# Diseño de una Interfaz Gráfica para el Problema del Caballo de Ajedrez

Luis Gerardo de la Fraga

Sección de Computación

CINVESTAV-IPN. Departamento de Ingeniería Eléctrica

Av. Instituto Politécnico Nacional 2508. 07300 México, D.F.

E-mail: fraga@cs.cinvestav.mx

## Resumen

El problema del caballo consiste en mover la pieza del caballo por todos las casillas de un tablero de ajedrez sin volver a pasar por un misma casilla. En este trabajo se presenta una modificación del algoritmo en [1] y el desarrollo de una interfaz gráfica en objetos usando Qt [2] para resolver el problema

**Palabras clave:** Diseño de interfaces gráficas, Programación en Objetos, Algoritmo del caballo, Optimización, Graficación.

## 1 Introducción

La pieza del caballo de ajedrez se mueve un cuadro en una dirección y luego una posición en diagonal. En la Fig. 1 la pieza del caballo está situada en la posición d4 [1] sobre el tablero y tiene ocho opciones para moverse en la tirada siguiente: a la posición b5, b3, c2, e2, f3, f5, e6 ó c6.

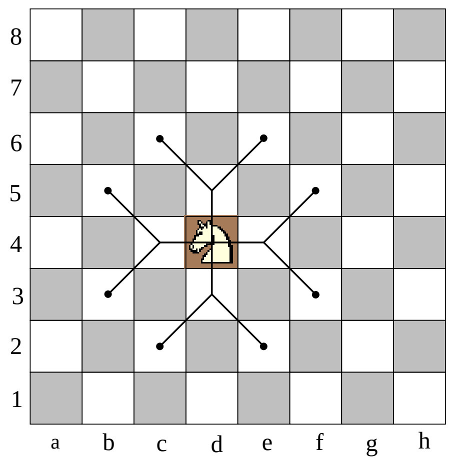


Figura 1: Un tablero de ajedrez con la pieza del caballo en la posición d4. Se marca con líneas los posibles movimientos del caballo, a b5, b3, c2, e2, f3, f5, e6 ó c6.

El *problema del caballo* consiste en mover el caballo por el tablero, de manera que visite todos las casillas sin volver a pasar por una casilla que ya visitó. O, en la jerga del ajedrez, el problema consiste en mover 64 veces el ca-

[1] En todo este artículo se usará la notación cartesiana para representar los movimientos del caballo. De forma estricta, según la notación, hay que anteponer una "C" a la coordenada para indicar que se mueve un caballo, sin embargo, como sólo hay una pieza, se simplifica aquí la notación presentando solamente la coordenada.

ballo sobre el tablero sin repetir una posición. Por ejemplo, una solución al problema puede ser el camino (o juego) 1. a1, 2. c2, 3. e3, 4. f1, 5. h2, ... , etcétera, donde el caballo está inicialmente en la casilla a1, hasta completar 64 tiradas y sin repetir la posición en ninguna de ellas. El problema del recorrido del caballo fue propuesto originalmente por el matemático Euler.

Un tablero normal de ajedrez es de tamaño 8×8 casillas. Se puede generalizar el problema del caballo a un tablero de tamaño $N \times N$ casillas, donde $N$ es mayor que tres. Es fácil comprobar, observando la Fig. 1, que es imposible mover un caballo en un tablero de tamaño menor a tres casillas por lado.

Para tratar de resolver este problema del caballo en un tablero normal sería necesario ir marcando de alguna manera cada casilla visitada. Se invita al lector a tratar de hacerlo para verificar que es una tarea engorrosa. Pero puede solucionarse por medio de una interfaz gráfica que lo resuelva sobre la pantalla del monitor de la computadora. Entonces, este es un problema donde la construcción de una interfaz gráfica está claramente justificado.

Qt es una caja de herramientas para el desarrollo de Interfaces Gráficas de Usuario (GUI, del inglés Graphical User Interface) multiplataforma. Qt es un producto de Troll Tech [2]. Qt es soportado por todas las variantes de Microsoft Windows y Unix/X Windows. Es distribuido bajo una licencia GPL, lo que significa que para el desarrollo de software público no es necesario comprar una licencia para su uso. Si se quiere tener software propietario es necesario comprar una licencia. También si se realiza software para ambiente Windows es necesario comprar el programa y la licencia.

Qt permite el desarrollo de prototipos rápidos y es gratuito bajo Linux, aparte de que tiene una excelente documentación, por estas razones fue escogido para realizar la interfaz gráfica para el problema del caballo. Qt introduce el mecanismo de señales–ranuras para el intercambio de mensajes entre objetos que es bastante intuitivo, en comparación al mecanismo de retrollamadas del desarrollo en X Windows [3]/ Motif [4]. Las señales son enviadas por un objeto a la ranura de otro. Este mecanismo provoca un sobreuso del procesador porque hay que usar un metacompilador (llamado MOC, Meta Object Compiler, en Qt) para traducir el código de C++ con señales y ranuras a código C++ simple.

# 2 Algoritmo para Resolver el Problema del Caballo

En [1] se propone un algoritmo para resolver el problema del caballo que será explicado a continuación. El movimiento del caballo puede representarse con dos arreglos de tamaño ocho; con la ayuda de la Fig. 2 los arreglos podrían ser a[8] = { 2, 1, -1, -2, -2, -1, 1, 2}, para indicar el movimiento horizontal, y b[8] = { -1, -2, -2, -1, 1, 2, 2, 1 }, para indicar el movimiento vertical.

Para el movimiento del caballo sobre el tablero se considera la heurística de *accesibilidad*. Esta se construye considerando en cada casilla el número de movimientos que puede hacer el caballo. La matriz de accesibilidad para un tablero normal puede verse en la Fig. 3 En una posición dada, se calcula la accesibilidad de todas de las posibles tiradas (máximo ocho) y se escoge la tirada que está libre (en la que no haya ya pasado el caballo) y tenga la menor accesibilidad. La idea de esta estrategia es mover el caballo primero a las casillas más problemáticas, y dejar abiertas aquellas que son

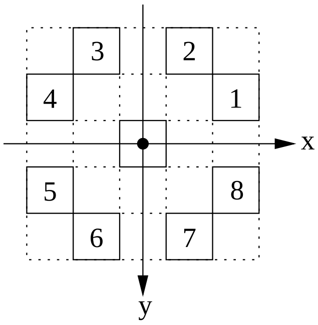


Figura 2: Esquema para resolver la tirada del caballo. El movimiento se puede representar con dos arreglos unidimensional, cada uno para marcar las $x$'s y $y$'s de las coordenadas 1. $(2,-1)$, 2. $(1,-2)$, 3. $(-1,-2)$, 4. $(-2,-1)$, 5. $(-2,1)$, 6. $(-1,2)$, 7. $(1,2)$, y 8. $(2,1)$. La posición actual del caballo es $(0,0)$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |
| 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |

Figura 3: La matriz de accesibilidad. Cada número representa el número de movimientos que el caballo puede hacer.

más fáciles de llegar, a fin de que conforme se vaya congestionando el tablero, se tenga una mayor probabilidad de encontrar casillas vacías al seguir tirando. Una vez que se ha escogido la tirada, debe restarse ésta de la matriz de accesibilidad.

Este algoritmo funciona en 63 de los 64 juegos posibles (cada juego posible corresponde a cada posición inicial, y hay 64 posiciones iniciales posibles). Cuando el caballo inicia en c5 se logran hacer 60 tiradas, y aunque esto es una buena aproximación, no es una solución. El problema que se descubrió aquí es que cuando se tienen varias posibles tiradas se escoge la primera de ellas de acuerdo al orden mostrado en la Fig. 2. Para corregir este nuevo problema se modificó el algoritmo como sigue: si se tienen dos o más posibles tiradas con la misma mínima accesibilidad, se multiplica la accesibilidad de las posibles tiradas por el valor de una *matriz de dirección*. Esta matriz de dirección tiene valores de 1 para las casillas en el borde del tablero y de 2 para las demás casillas. De esta forma el caballo tenderá a pasar primero por las casillas de los bordes.

Con este algoritmo modificado de obtuvieron soluciones para todos los 64 juegos posibles. Son diferentes las soluciones del algoritmo original y el modificado cuando el caballo inicia en las posiciones d8, e8, h8, e7, g7, e6, g6, a5, c5, h5, a4, c4, e4, f4, g4, d3, c2, h2, b1, g1, y h1; esto es, son diferentes 21 posiciones de 64, que corresponden a un tercio de los juegos. Estos datos muestras que existen más de 64 soluciones para el problema del caballo.

Un ejemplo de un juego, o una solución del problema, cuando la posición inicial es a1, es: 1 a1, 2 b3, 3 c1, 4 a2, 5 b4, 6 a6, 7 b8, 8 d7, 9 f8, 10 h7, 11 g5, 12 h3, 13 g1, 14 e2, 15 g3, 16 h1, 17 f2, 18 d1, 19 b2, 20 a4, 21 c3, 22 b1, 23 a3, 24 c2, 25 e1, 26 d3, 27 c5, 28 e4, 29 d2, 30 f1, 31 h2, 32 f3, 33 h4, 34 g2, 35 e3, 36 g4, 37 h6, 38 g8, 39 f6, 40 h5, 41 f4, 42 d5, 43 b6, 44 a8, 45 c7, 46 e8, 47 g7, 48 e6, 49 d4, 50 b5, 51 a7, 52 c8, 53 e7, 54 f5, 55 d6, 56 c4, 57 a5, 58 b7, 59 d8, 60 c6, 61 e5, 62 f7, 63 h8, 64 g6. Esta solución generada es igual indistintamente al algoritmo de solución que se use.

# 3 Diseño de la Interfaz Gráfica

Los componentes para diseñar una interfaz gráfica (de aquí en adelante se abreviará como GUI) se llaman *widgets*. Widgets comunes son: botones, barras de scroll, cuadros de diálogo, menús, botones con imágenes (o iconos), ventanas, etc.

La librería de Qt provee toda la funcionalidad para el dibujo de widgets, sus estilos de dibujo, y para controlar los eventos de teclado y ratón, además de poder diseñar nuevos widgets inexistentes, como podría ser una manija circular (como las que controlan el volumen en un radio). El programador se enfoca a a la funcionalidad y la comunicación entre los diferentes widgets.

Para el problema del caballo, en términos generales, la GUI debe contar con:

1. El dibujo de un tablero de ajedrez de tamaño 3×3 hasta 8×8

2. Dibujar la imagen de un caballo sobre las casillas del tablero.

3. Tener los widgets necesarios para iniciar un juego (esto es, encontrar una solución), salir de la aplicación. iniciar un nuevo juego y seleccionar entre que juegue la computadora con el algoritmo programado, o dejar que el usuario intenté por él mismo encontrar una solución.

Como especificación para la aplicación, el tamaño del tablero se selecciona como un parámetro para el programa que se llamará *knight* (caballo, en inglés).

La GUI diseñada puede verse en la Fig. 4. Esta consta de dos partes, la izquierda, con los widgets de control, y la derecha con el tablero. En la parte izquierda, de arriba hacia abajo, consta de:

1. Un widget de tipo QLabel para desplegar mensajes sobre el estado del juego.

2. Un widget QPushButton para iniciar un juego.

3. Un widget QComboBox para selecionar el jugador (la computadora o un humano, el usuario ).

4. Dos widgets QSpinBox para seleccionar la posición por medio del teclado.

5. Un widget QLabel para mostrar la posición actual del caballo.

6. Dos QPushButton más para iniciar un nuevo juego y para salir de la aplicación.

La parte de la derecha, que forma el teclado, se diseñó de manera que cada cuadro es también un QButton, esto es, se puede seleccionar la posición del caballo al hacer un click sobre una casilla.

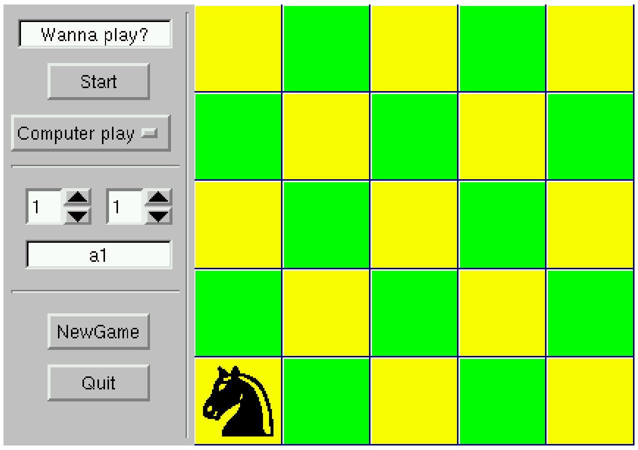


Figura 4: Imagen de la interfaz diseñada. Se muestra la vista inicial al arrancar con un tablero de tamaño 5×5.

## 3.1 Jerarquía de objetos

La GUI diseñada consta de tres objetos:

1. Un nuevo widget de tipo QButton para una casilla del tablero,

2. Un objeto para todo el tablero.

3. Un objeto para enmarcar los widgets de control (presentados en la sección anterior) y el tablero.

Cada uno de estos tres objetos se identificará con el nombre *Botón*, *Tablero* y *Juego*, respectivamente. El objeto Tablero contiene un arreglo de $N \times N$ objetos Botón, donde $N$ es el número de casillas por lado del tablero. El objeto Juego contiene un objeto Tablero. Los objetos Tablero y Juego son también widgets generales de tipo QWidget (de esta forma pueden comunicarse por medio de señales y ranuras).

Escoger los objetos para el diseño fue de forma bastante intuitiva como se describió al inicio de la sección 3.

El soporte para manejo del teclado viene incluida automáticamente por Qt: con la tecla TAB se puede recorrer los widgets de control; con la tecla ESPACIO se oprime cualquiera de los tres widgets QPushButton y el widget QComboBox. Al pasar el control del teclado (llamado *focus* en Qt) por un QSpinBox se puede introducir la posición editando su contenido, que debe ser un número entre 1 y N, inclusive.

## 3.2 Uso y comportamiento de la interfaz

Un juego consta de las siguientes etapas: 1) Seleccionar la posición inicial del caballo, 2) Seleccionar al jugador, y 3) Buscar la solución. Para marcar la posición del caballo durante el recorrido se escogió imprimir el número de la tirada en la casilla recorrida.

La etapa (1), la posición inicial, puede seleccionarse de dos formas: 1.a) Haciendo un click en el Botón correspondiente a una casilla; esto pondrá la nueva posición en los QSpinBox y en el QLabel de la posición. La imagen del caballo se borra de la posición anterior y se dibuja en la nueva selección. 2.a) Oprimiendo los QSpinBox en las puntas de flecha (ver la Fig. 4) o por teclado como ya se explicó al final de la subsección anterior; esto marca la nueva posición en el QLabel de la posición, se borra la imagen del caballo de la posición anterior y se dibuja sobre la nueva posición.

La aparente recursividad en el párrafo anterior —al oprimir una casilla pone al día el contenido de los QSpinBox y la imagen del caballo y al oprimir la flecha de un QSpinBox se pone al día la imagen del caballo en la casilla correspondiente— se resuelve de la forma siguiente: Las casillas borrada y la nueva con la imagen del caballo son manejadas por los QSpinBox. Cuando se oprime con el ratón un Botón, que corresponde a una casilla del tablero, este evalúa si es una casilla diferente de la inicial, y si es diferente pone el valor de la casilla en los QSpinBox. Al ponerse el valor en el QSpinBox se llama al método que pone al día la imagen de los dos Botónes. Cada Botón se comunica al Tablero cuando se oprime por medio de una señal en el Botón y una ranura en el Tablero, y a su vez el Tablero manda una señal a una ranura del objeto Juego.

La etapa (2) se realiza con el widget QComboBox.

La ultima etapa (3) se realiza oprimiendo QPushBotton “Start” (ver Fig. 4). Para iniciar y se pone el letrero “Playing” (jugando) en el primer widget de control. Si es el humano el que juega, se checa que sea una tirada válida cada vez que se oprima un Botón. Se pone el letrero de "Game Over" (juego terminado) cuando se han recorrido todas las casillas o cuando ya no hay otra posibilidad de tirar. Si es la computadora la que juega se pone un reloj (timer) dentro de Tablero por medio de una llamada al método `startTimer(1200)`, que cada 1.2 segundos mandará una señal al método `timerEvent( QTimerEvent * )`, que manejará automáticamente la solución por el algoritmo presentado en la sección 2.

Finalmente se puede parar e iniciar un nuevo juego oprimiendo el widget "NewGame" (nuevo juego) y se puede salir del programa con el widget "Quit".

La solución encontrada por el algoritmo modificado (ver sec. 2) para la posición inicial a1, puede verse en la Fig. 5.


Figura 5: La interfaz gráfica mostrando la solución encontrada por la aplicación cuando el tablero es de tamaño 5×5 y la posición inicial es a1.

# 4 Conclusiones

Se realizó una modificación al algoritmo presentado en [1] para solucionar el problema del recorrido del caballo en un tablero de ajedrez. La modificación propuesta permite encontrar una solución para cualquiera de las 64 posiciones iniciales. Dado que hay 21 soluciones diferente entre el algoritmo original y el modificado, y el algoritmo original falla cuando inicia en la posición c5, se concluye que hay más de 64 soluciones o formas de recorrer el tablero.

Se construyó una interfaz gráfica, para solucionar el problema del caballo, usando las librerías de Qt [2]. Esta librerías permiten del desarrollo de interfaces gráficas en C/C++, y son de uso público bajo el sistema operativo Linux. La documentación de Qt es muy completa (en inglés) y tiene una forma muy intuitiva de comunicar los objetos creados por medio del mecanismo señal/ranura. Qt también ofrece una gran variedad de widgets y permite fácilmente la creación de otros nuevos.

Los archivos fuente en C del algoritmo modificado y los archivos fuente de la interfaz pueden localizarse en la URL http://delta.cs.cinvestav.mv/~fraga/papers/knight/.

# Agradecimientos

Este trabajo ha sido parcialmente apoyado por el proyecto 29944-E del Consejo Nacional de Ciencia y Tecnología (CONACyT).

# Referencias

[1] P.J. Deitel and H.M. Deitel. *C++ how to program*. Printice Hall, 2000.

[2] Qt en internet en la URL http://www.trolltech.com/.

[3] El sistema X Windows en la URL http://www.x.org/.

[4] Documentación de Motif en la URL http://www.opengroup.org/ ó http://www.rahul.net/kenton/xsites.html.