StopFcn=@otimer_stop;

    start(tiempo_O)

    switch estrategia
        case 1
            pulsa2=1;
            casilla=movAleatorio;
        case 2
            tableroaux=tablero;
            disponibleaux=disponible;
            casilla=movOptimo;
        case 3
            tableroaux=tablero;
            disponibleaux=disponible;
            casilla=movOptimo_poda;
    end

    stop(tiempo_O)

    function otimer(~,~)
```

```
        t=t-1;
%           textotimer . String=num2str(t);
    end

    function otimer_stop (~,~)
        if pulsa2==1
            pulsa2=0;
            tablero ( casilla )=-1;
            b{ casilla }.CData=oimagen;
        else
            casilla =movAleatorio;
            tablero ( casilla )=-1;
            b{ casilla }.CData=oimagen;
        end
        matatimer(tiempo_O)
        play(poneficha)
        finpartida
    end
end
```

Independientemente del turno, cuando se pone una ficha se llama a la función *finpartida* expuesta en el código 4.21. Esta función empieza comprobando si la partida ha acabado con la función auxiliar *compruebaGanador* (código 4.27), cuyo funcionamiento se explica más adelante. Si la partida no ha acabado aún se llama a la función del turno siguiente y se cambia el objeto de texto que, como se pudo apreciar en la figura 4.7, varía de un turno al siguiente. En caso de que la partida haya acabado ya sea en victoria del usuario, de la máquina o empate, se actualizan las puntuaciones del objeto de texto correspondiente y se llama a la función *siguientepartida*.

Código 4.21 Fin de la partida.

```
function finpartida
    [ganador,a]=compruebaGanador(tablero);
    t=T;
    if ganador==0 && a==1 %la partida no ha acabado
        if jugadorActual==1
            textotiemporest . Position =[100,10,350,40];
            textotiemporest . String='TURNO DE LA MÁQUINA';
            textotiemporest .ForegroundColor='red';
            textotimer . Visible =' off ' ;
            pause(1)
            turno_O
        else
            textotiemporest . Position =[130,10,260,40];
            textotiemporest . String='ES TU TURNO';
            textotiemporest .ForegroundColor='blue';
            textotimer .ForegroundColor='blue';
            textotimer . Visible='on';
            textotimer . String=num2str(t);
            turno_X
        end
    elseif ganador==1 %gana el usuario
        puntosusuario=puntosusuario+1;
        textopuntosusuario . String=num2str(puntosusuario);
        siguientepartida
    elseif ganador==-1 %gana la maquina
        puntosmaquina=puntosmaquina+1;
        textopuntosmaquina . String=num2str(puntosmaquina);
        siguientepartida
    elseif ganador==0 && a~=1 %empate
        siguientepartida
    end
end
```

La función *siguientepartida* es la más extensa no por complejidad, sino porque se encarga de reiniciar todo los valores, parámetros, imágenes, propiedades evento–funcionales, etc. necesarios para comenzar una nueva partida. Destacar especialmente en el código 4.22 el uso de cuadros de diálogo de tipo `questiondlg`, que permiten hacer preguntas al usuario y realizar unas tareas u otras según sus respuestas.

Código 4.22 Siguiente partida (I).

```
function siguientepartida
    delete( timerfindall )
    %Preguntamos siguiente partida o juego
    if numeropartida== totalpartidas
        if puntosusuario<puntosmaquina
            play( derrota )
            answer=questdlg([ 'Ha ganado la máquina por ',num2str(puntosmaquina),' a ',num2str(
                puntosusuario) ,...
                ', ¿quieres jugar otra vez?' ], '¡Mala suerte!', 'Si', 'No', 'Si');
        elseif puntosusuario>puntosmaquina
            play( victoria )
            answer=questdlg([ 'Has ganado por ',num2str(puntosusuario), ' a ',num2str(puntosmaquina),...
                ', ¿quieres jugar otra vez?' ], '¡Enhorabuena!', 'Si', 'No', 'Si');
        elseif puntosusuario==puntosmaquina
            play(empate)
            answer=questdlg([ 'Ha habido un empate a ',num2str(puntosmaquina),...
                ', ¿quieres jugar otra vez?' ], '¡Empate!', 'Si', 'No', 'Si');
        end
    else
        answer=questdlg([ 'Llevas ',num2str(numeropartida),' de ',num2str( totalpartidas ) ,...
            ' partidas, ¿quieres seguir jugando?' ], 'Fin de la partida', 'Si', 'No', 'Si');
    end
```

Si la respuesta del usuario es “Sí”, se diferencia el caso de que no queden más partidas (hay que reiniciar el juego entero) o que no sea la última (solo se reinicia el tablero, es decir, la propiedad “CData” de los botones, y el tiempo restante, además de actualizar el numero de la partida actual y el texto de cuenta atrás como en un cambio de turno normal).

Código 4.23 Siguiente partida (II).

```
    switch answer
        case 'Si'
            if numeropartida== totalpartidas
                tablero=zeros(n);
                for ii=1:n
                    for jj=1:n
                        b{ii,jj}.CData=fondo(1:ladocasilla,1:ladocasilla,:);
                    end
                end
                totalpartidas=3;
                numeropartida=1;
                T=20;
                puntosusuario=0;
                puntosmaquina=0;
                textopuntosusuario.String='';
                textopuntosmaquina.String='';
                textotimer.String='';
                entradapartidas.String=num2str( totalpartidas );
                entradapartidas.Style='edit';
                entradatiempo.String=num2str(T);
                entradatiempo.Style='edit';
                textopartidas.Visible='off';
                textode.Visible='off';
                textonum1.Visible='off';
```

```
                textonum2.Visible='off';
                textotiemporest.Visible='off';
                textotimer.Visible='off';
                lista1.Callback=@elige1;
                lista2.Callback=@elige2;
                lista3.Callback=@elige3;
                lista1.Value=1;
                lista2.Value=1;
                estrategia=1;
                n=3;
                primerJugador=comienzoAleatorio;
                botoncomienzo.Callback=@comprobaciones_previas;
            else
                tablero=zeros(n);
                for ii=1:n
                    for jj=1:n
                        b{ii,jj}.CData=fondo(1:ladocasilla,1:ladocasilla,:);
                    end
                end
                numeropartida=numeropartida+1;
                textonum1.String=num2str(numeropartida);
                textotimer.String=num2str(T);
                if jugadorActual==-1
                    textotiemporest.Position=[130,10,260,40];
                    textotiemporest.String='ES TU TURNO';
                    textotiemporest.ForegroundColor='blue';
                    textotimer.Visible='on';
                    textotimer.String=num2str(t);
                    botoncomienzo.Callback=@turno_X;
                else
                    textotiemporest.Position=[100,10,350,40];
                    textotiemporest.String='TURNO DE LA MÁQUINA';
                    textotiemporest.ForegroundColor='red';
                    textotimer.Visible='off';
                    botoncomienzo.Callback=@turno_O;
                end
            end
```

En caso de que el usuario no quiera seguir jugando y no queden partidas por jugar en el juego, la aplicación se cierra. Si quedasen partidas, se vuelve a recurrir a `questiondlg` para mostrar el número de partidas que se han jugado del total y el resultado hasta ese momento y preguntar si se quiere empezar un juego nuevo. Ante una respuesta afirmativa se reinicia el juego exactamente igual que si se hubiesen jugado todas las partidas y, si la respuesta fuese "No", se cierra la aplicación.

Código 4.24 Siguiente partida (III).

```
        case 'No'
            if numeropartida==totalpartidas
                close all
            else
                if puntosusuario<puntosmaquina
                    play(derrota)
                    answer2=questdlg(['Se han jugado ',num2str(numeropartida),' de ',num2str(
                        totalpartidas),...
                        ' partidas y ha ganado la máquina por ',num2str(puntosmaquina),' a ',...
                        num2str(puntosusuario),', ¿quieres jugar otra vez?'],'¡Mala suerte!','Si','No'
                            ,'Si');
                elseif puntosusuario>puntosmaquina
                    play(victoria)
                    answer2=questdlg(['Se han jugado ',num2str(numeropartida),' de ',num2str(
                        totalpartidas),...
```

```
                    ' partidas y has ganado por ',num2str(puntosusuario),' a ',num2str(
                        puntosmaquina),...
                    ', ¿quieres jugar otra vez?'],'¡Enhorabuena!','Si','No','Si');
                elseif puntosusuario==puntosmaquina
                    play(empate)
                    answer2=questdlg(['Se han jugado ',num2str(numeropartida),' de ',num2str(
                        totalpartidas ),...
                    ' partidas y ha habido un empate a ',num2str(puntosmaquina),...
                    ', ¿quieres jugar otra vez?'],'¡Empate!','Si','No','Si');
                end
                switch answer2
                    case 'Si'
                        tablero=zeros(n);
                        for ii=1:n
                            for jj=1:n
                                b{ii,jj}.CData=fondo(1:ladocasilla,1:ladocasilla,:);
                            end
                        end
                        totalpartidas=3;
                        numeropartida=1;
                        T=20;
                        puntosusuario=0;
                        puntosmaquina=0;
                        textopuntosusuario.String='';
                        textopuntosmaquina.String='';
                        textotimer.String='';
                        entradapartidas.String=num2str(totalpartidas);
                        entradapartidas.Style='edit';
                        entradatiempo.String=num2str(T);
                        entradatiempo.Style='edit';
                        textopartidas.Visible='off';
                        textode.Visible='off';
                        textonum1.Visible='off';
                        textonum2.Visible='off';
                        textotiemporest.Visible='off';
                        textotimer.Visible='off';
                        lista1.Callback=@elige1;
                        lista2.Callback=@elige2;
                        lista3.Callback=@elige3;
                        lista1.Value=1;
                        lista2.Value=1;
                        estrategia=1;
                        primerJugador=comienzoAleatorio;
                        n=3;
                        botoncomienzo.Callback=@comprobaciones_previas;
                    case 'No'
                        close all
                end
            end
        end
    end
end
```

Se presentan a continuación en el código 4.25 las funciones asociadas a las propiedades evento–funcionales de los menús desplegables descritos en la sección 4.2. Al seleccionar una opción de un menú, se está cambiando la propiedad "Value" del objeto, que es usada para actualizar las variables pertinentes. Comentar que la función *elige3*, encargada de dibujar el tablero, es un poco más compleja porque debe borrar primero el panel y la matriz de celdas existente antes de dibujar los nuevos.

Código 4.25 Funciones de llamada de los menús desplegables.

```
function elige1 (~,~)
    if lista1 .Value==1
        estrategia =1; %aleatorio
    elseif lista1 .Value==2
        estrategia =2; %minimax
    elseif lista1 .Value==3
        estrategia =3; %poda
    end
end
function elige2 (~,~)
    if lista2 .Value==1
        primerJugador=comienzoAleatorio;
    elseif lista2 .Value==2
        primerJugador=1;
    elseif lista2 .Value==3
        primerJugador=-1;
    end
end
function elige3 (~,~)
    delete(p)
    for ii=1:n^2
        delete(b{ii})
    end
    b=[];
    p=[];
    if lista3 .Value==1
        n=3;
    elseif lista3 .Value==2
        n=4;
    elseif lista3 .Value==3
        n=5;
    end
    creatablero
end
```

En la función *elige2* existe una llamada a una función auxiliar (código 4.26) que usa la orden `rand` de Matlab para escoger de forma "aleatoria" uno de los dos jugadores para que sea el primero en jugar.

Código 4.26 Jugador inicial.

```
function jugadorActual = comienzoAleatorio
    if rand(1)<0.5
        jugadorActual=jugadores(1);
    else
        jugadorActual=jugadores(2);
    end
end
```

Para acabar, se muestra en el código 4.27 la función auxiliar que se usa en varios momentos a lo largo del programa *compruebaGanador*. A pesar de su longitud, su funcionamiento es sencillo: debe comprobar las líneas horizontales, verticales y ambas diagonales en busca de tres, cuatro o cinco fichas consecutivas del mismo jugador, dependiendo de las dimensiones del tablero que se hayan seleccionado.

Código 4.27 Comprobación del ganador.

```
function [ganador,a] = compruebaGanador(tablero)
    a=1;
    ganador=0;
    switch n
```

```
case 3
    % Comprobacion lineas horizontales
    for i=1:3
        if tablero(i,1)==tablero(i,2) && tablero(i,2)==tablero(i,3) && tablero(i,1)~=0
            a=2;
            ganador=tablero(i,1);
            break
        end
    end
    % Comprobacion lineas verticales
    for j=1:3
        if tablero(1,j)==tablero(2,j) && tablero(2,j)==tablero(3,j) && tablero(1,j)~=0
            a=2;
            ganador=tablero(1,j);
            break
        end
    end
    % Comprobacion diagonal principal
    if tablero(1,1)==tablero(2,2) && tablero(2,2)==tablero(3,3) && tablero(1,1)~=0
        a=2;
        ganador=tablero(2,2);
    end
    % Comprobacion diagonal secundaria
    if tablero(3,1)==tablero(2,2) && tablero(2,2)==tablero(1,3) && tablero(3,1)~=0
        a=2;
        ganador=tablero(3,1);
    end
    % Empate
    x=find(tablero~=0);
    if length(x)==n^2 && ganador==0 %si no hay casillas libres y no hay ganador tenemos un
        empate
        a=2;
    end
case 4
    for i=1:4
        if tablero(i,1)==tablero(i,2) && tablero(i,2)==tablero(i,3) && tablero(i,3)==tablero(i
            ,4) && tablero(i,1)~=0
            a=2;
            ganador=tablero(i,1);
            break
        end
    end
    for j=1:4
        if tablero(1,j)==tablero(2,j) && tablero(2,j)==tablero(3,j) && tablero(3,j)==tablero
            (4,j) && tablero(1,j)~=0
            a=2;
            ganador=tablero(1,j);
            break
        end
    end
    if tablero(1,1)==tablero(2,2) && tablero(2,2)==tablero(3,3) && tablero(3,3)==tablero(4,4)
        && tablero(1,1)~=0
        a=2;
        ganador=tablero(2,2);
    end
    if tablero(4,1)==tablero(3,2) && tablero(3,2)==tablero(2,3) && tablero(2,3)==tablero(1,4)
        && tablero(4,1)~=0
        a=2;
        ganador=tablero(4,1);
    end
    x=find(tablero~=0);
```

```
            if length(x)==n^2 && ganador==0 %si no hay casillas libres y no hay ganador tenemos un
                empate
                a=2;
            end
        case 5
            for i=1:5
                if tablero(i,1)==tablero(i,2) && tablero(i,2)==tablero(i,3) && tablero(i,3)==tablero(i
                    ,4) && tablero(i,4)==tablero(i,5) && tablero(i,1)~=0
                    a=2;
                    ganador=tablero(i,1);
                    break
                end
            end
            for j=1:5
                if tablero(1,j)==tablero(2,j) && tablero(2,j)==tablero(3,j) && tablero(3,j)==tablero
                    (4,j) && tablero(4,j)==tablero(5,j) && tablero(1,j)~=0
                    a=2;
                    ganador=tablero(1,j);
                    break
                end
            end
            if tablero(1,1)==tablero(2,2) && tablero(2,2)==tablero(3,3) && tablero(3,3)==tablero(4,4)
                && tablero(4,4)==tablero(5,5) && tablero(1,1)~=0
                a=2;
                ganador=tablero(2,2);
            end
            if tablero(5,1)==tablero(4,2) && tablero(4,2)==tablero(3,3) && tablero(3,3)==tablero(2,4)
                && tablero(2,4)==tablero(1,5) && tablero(5,1)~=0
                a=2;
                ganador=tablero(5,1);
            end
            x=find(tablero~=0);
            if length(x)==n^2 && ganador==0 %si no hay casillas libres y no hay ganador tenemos un
                empate
                a=2;
            end
    end
end
```

5 Conclusiones

En este último capítulo se resumen los resultados obtenidos, se muestra la aplicación en funcionamiento y se comenta la línea a seguir en los siguientes pasos de desarrollo de la aplicación.

Se ha conseguido crear una aplicación desde cero totalmente personalizada y única, con varias opciones de juego a elegir por el usuario y con una estética atractiva. El programa es bastante robusto, previniendo posibles errores provocados por el usuario en la introducción de valores o por el cierre directo de la ventana de la aplicación. En las siguientes figuras se pueden ver varias situaciones posibles en la aplicación:

Figura 5.1 Ventana de la aplicacion (I).

En la figura 5.1 se ve un mensaje de error debido a un tiempo de respuesta no válido. Efectivamente, en el objeto editable donde se escribe el tiempo hay una letra. Lo mismo pasaría con una entrada no válida del número de partidas. El mensaje permanece ahí hasta que se acepta y una vez solucionados los errores habrá que pulsar de nuevo el botón de comienzo.

Figura 5.2 Ventana de la aplicacion (II).

En la figura 5.2 se muestra un momento concreto de una partida en un tablero 3×3, usando como estrategia el algoritmo *Minimax* y restando 16 segundos al usuario para realizar su movimiento.

Figura 5.3 Ventana de la aplicacion (III).

Cuando se acaba el juego se muestra el resultado y se pregunta al usuario si quiere seguir jugando (figura 5.3), además de reproducirse el sonido correspondiente (victoria, empate o derrota)

Figura 5.4 Ventana de la aplicacion (IV).

En el caso de que acabe una partida y antes de empezar la siguiente, siempre se pregunta al usuario si quiere seguir jugando, mediante un mensaje en un cuadro de diálogo como el que se ve en la figura 5.4.

Figura 5.5 Ventana de la aplicacion (V).

Con una vista del transcurso de una partida en el tablero 5×5 (figura 5.5) se terminan las imágenes de la aplicación, aunque se podrían mostrar muchos más ejemplos de las características y particularidades de la misma.

Por otro lado, la implementación del algoritmo *Minimax* y de la *Poda Alfa–Beta* en el tres en raya se ha podido hacer completa, es decir, analizando todo el árbol de juego. Para el cuatro y cinco en raya esto ha sido imposible, ya que conducía irremediablemente a tiempos de ejecución altísimos. Para solventar esto, se limitó el numero de movimientos hacia adelante que se analizaban, la profundidad. Es aquí donde se ve la

efectividad de la *Poda Alfa–Beta*, permitiendo aumentar la profundidad máxima a analizar de 4 a 8 en el caso del 4×4 y de 3 a 6 en el 5×5. Esta técnica duplica la profundidad con la que se analiza el árbol en aproximadamente el mismo tiempo. La otra opción era mantener la misma profundidad para que disminuyera el tiempo de ejecución.

El hecho de no evaluar todos los estados del árbol de juego planteó un nuevo problema, la asignación de una puntuación a un estado de una partida sin acabar. La solución fue asignar una puntuación de 0 a estos estados lo que provoca que el programa, aunque es imbatible, realice movimientos extraños que parecen no ser los óptimos.

La aplicación es, sin dudas, mejorable. En una etapa posterior en el desarrollo de esta aplicación se deberían de tratar los siguientes aspectos:

- Creación de un cuadro de ayuda, que contenga información sobre las opciones y funcionalidades de la aplicación. Se mostraría el mensaje al pulsar un botón antes de comenzar el juego.
- Mejora del código *Minimax*, intentando usar las propiedades de simetría del tablero (al menos al inicio de la partida) para ahorrar tiempo de ejecución y poder analizar más movimientos hacia adelante. Por ejemplo, en el tres en raya el primer jugador tiene en realidad solo 3 posibilidades (esquina, borde o centro) y no las 9 que actualmente analiza el código. Esto podría reducir el tiempo de ejecución, aunque siempre a expensas de las limitaciones propias de Matlab.
- Implementación de una función de evaluación para estados intermedios, que devuelva una puntuación acorde con la situación del tablero. Por ejemplo, el el cuatro en raya un movimiento que consiga alinear tres fichas del mismo jugador seguidas debe tener buena puntuación. Esta función podría llegar a ser tremendamente compleja y extensa.

Apéndice A
Códigos de Matlab

Se recogen en este apéndice el código completo de la aplicación y el programa auxiliar que genera las imágenes de las fichas mediante el tratamiento digital (matricial).

Código A.1 Aplicación tres en raya.

```
function tresenraya

close all, clear all, clc

n=3;                                %dimensiones tablero :3,4,5
ladocasilla =[];
totalpartidas =3;                   %numero total de partidas en un juego
numeropartida=1;                    %numero de la partida actual
tablero =[];                        %matriz del tablero :-1(maquina,O),0( libre ) ,1( usuario ,X)
tableroaux = tablero ;
casilla =[];                        %casilla del tablero en la que va a poner una ficha :1,2,...,9
disponible =[];                     %vector de casillas libres
disponibleaux =[];
ganador=[];                         %ganador de una partida :-1,1
a=[];                               %variable para controlar los empates:1(quedan huecos) ,2( no quedan
      huecos)
puntosusuario=0;                    %partidas ganadas por el usuario
puntosmaquina=0;                    %partidas ganadas por la maquina
jugadores =[-1,1];                  %vector con los dos jugadores
jugadorActual =[];                  %jugador actual :-1,1
primerJugador=comienzoAleatorio;    %jugador que mueve primero en una partida :-1,1
estrategia =1;                      %vector con las 3 estrategias posibles : aleatorio ,minimax,poda
T=20;                               %tiempo de respuesta
t=T;                                %tiempo restante en cada movimiento:T,T -1,...,0
answer=[];                          %respuesta al questdlg para seguir jugando o no:' Si ',' No'

ximagen=[];
oimagen=[];
b=[];
p=[];

tiempo_X=[];
pulsa=0;
tiempo_O=[];
pulsa2=0;

%Figura Principal
margenlat=30;
margeninf=65;
borde=14;
```

```
raya=6-(n-3)*2;

fondo=imread('madera.jpg');

hfig=figure('Name','Tres en raya',...
    'Position',[1 1 1200 640],...
    'Color',[0.65 0.65 0.65],...
    'menubar','none',...
    'numbertitle','off',...
    'Visible','off',...
    'Resize','off',...
    'DeleteFcn',@cierra);
ejes=axes('position',[0 0 1 1]);
X=image(fondo);
axis off
movegui(hfig,'center');

fondoopciones=[215 164 108]/255;
fondotexto=[235 190 120]/255;
h=1;
%Texto contador partidas
textopartidas=uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'String','PARTIDA',...
    'Position',[150,600,150,35],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'Visible','off');
textonum1=uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'Position',[285,600,40,35],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'FontWeight','bold',...
    'Visible','off');
textode=uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'String','DE',...
    'Position',[325,600,50,35],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'Visible','off');
textonum2=uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
    'Position',[375,600,40,35],...
    'HorizontalAlignment','center',...
    'FontSize',20,...
    'FontWeight','bold',...
    'Visible','off');

%Panel derecha
margensup=20;
margendcha=20;
anchopanel=580;
altopanel=440;
xpanel=1200-margendcha-anchopanel;
ypanel=640-margensup-altopanel;

%Texto estrategia
altopanel=altopanel-45;
textoestrategia=uicontrol('Style','text',...
    'BackgroundColor',fondotexto,...
```

```
        'String','ESTRATEGIA DE RESPUESTA',...
        'Position',[xpanel+40,ypanel+altopanel ,500,35],...
        'HorizontalAlignment','center' ,...
        'FontSize' ,20,...
        'FontWeight','bold');
 lista1 =uicontrol('style','popupmenu',...
        'position',[xpanel+40,ypanel+altopanel -50,500,35],...
        'FontSize' ,16,...
        'string',{'Aleatorio','Minimax','Poda' },...
        'BackgroundColor',fondoopciones ,...
        'callback',@elige1);

 %Texto jugador inicial
textocomienzo=uicontrol('Style','text' ,...
        'BackgroundColor',fondotexto ,...
        'String','ELECCIÓN JUGADOR INICIAL',...
        'Position',[xpanel+40,ypanel+3*altopanel /4,500,35],...
        'HorizontalAlignment','center' ,...
        'FontSize' ,20,...
        'FontWeight','bold');
 lista2 =uicontrol('style','popupmenu',...
        'position',[xpanel+40,ypanel+3*altopanel /4-50,500,35],...
        'FontSize' ,16,...
        'string',{'Aleatorio','Usuario','Máquina' },...
        'BackgroundColor',fondoopciones ,...
        'callback',@elige2);

 %Texto eleccion tiempo de respuesta
textocomienzo=uicontrol('Style','text' ,...
        'BackgroundColor',fondotexto ,...
        'String','TIEMPO DE RESPUESTA',...
        'Position',[xpanel+40,ypanel+altopanel /2-20,420,35],...
        'HorizontalAlignment','center' ,...
        'FontSize' ,20,...
        'FontWeight','bold');
entradatiempo=uicontrol('Style','edit' ,...
        'Position',[xpanel+480,ypanel+altopanel /2-20,60,35],...
        'BackgroundColor',fondoopciones ,...
        'FontSize' ,20,...
        'FontWeight','bold' ,...
        'String',num2str(T));

 %Texto eleccion numero partidas
textocomienzo=uicontrol('Style','text' ,...
        'BackgroundColor',fondotexto ,...
        'String','Nº DE PARTIDAS POR JUEGO',...
        'Position',[xpanel+40,ypanel+altopanel /2-80,420,35],...
        'HorizontalAlignment','center' ,...
        'FontSize' ,20,...
        'FontWeight','bold');
 entradapartidas =uicontrol('Style','edit' ,...
        'Position',[xpanel+480,ypanel+altopanel /2-80,60,35],...
        'BackgroundColor',fondoopciones ,...
        'FontSize' ,20,...
        'FontWeight','bold' ,...
        'String',num2str( totalpartidas ));

 %Texto eleccion tamaño tablero
 textotablero =uicontrol('Style','text' ,...
        'BackgroundColor',fondotexto ,...
        'String','TAMAÑO DEL TABLERO',...
        'Position',[xpanel+40,ypanel+altopanel /2-140,420,35],...
```

```
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold');
 lista3 =uicontrol('style','popupmenu',...
        'position',[xpanel+480,ypanel+altopanel/2-140,60,35],...
        'FontSize',16,...
        'string',{'3x3','4x4','5x5'},...
        'BackgroundColor',fondoopciones,...
        'callback',@elige3);

 %Texto empezar partida
 textoempezarpartida =uicontrol('Style','text',...
        'BackgroundColor',fondotexto,...
        'String','COMIENZO PARTIDA',...
        'Position',[xpanel+40,ypanel-50,400,35],...
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold');
botoncomienzo=uicontrol('Style','pushbutton',...
        'Callback',@comprobaciones_previas,...
        'position',[xpanel+460,ypanel-63,61,61],...
        'BackgroundColor',fondoopciones,...
        'String','START',...
        'FontWeight','bold',...
        'FontSize',12);

 %Texto puntuacion
 textopuntuacion =uicontrol('Style','text',...
        'BackgroundColor',fondotexto,...
        'String','PUNTUACIÓN',...
        'Position',[xpanel+anchopanel/2-100,ypanel-100,200,35],...
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold');
 textousuario =uicontrol('Style','text',...
        'BackgroundColor',fondotexto,...
        'String','USUARIO',...
        'Position',[xpanel+40,ypanel-150,150,35],...
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold');
 textopuntosusuario =uicontrol('Style','text',...
        'Position',[xpanel+210,ypanel-150,40,35],...
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold',...
        'BackgroundColor',fondoopciones,...
        'ForegroundColor','blue');
textomaquina=uicontrol('Style','text',...
        'BackgroundColor',fondotexto,...
        'String','MÁQUINA',...
        'Position',[xpanel+330,ypanel-150,150,35],...
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold');
textopuntosmaquina=uicontrol('Style','text',...
        'Position',[xpanel+anchopanel-80,ypanel-150,40,35],...
        'HorizontalAlignment','center',...
        'FontSize',20,...
        'FontWeight','bold',...
        'BackgroundColor',fondoopciones,...
        'ForegroundColor','red');
```

```
%Texto cuenta atras
 textotiemporest =uicontrol(' Style ',' text ' ,...
      'BackgroundColor',fondotexto ,...
      ' String ','TIEMPO RESTANTE',...
      ' Position ' ,[130,10,260,40],...
      'HorizontalAlignment',' center ' ,...
      'FontSize' ,20,...
      ' Visible ',' off ');
 textotimer =uicontrol(' Style ',' text ' ,...
      'BackgroundColor',fondotexto ,...
      ' Position ' ,[390,10,50,40],...
      ' String ',num2str(t) ,...
      'FontSize' ,20,...
      'FontWeight','bold' ,...
      'HorizontalAlignment',' center ' ,...
      ' Visible ',' off ');

 creatablero

%Objetos de Sonido
[y1,Fs1]=audioread(' poneficha .mp3',[1,22050/3]);
poneficha=audioplayer(y1,Fs1);

[y2,Fs2]=audioread(' victoria .mp3');
 victoria =audioplayer(y2,Fs2);

[y3,Fs3]=audioread(' derrota .wav');
derrota=audioplayer(y3,Fs3);

[y4,Fs4]=audioread('empate.wav');
empate=audioplayer(y4,Fs4);

figure(hfig)
hfig . Visible='on';

%% FUNCIONES AUXILIARES %%

function comprobaciones_previas(~,~)
    e=[];
    v3=isempty(entradatiempo. String );
    v4=isempty( entradapartidas . String );
    v5=str2double(entradatiempo . String );
    v6=str2double( entradapartidas . String );
    if v3==1
        e=errordlg('No has puesto tiempo de respuesta ','Error');
        waitfor (e)
    end
    if v4==1
        e=errordlg('No has puesto numero de partidas ','Error');
        waitfor (e)
    end
    if isnan(v5) && v3==0
        e=errordlg('Tiempo de respuesta no valido ','Error');
        waitfor (e)
    end
    if isnan(v6) && v4==0
        e=errordlg('Numero de partidas no valido ','Error');
        waitfor (e)
    end
    if isempty(e)
        comienzo
```

```
    end
end

function creatablero

    ladocasilla =480/n;
    %Panel (tablero)
    p=uipanel('Units','pixels',...
        'Position',[margenlat-borde/2,...
        margeninf-borde/2,...
        ladocasilla*n+2*borde+(n-1)*raya,...
        ladocasilla*n+2*borde+(n-1)*raya],...
        'BorderType','none',...
        'BackgroundColor','black');
    %Botones (casillas tablero)
    for ii=1:n
        for jj=1:n
            b{ii,jj}=uicontrol('Style','pushbutton',...
                'Callback',[],...
                'position',[margenlat+borde/2+(jj-1)*ladocasilla+(jj-1)*raya,...
                margeninf+borde/2+(n-ii)*ladocasilla+(n-ii)*raya,...
                ladocasilla,ladocasilla],...
                'BackgroundColor','white',...
                'CData',fondo(1:ladocasilla,1:ladocasilla,:));
        end
    end
end

function comienzo
    botoncomienzo.Callback=[];

    tablero=zeros(n);

    str1=get(entradatiempo,'string');
    T=str2double(str1);
    t=T;
    entradatiempo.Style='text';
    str2=get(entradapartidas,'string');
    entradapartidas.Style='text';
    totalpartidas=str2double(str2);
    numeropartida=1;
    textonum1.String=num2str(numeropartida);
    textonum2.String=str2;

    lista1.Callback=[];
    lista2.Callback=[];
    lista3.Callback=[];

    textopartidas.Visible='on';
    textode.Visible='on';
    textonum1.Visible='on';
    textonum2.Visible='on';

    textopuntosusuario.String=num2str(puntosusuario);
    textopuntosmaquina.String=num2str(puntosmaquina);

    xfoto=imread('xficha2.jpg');
    ximagen=imresize(xfoto,[ladocasilla ladocasilla]);
    ofoto=imread('oficha2.jpg');
    oimagen=imresize(ofoto,[ladocasilla ladocasilla]);

    jugadorActual=primerJugador;
```

```
    if jugadorActual==1
        textotiemporest . Visible='on'; 
        textotiemporest . String='ES TU TURNO';
        textotiemporest .ForegroundColor='blue';
        textotimer . Visible='on'; textotimer .
        String=num2str(t);
        textotimer .ForegroundColor='blue';
        turno_X
    else
        textotiemporest . Visible='on';
        textotiemporest . Position =[100,10,350,40];
        textotiemporest . String='TURNO DE LA MÁQUINA';
        textotiemporest .ForegroundColor='red';
        textotimer . Visible =' off ' ;
        pause(1)
        turno_O
    end
end

function turno_X(~,~)
    botoncomienzo.Callback=[];
    jugadorActual=1;
    disponible =find( tablero ==0);
    tiempo_X=timer;
    tiempo_X.Period=1;
    tiempo_X.StartDelay=1;
    tiempo_X.TasksToExecute=T;
    tiempo_X.ExecutionMode='fixedRate';
    tiempo_X.TimerFcn=@xtimer;
    tiempo_X.StopFcn=@xtimer_stop; for
    ii =1:n
        for jj =1:n
            if tablero ( ii , jj )==0
                b{ ii , jj }.Callback=@movUsuario;
            end
        end
    end

    start (tiempo_X)

    function xtimer (~,~)
        t=t-1;
        textotimer . String =num2str(t);
    end

    function xtimer_stop (~,~)
        for ii =1:n
            for jj =1:n
                b{ ii , jj }.Callback =[];
            end
        end
        if pulsa==1
            pulsa=0;
            tablero ( casilla )=1;
            b{ casilla }.CData=ximagen;
        else
            casilla =movAleatorio;
            tablero ( casilla )=1;
            b{ casilla }.CData=ximagen;
        end
        matatimer(tiempo_X)
        play( poneficha )
```

```
            finpartida
        end

    end

    function turno_O(~,~)
        botoncomienzo.Callback =[];
        jugadorActual=-1;
        disponible =find( tablero ==0);
        tiempo_O=timer;
        tiempo_O.Period=1;
        tiempo_O.StartDelay=1;
        tiempo_O.TasksToExecute=T;
        tiempo_O.ExecutionMode='fixedRate';
        tiempo_O.TimerFcn=@otimer;
        tiempo_O.StopFcn=@otimer_stop;

         start (tiempo_O)

        switch   estrategia
            case  1
                pulsa2=1;
                casilla =movAleatorio;
            case  2
                tableroaux = tablero ;
                disponibleaux =disponible ;
                casilla =movOptimo;
            case  3
                tableroaux = tablero ;
                disponibleaux =disponible ;
                casilla =movOptimo_poda;
        end

        stop(tiempo_O)

        function  otimer(~,~)
            t=t-1;
%             textotimer . String =num2str(t);
        end

        function  otimer_stop (~,~)
            if  pulsa2==1
                pulsa2=0;
                tablero ( casilla )=-1;
                b{ casilla }.CData=oimagen;
            else
                casilla =movAleatorio;
                tablero ( casilla )=-1;
                b{ casilla }.CData=oimagen;
            end
            matatimer(tiempo_O)
            play(poneficha)
             finpartida
        end
    end

    function   finpartida
        [ganador,a]=compruebaGanador(tablero);
        t=T;
        if  ganador==0 && a==1  %la partida no ha acabado
            if  jugadorActual==1
                 textotiemporest . Position =[100,10,350,40];
```

```
            textotiemporest . String='TURNO DE LA MÁQUINA';
            textotiemporest .ForegroundColor='red';
            textotimer . Visible =' off ' ;
            pause(1)
            turno_O
        else
            textotiemporest . Position =[130,10,260,40];
            textotiemporest . String='ES TU TURNO';
            textotiemporest .ForegroundColor='blue';
            textotimer .ForegroundColor='blue';
            textotimer . Visible='on'; textotimer .
            String=num2str(t); turno_X

        end
    elseif ganador==1 %gana el usuario
        puntosusuario=puntosusuario+1;
        textopuntosusuario . String=num2str(puntosusuario);
        siguientepartida
    elseif ganador==-1 %gana la maquina
        puntosmaquina=puntosmaquina+1;
        textopuntosmaquina. String=num2str(puntosmaquina);
        siguientepartida
    elseif ganador==0 && a~=1 %empate
        siguientepartida
    end
end

function siguientepartida
    delete( timerfindall )
    %Preguntamos siguiente partida o juego
    if numeropartida== totalpartidas
        if puntosusuario<puntosmaquina
            play( derrota )
            answer=questdlg([' Ha ganado la máquina por ',num2str(puntosmaquina),' a ',num2str(
                puntosusuario) ,...
                ', ¿quieres jugar otra vez?'], '¡Mala suerte!','Si','No','Si');
        elseif puntosusuario>puntosmaquina
            play( victoria )
            answer=questdlg([' Has ganado por ',num2str(puntosusuario),' a ',num2str(puntosmaquina),...
                ', ¿quieres jugar otra vez?'], '¡Enhorabuena!','Si','No','Si');
        elseif puntosusuario==puntosmaquina
            play(empate)
            answer=questdlg([' Ha habido un empate a ',num2str(puntosmaquina),...
                ', ¿quieres jugar otra vez?'], '¡Empate!','Si','No','Si');
        end
    else
        answer=questdlg([' Llevas ',num2str(numeropartida),' de ',num2str( totalpartidas ) ,...
            ' partidas , ¿quieres seguir jugando?'], 'Fin de la partida','Si','No','Si');
    end
    switch answer
        case 'Si'
            if numeropartida== totalpartidas
                tablero=zeros(n);
                for ii=1:n
                    for jj=1:n
                        b{ii,jj}.CData=fondo(1: ladocasilla ,1: ladocasilla ,:) ;
                    end
                end
                totalpartidas =3;
                numeropartida=1;
                T=20;
                puntosusuario=0;
```

```
                puntosmaquina=0;
                textopuntosusuario . String ='' ;
                textopuntosmaquina . String ='' ;
                textotimer . String ='' ;
                entradapartidas . String =num2str( totalpartidas ) ;
                entradapartidas . Style =' edit ' ;
                entradatiempo . String =num2str(T);
                entradatiempo . Style =' edit ' ;
                textopartidas . Visible =' off ' ;
                textode . Visible =' off ' ;
                textonum1. Visible =' off ' ;
                textonum2. Visible =' off ' ;
                textotiemporest . Visible =' off ' ;
                textotimer . Visible =' off ' ;
                lista1 .Callback=@elige1;
                lista2 .Callback=@elige2;
                lista3 .Callback=@elige3;
                lista1 .Value=1;
                lista2 .Value=1;
                estrategia =1;
                n=3;
                primerJugador=comienzoAleatorio;
                botoncomienzo.Callback=@comprobaciones_previas;
            else
                tablero =zeros(n);
                for ii =1:n
                    for jj =1:n
                        b{ ii , jj }.CData=fondo(1: ladocasilla ,1: ladocasilla ,:) ;
                    end
                end
                numeropartida=numeropartida+1;
                textonum1. String =num2str(numeropartida);
                textotimer . String =num2str(T);
                if jugadorActual==-1
                    textotiemporest . Position =[130,10,260,40];
                    textotiemporest . String ='ES TU TURNO';
                    textotiemporest .ForegroundColor='blue';
                    textotimer . Visible ='on';
                    textotimer . String =num2str(t);
                    botoncomienzo.Callback=@turno_X;
                else
                    textotiemporest . Position =[100,10,350,40];
                    textotiemporest . String ='TURNO DE LA MÁQUINA';
                    textotiemporest .ForegroundColor='red';
                    textotimer . Visible =' off ' ;
                    botoncomienzo.Callback=@turno_O;
                end
            end
        case 'No'
            if numeropartida== totalpartidas
                close all
            else
                if puntosusuario<puntosmaquina
                    play( derrota )
                    answer2=questdlg([ 'Se han jugado ',num2str(numeropartida),' de ',num2str(
                        totalpartidas ) ,...
                        ' partidas y ha ganado la máquina por ',num2str(puntosmaquina),' a ' ,...
                        num2str(puntosusuario), ', ¿quieres jugar otra vez?' ], '¡Mala suerte !', 'Si', 'No'
                        , 'Si' );
                elseif puntosusuario>puntosmaquina
                    play( victoria )
```

```
                answer2=questdlg([ 'Se han jugado ' ,num2str(numeropartida),' de ' ,num2str(
                    totalpartidas ) ,...
                    ' partidas y has ganado por ' ,num2str(puntosusuario) ,' a ' ,num2str(
                        puntosmaquina) ,...
                    ' , ¿quieres jugar otra vez?' ], '¡Enhorabuena!' , 'Si' , 'No' , 'Si' );
            elseif puntosusuario==puntosmaquina
                play(empate)
                answer2=questdlg([ 'Se han jugado ' ,num2str(numeropartida),' de ' ,num2str(
                    totalpartidas ) ,...
                    ' partidas y ha habido un empate a ' ,num2str(puntosmaquina) ,...
                    ' , ¿quieres jugar otra vez?' ], '¡Empate!' , 'Si' , 'No' , 'Si' );
            end
            switch answer2
                case 'Si'
                    tablero =zeros(n);
                    for ii =1:n
                        for jj =1:n
                            b{ ii , jj }.CData=fondo(1: ladocasilla ,1: ladocasilla ,:) ;
                        end
                    end
                    totalpartidas =3;
                    numeropartida=1;
                    T=20;
                    puntosusuario =0;
                    puntosmaquina=0;
                    textopuntosusuario . String ='' ;
                    textopuntosmaquina . String ='' ;
                    textotimer . String ='' ;
                    entradapartidas . String =num2str( totalpartidas ) ;
                    entradapartidas . Style =' edit ' ;
                    entradatiempo . String =num2str(T);
                    entradatiempo . Style =' edit ' ;
                    textopartidas . Visible =' off ' ;
                    textode . Visible =' off ' ;
                    textonum1. Visible =' off ' ;
                    textonum2. Visible =' off ' ;
                    textotiemporest . Visible =' off ' ;
                    textotimer . Visible =' off ' ;
                    lista1 . Callback=@elige1;
                    lista2 . Callback=@elige2;
                    lista3 . Callback=@elige3;
                    lista1 . Value=1;
                    lista2 . Value=1;
                    estrategia =1;
                    primerJugador=comienzoAleatorio;
                    n=3;
                    botoncomienzo.Callback=@comprobaciones_previas;
                case 'No'
                    close all
            end
        end
    end
end

function elige1 (~,~)
    if lista1 . Value==1
        estrategia =1; %aleatorio
    elseif lista1 . Value==2
        estrategia =2; %minimax
    elseif lista1 . Value==3
        estrategia =3; %poda
    end
```

```
end
function elige2 (~,~)
    if lista2 .Value==1
        primerJugador=comienzoAleatorio;
    elseif lista2 .Value==2
        primerJugador=1;
    elseif lista2 .Value==3
        primerJugador=-1;
    end
end
function elige3 (~,~)
    delete(p)
    for ii=1:n^2
        delete(b{ii})
    end
    b=[];
    p=[];
    if lista3 .Value==1
        n=3;
    elseif lista3 .Value==2
        n=4;
    elseif lista3 .Value==3
        n=5;
    end
    creatablero
end

function casilla = movAleatorio
    %Funcion que escoge al azar una casilla libre
    x=randi(length(disponible));
    casilla=disponible(x);
end

function casilla = movOptimo
    mejorPuntuacion=-Inf;
    for k=1:length(disponibleaux)
        tableroaux(disponibleaux(k))=-1;
        sig_disponible=find(tableroaux==0);
        puntuacion=minimax(tableroaux,0,false, sig_disponible);
        tableroaux(disponibleaux(k))=0;
        if puntuacion>mejorPuntuacion
            mejorPuntuacion=puntuacion;
            mejorMov=disponibleaux(k);
        end
    end
    casilla=mejorMov;
    pulsa2=1;
end

function mejorPuntuacion = minimax(tableroaux,prof,isMax,disponibleaux)
    [ganador,a]=compruebaGanador(tableroaux);
    if ganador==1
        mejorPuntuacion=-100+prof;
    elseif ganador==-1
        mejorPuntuacion=100-prof;
    elseif a==2 && ganador==0
        mejorPuntuacion=0;
    elseif n==4 && prof>4 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    elseif n==5 && prof>3 && a==1 && ganador==0
        mejorPuntuacion=0;
```

```
            a=2;
        end
        if isMax==true && a==1 %si le toca mover a la maquina
            mejorPuntuacion=-Inf;
            for k=1:length(disponibleaux)
                tableroaux(disponibleaux(k))=-1;
                sig_disponible =find(tableroaux==0);
                puntuacion=minimax(tableroaux,prof+1,false, sig_disponible);
                tableroaux(disponibleaux(k))=0;
                if puntuacion>mejorPuntuacion
                    mejorPuntuacion=puntuacion;
                    mejorMov=disponibleaux(k);
                end
            end
        elseif isMax==false && a==1 %si le toca mover al usuario
            mejorPuntuacion=Inf;
            for k=1:length(disponibleaux)
                tableroaux(disponibleaux(k))=1;
                sig_disponible =find(tableroaux==0);
                puntuacion=minimax(tableroaux,prof+1,true, sig_disponible);
                tableroaux(disponibleaux(k))=0;
                if puntuacion<mejorPuntuacion
                    mejorPuntuacion=puntuacion;
                    mejorMov=disponibleaux(k);
                end
            end
        end
    end

function casilla = movOptimo_poda
    mejorPuntuacion=-Inf;
    for k=1:length(disponibleaux)
        tableroaux(disponibleaux(k))=-1;
        sig_disponible =find(tableroaux==0);
        puntuacion=minimax_poda(tableroaux,0,-Inf,Inf,false, sig_disponible);
        tableroaux(disponibleaux(k))=0;
        if puntuacion>mejorPuntuacion
            mejorPuntuacion=puntuacion;
            mejorMov=disponibleaux(k);
        end
    end
    casilla =mejorMov;
    pulsa2=1;
end

function mejorPuntuacion = minimax_poda(tableroaux,prof,alpha,beta,isMax,disponibleaux)
    [ganador,a]=compruebaGanador(tableroaux);
    if ganador==1
        mejorPuntuacion=-100+prof;
    elseif ganador==-1
        mejorPuntuacion=100-prof;
    elseif a==2 && ganador==0
        mejorPuntuacion=0;
    elseif n==4 && prof>8 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    elseif n==5 && prof>6 && a==1 && ganador==0
        mejorPuntuacion=0;
        a=2;
    end
    if isMax==true && a==1 %si le toca mover a la maquina
        mejorPuntuacion=-Inf;
```

```
        for k=1:length(disponibleaux)
            tableroaux(disponibleaux(k))=-1;
            sig_disponible =find(tableroaux==0);
            puntuacion=minimax_poda(tableroaux,prof+1,alpha,beta, false , sig_disponible );
            tableroaux(disponibleaux(k))=0;
            if puntuacion>mejorPuntuacion
                mejorPuntuacion=puntuacion;
                mejorMov=disponibleaux(k);
            end
            alpha=max(alpha,puntuacion);
            if beta<=alpha
                break
            end
        end
    elseif isMax==false && a==1 %si le toca mover al usuario
        mejorPuntuacion=Inf;
        for k=1:length(disponibleaux)
            tableroaux(disponibleaux(k))=1;
            sig_disponible =find(tableroaux==0);
            puntuacion=minimax_poda(tableroaux,prof+1,alpha,beta, true , sig_disponible );
            tableroaux(disponibleaux(k))=0;
            if puntuacion<mejorPuntuacion
                mejorPuntuacion=puntuacion;
                mejorMov=disponibleaux(k);
            end
            beta=min(beta,puntuacion);
            if beta<=alpha
                break
            end
        end
    end
end

function movUsuario(b,~)
    v=get(b,'Position');
    x=v(1); y=v(2);
    ii =floor((n-(x-margeninf-borde/2)/(ladocasilla+raya)));
    jj =floor((y-margenlat-borde/2)/(ladocasilla+raya)+1);
    casilla =(n^2+1)-((ii-1)*n+jj);
    pulsa=1;
    stop(tiempo_X)
end

function jugadorActual = comienzoAleatorio
    if rand(1)<0.5
        jugadorActual=jugadores(1);
    else
        jugadorActual=jugadores(2);
    end
end

function [ganador,a] = compruebaGanador(tablero)
    a=1;
    ganador=0;
    switch n
        case 3
            % Comprobacion lineas horizontales
            for i=1:3
                if tablero(i,1)==tablero(i,2) && tablero(i,2)==tablero(i,3) && tablero(i,1)~=0
                    a=2;
                    ganador=tablero(i,1);
                    break
```

```
            end
        end
        % Comprobacion lineas  verticales
        for  j=1:3
            if  tablero (1, j)==tablero (2, j)  && tablero(2, j)==tablero (3, j)  && tablero(1, j)~=0
                a=2;
                ganador=tablero (1, j);
                break
            end
        end
        % Comprobacion diagonal principal
        if  tablero (1,1)==tablero (2,2)  && tablero(2,2)==tablero (3,3)  && tablero(1,1)~=0
            a=2;
            ganador=tablero (2,2);
        end
        % Comprobacion diagonal secundaria
        if  tablero (3,1)==tablero (2,2)  && tablero(2,2)==tablero (1,3)  && tablero(3,1)~=0
            a=2;
            ganador=tablero (3,1);
        end
        % Empate
        x=find(tablero~=0);
        if  length(x)==n^2 && ganador==0 %si no hay casillas  libres  y  no hay  ganador  tenemos un
            empate
            a=2;
        end
    case  4
        for  i=1:4
            if  tablero (i,1)==tablero (i,2)  && tablero(i,2)==tablero (i,3)  && tablero(i,3)==tablero(i
                ,4)  && tablero(i,1)~=0
                a=2;
                ganador=tablero (i,1);
                break
            end
        end
        for  j=1:4
            if  tablero (1, j)==tablero (2, j)  && tablero(2, j)==tablero (3, j)  && tablero(3, j)==tablero
                (4, j)  && tablero(1, j)~=0
                a=2;
                ganador=tablero (1, j);
                break
            end
        end
        if  tablero (1,1)==tablero (2,2)  && tablero(2,2)==tablero (3,3)  && tablero(3,3)==tablero (4,4)
            && tablero(1,1)~=0
            a=2;
            ganador=tablero (2,2);
        end
        if  tablero (4,1)==tablero (3,2)  && tablero(3,2)==tablero (2,3)  && tablero(2,3)==tablero (1,4)
            && tablero(4,1)~=0
            a=2;
            ganador=tablero (4,1);
        end
        x=find(tablero~=0);
        if  length(x)==n^2 && ganador==0 %si no hay casillas  libres  y  no hay  ganador  tenemos un
            empate
            a=2;
        end
    case  5
        for  i=1:5
            if  tablero (i,1)==tablero (i,2)  && tablero(i,2)==tablero (i,3)  && tablero(i,3)==tablero(i
                ,4)  && tablero(i,4)==tablero(i,5)  && tablero(i,1)~=0
```

```
                    a=2;
                    ganador=tablero(i,1);
                    break
                end
            end
            for j=1:5
                if tablero(1,j)==tablero(2,j) && tablero(2,j)==tablero(3,j) && tablero(3,j)==tablero
                    (4,j) && tablero(4,j)==tablero(5,j) && tablero(1,j)~=0
                    a=2;
                    ganador=tablero(1,j);
                    break
                end
            end
            if tablero(1,1)==tablero(2,2) && tablero(2,2)==tablero(3,3) && tablero(3,3)==tablero(4,4)
                && tablero(4,4)==tablero(5,5) && tablero(1,1)~=0
                a=2;
                ganador=tablero(2,2);
            end
            if tablero(5,1)==tablero(4,2) && tablero(4,2)==tablero(3,3) && tablero(3,3)==tablero(2,4)
                && tablero(2,4)==tablero(1,5) && tablero(5,1)~=0
                a=2;
                ganador=tablero(5,1);
            end
            x=find(tablero~=0);
            if length(x)==n^2 && ganador==0 %si no hay casillas libres y no hay ganador tenemos un
                empate
                a=2;
            end
        end
    end

    function matatimer(tiempo)
        delete(tiempo)
    end

    function cierra(~,~)
        delete(timerfindall)
    end

    end
```

Código A.2 Generación de fichas.

```
clear all, close all, clc

fondo=imread('madera.jpg');

jj=1:100;
J=jj(ones(1,100),:);
I=J';
fondo2=fondo(1:100,1:100,:);
roja=fondo2(:,:,1);
verde=fondo2(:,:,2);
azul=fondo2(:,:,3);
azul((J-I>-12 & J-I<12)|(J-(100-I+1)>-12 & J-(100-I+1)<12))=250;
verde((J-I>-12 & J-I<12)|(J-(100-I+1)>-12 & J-(100-I+1)<12))=verde((J-I>-12 & J-I<12)|(J-(100-I+1)
    >-12 & J-(100-I+1)<12))/3;
roja((J-I>-12 & J-I<12)|(J-(100-I+1)>-12 & J-(100-I+1)<12))=roja((J-I>-12 & J-I<12)|(J-(100-I+1)>-12
    & J-(100-I+1)<12))/3;
fondo2(:,:,1)=roja;
fondo2(:,:,2)=verde;
```

```
fondo2 (:,:,3) =azul;
figure
a1=axes('position',[0 0 1 1]);
X=image(fondo2);
axis off

jj=1:100;
J=jj(ones(1,100),:);
I=J';
fondo2=fondo(1:100,1:100,:);
roja=fondo2(:,:,1);
verde=fondo2(:,:,2);
azul=fondo2(:,:,3);
roja((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I-50).^2<47^2)=250;
verde((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I-50).^2<47^2)=verde((J-50).^2+(I-50).^2>30^2 & (J-50)
    .^2+(I-50).^2<47^2)/3;
azul((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I-50).^2<47^2)=azul((J-50).^2+(I-50).^2>30^2 & (J-50).^2+(I
    -50).^2<47^2)/3;
fondo2(:,:,1) =roja;
fondo2(:,:,2) =verde;
fondo2(:,:,3) =azul;
figure
a1=axes('position',[0 0 1 1]);
X=image(fondo2);
axis off
```

Índice de Figuras

Índice de Códigos

Bibliografía

[1] Bolon, T. (2013). *How to never lose at Tic-Tac-Toe*. BookCountry.

[2] Zaslavsky, C. (1982). *Tic tac toe : and other three-in-a row games from ancient Egypt to the modern computer*. Harpercollins.

[3] Uiterwijk, JWHM. & van den Herik, HJ. (2000). *The Advantage of the Initiative. Information Sciences*. (Vol. 122(1), pp. 43-58). Department of Computer Science, Institute for Knowledge and Agent Technology IKAT, Universiteit Maastricht. https://doi.org/10.1016/S0020-0255(99)00095-X

[4] van den Herik, HJ. & Uiterwijk, JWHM. & van Rijswijck, J. (2002). *Games solved: Now and in the future, Artificial Intelligence*. (Vol. 134(1–2), pp. 277-311). Department of Computer Science, Institute for Knowledge and Agent Technology IKAT, Universiteit Maastricht & Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada. https://doi.org/10.1016/S0004-3702(01)00152-7

[5] Russell, SJ. & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4^{th} ed.). Pearson Education.

[6] Kutschera, A. (7 de abril de 2018). *The best opening move in a game of tic-tac-toe*. http://blog.maxant.co.uk/pebble/2018/04/07/1523086680000.html

[7] GeeksforGeeks. (31 de marzo de 2021). *Minimax Algorithm in Game Theory | Set 1 (Introduction)*. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[8] GeeksforGeeks. (9 de diciembre de 2019). *Minimax Algorithm in Game Theory | Set 2 (Introduction to Evaluation Function)*. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-2-evaluation-function/

[9] GeeksforGeeks. (12 de junio de 2021). *Minimax Algorithm in Game Theory | Set 3 (Tic-Tac-Toe AI – Finding optimal move)*. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/

[10] GeeksforGeeks. (15 de junio de 2021). *Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)*. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

[11] Sancho Caparrini, F. (24 de octubre de 2019). *Minimax: Juegos con adversario*. http://www.cs.us.es/ fsancho/?e=107

[12] The Coding Train. (11 de diciembre de 2019). *Coding Challenge 154: Tic Tac Toe AI with Minimax Algorithm*. [Archivo de vídeo]. Youtube. https://www.youtube.com/watch?v=trKjYdBASyQ

[13] Sebastian Lague. (20 de abril de 2018). *Algorithms Explained – minimax and alpha-beta pruning*. [Archivo de vídeo]. Youtube. https://www.youtube.com/watch?v=l-hh51ncgDI

[14] MIT OpenCourseWare (Instructor: Patrick Winston).(10 de enero de 2014). *6. Search: Games, Minimax, and Alpha-Beta*. [Archivo de vídeo]. Youtube. https://youtube.com/watch?v=STjW3eH0Cik

[15] Lección 3: Funciones y Gestión de Códigos. (2020). *Matematica Computacional*. Departamento de Matemática Aplicada II, Universidad de Sevilla. Sin publicar.

[16] Lección 4: Objetos Gráficos y Objetos Multimedia. (2020). *Matematica Computacional*. Departamento de Matemática Aplicada II, Universidad de Sevilla. Sin publicar.

[17] Lección 5. Aplicaciones en Matlab (Apps). (2020). *Matematica Computacional*. Departamento de Matemática Aplicada II, Universidad de Sevilla. Sin publicar.