Diseño asistido por computador

AJEDREZ

Aplicación realizada por:

Rafael García Fernández

INTRODUCCIÓN

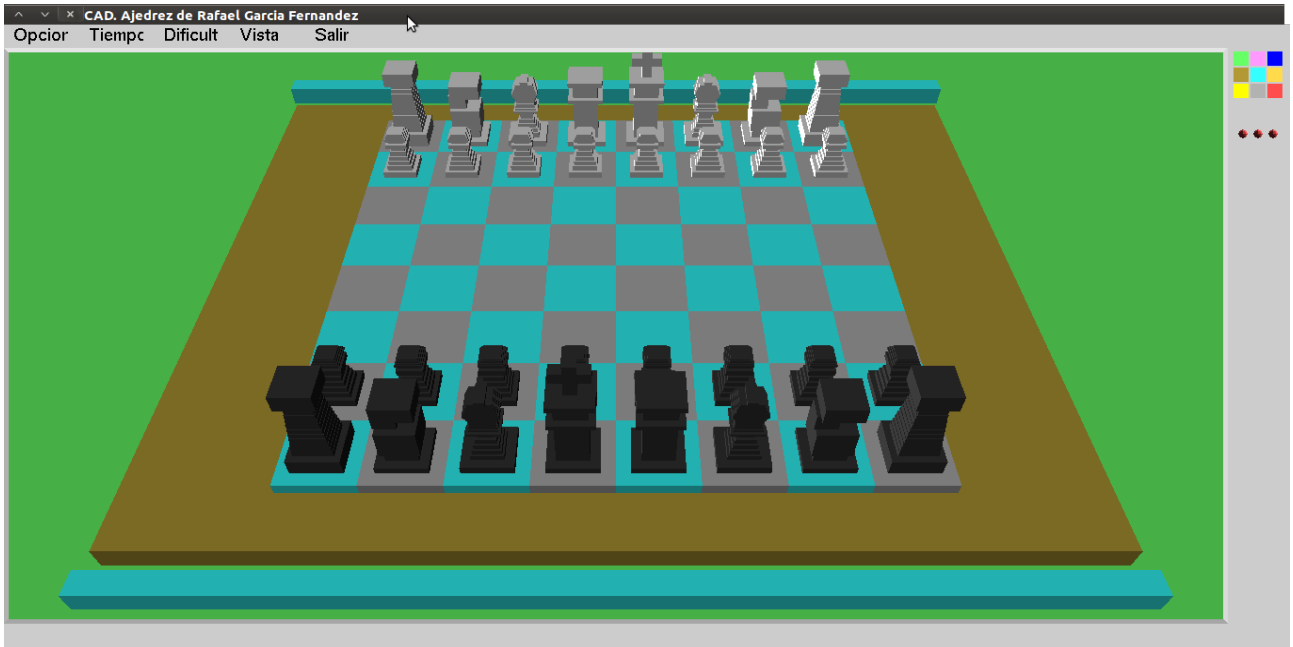
El ajedrez es un juego conocido mundialmente. Jugado entre 2 personas, cada una de las cuales tiene 16 piezas colocadas sobre un tablero formado por 64 casillas. Está considerado como un deporte. Originalmente fue inventado como un juego para personas, pero con la creación de las computadoras y programas comerciales de ajedrez una partida puede ser jugada por 2 personas, por una persona contra un programa de ordenador o incluso por 2 programas de ajedrez entre si.

Se juega sobre un tablero cuadriculado de 8x8 casillas, alternando casillas de color blanco y negro, que constituyen las 64 posibles posiciones sobre las que se puede colocar una pieza en el desarrollo de la partida. Cada jugador comienza la partida con 16 piezas: un rey, una dama, dos alfiles, dos caballos, dos torres y ocho peones. Es un juego de inteligencia en el cual el objetivo es “derrocar” al rey del oponente. Esto se consigue amenazando la casilla que ocupa el rey con alguna de nuestras piezas sin que el otro jugador pueda proteger su rey interponiendo otra pieza entre su rey y la pieza que lo amenaza, mover su rey a una casilla libre de amenaza o comer la pieza que lo esta amenazando. Esto trae como resultado el Jaque Mate y con ello el final de la partida. La partida también puede acabar en tablas por alguno de los siguientes casos: acuerdo común, cuando a ninguno de los jugadores no le quedan suficientes piezas para infringir un jaque mate al rival, se repite 3 veces la misma posición de todas las piezas en el tablero o cuando a un jugador no pueda realizar en su turno ningún movimiento reglamentario, pero el rey no se encuentra ahogado. Este último es también llamado tablas por ahogado.

MANUAL DE JUEGO

Ejecutar la aplicación

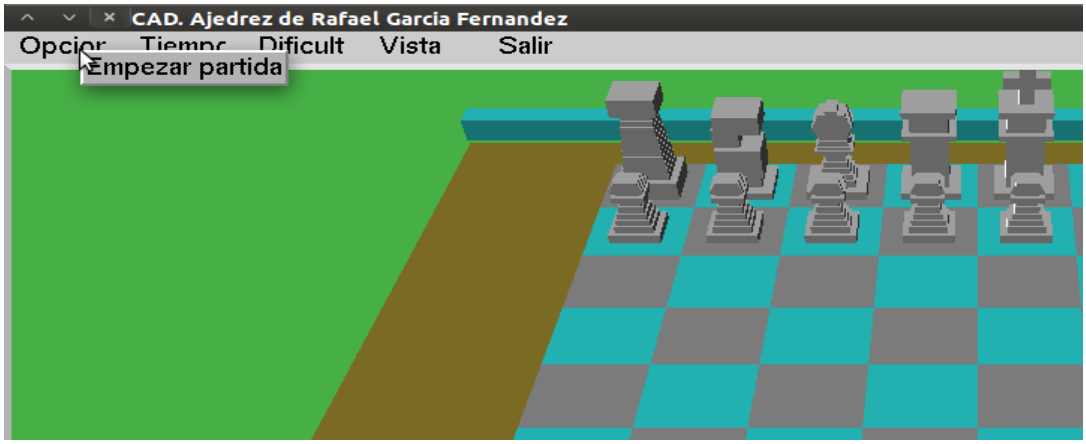
Para ejecutar la aplicación simplemente abrimos una terminal, nos situamos en el directorio en el que se encuentra la aplicación (proyecto) y tecleamos ./ajedrez. Maximizamos y nos encontramos esto:



Captura 1 – Pantalla de inicio

¿Como se juega? Inicio rápido

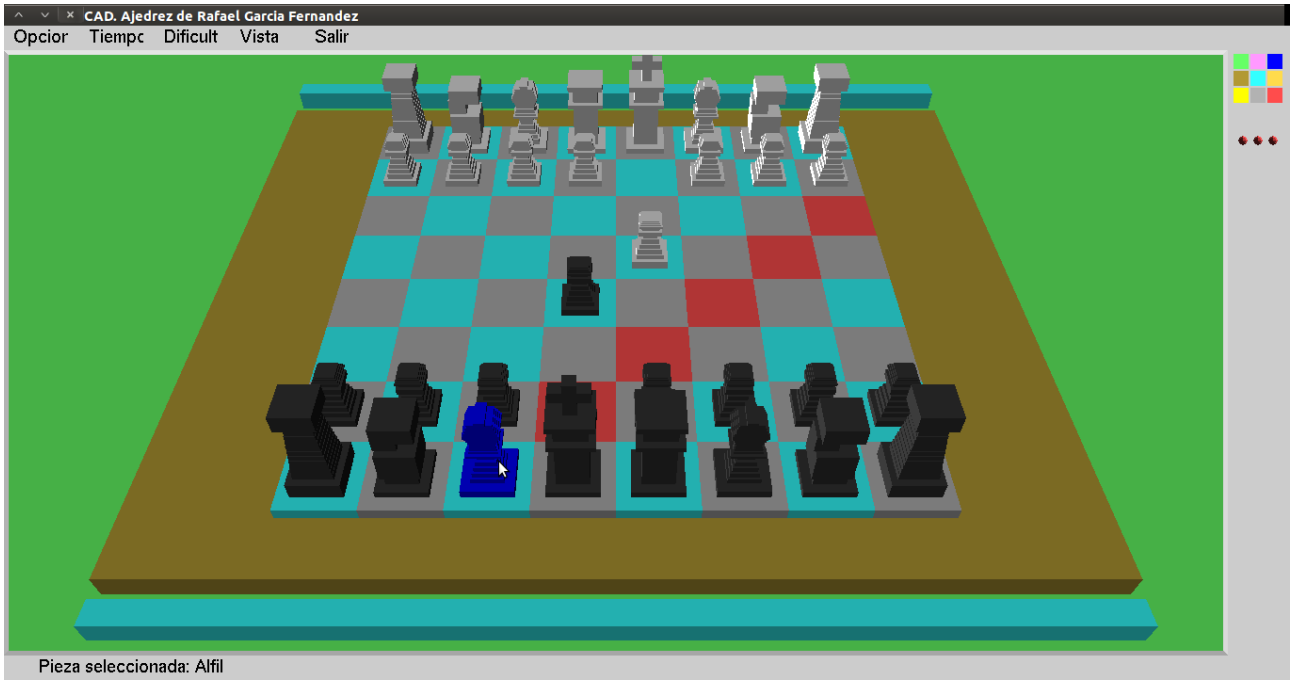
Para empezar a jugar, pinchamos en opciones en el menú de arriba y posteriormente en “Empezar partida”. Una vez hecho esto, la partida se da comenzada y el tiempo (activado por defecto, ver apartado más adelante) correrá para el jugador que maneja las piezas negras.



Captura 2. Empezar partida

Para mover una pieza simplemente se debe hacer click sobre ella. La pieza cambiará su color original por el azul, lo que significa que ha sido seleccionada. Una vez hecho esto, solo nos queda mover a una posición válida para la pieza y para la situación del tablero. Por defecto el modo amateur esta activado, por lo que las casillas iluminadas nos dicen cuales son las casillas a las que nos podemos mover.

Si por casualidad hemos seleccionado una pieza y posteriormente lo hemos pensando mejor y queremos mover otra, simplemente hacemos click en cualquier sitio de la pantalla para anular la selección (la pieza volverá a su color habitual)



Captura 3. Mover pieza

Menús

En la parte superior de la pantalla podemos ver la solapa que contiene los distintos menús existentes en la aplicación. Vamos a verlos uno por uno:

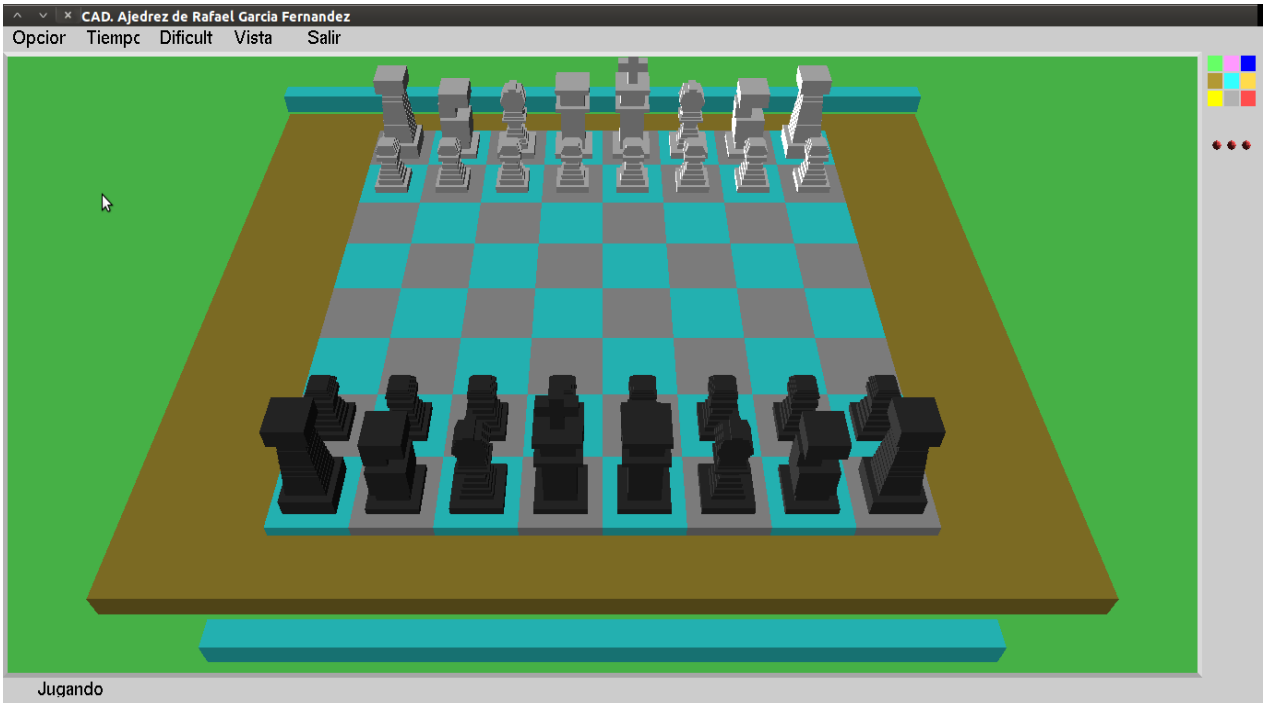
Captura 4. Menús

– Opciones

Contiene solamente la opción “Comenzar partida” ya comentada previamente. Es usada para dar comienzo a la partida.

– Tiempo

Contiene las opciones referentes al tiempo de partida. Con *sin limite* podemos seleccionar que nuestra partida no tengo limite de tiempo, mientras que con *5 minutos* o *10 minutos* elegimos una partida cronometrada con el tiempo estipulado para cada uno de los jugadores. La opción elegida por defecto es la de 5 minutos.



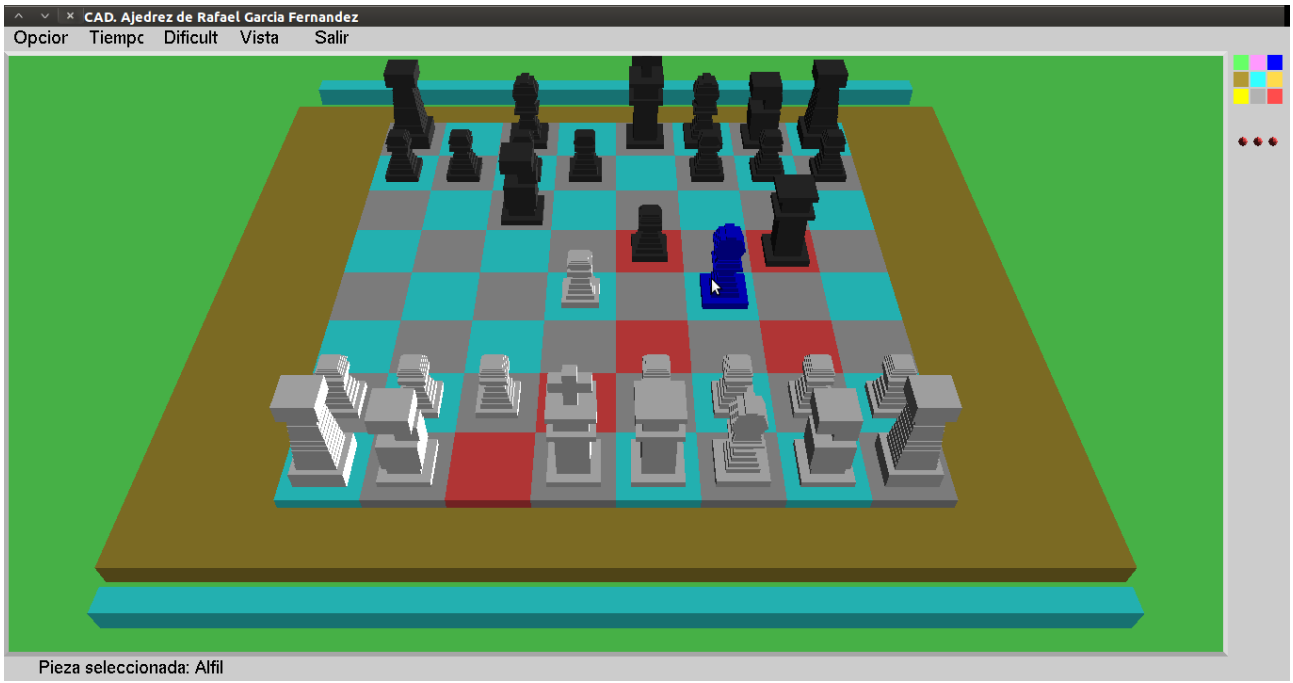
Captura 5. Barras de tiempo

Como podemos ver en esta imagen, el tiempo restante para cada jugador es representado por la barra azul colocada detrás de sus piezas.

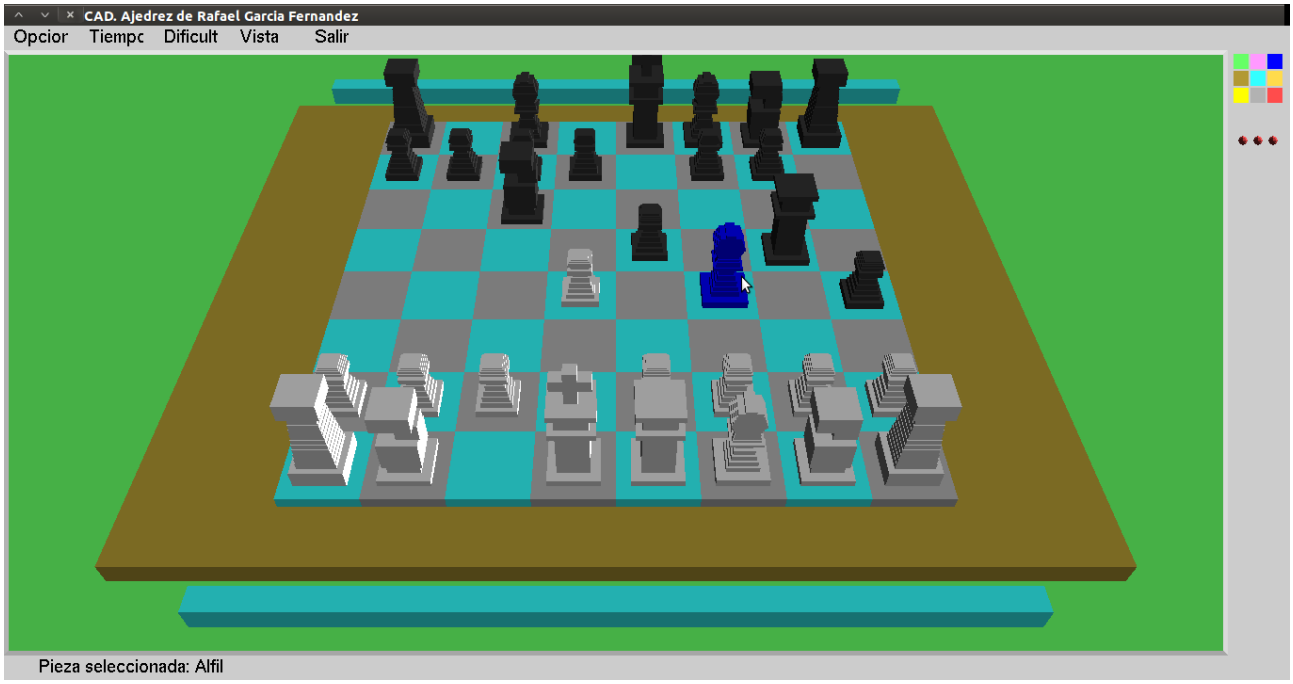
Si por el contrario elegimos jugar la partida sin limite de tiempo, estas 2 barras de juego no saldrán en la imagen.

– Dificultad

En la pestaña de dificultad nos encontramos con 2 opciones: *modo amateur y modo profesional*. En el modo amateur las ayudas para mover las piezas estarán activadas. Cuando una pieza sea elegida, todas las posibles casillas a las que esta puede moverse serán iluminadas de color rojo. Una vez que la pieza sea movida o se cancele la selección de esa pieza, estas casillas volverán a su color original. En el modo profesional estas ayudas están desactivadas, por lo que el usuario deberá conocer a qué casillas puede moverse y a cuales no.



Captura 6. Modo amateur



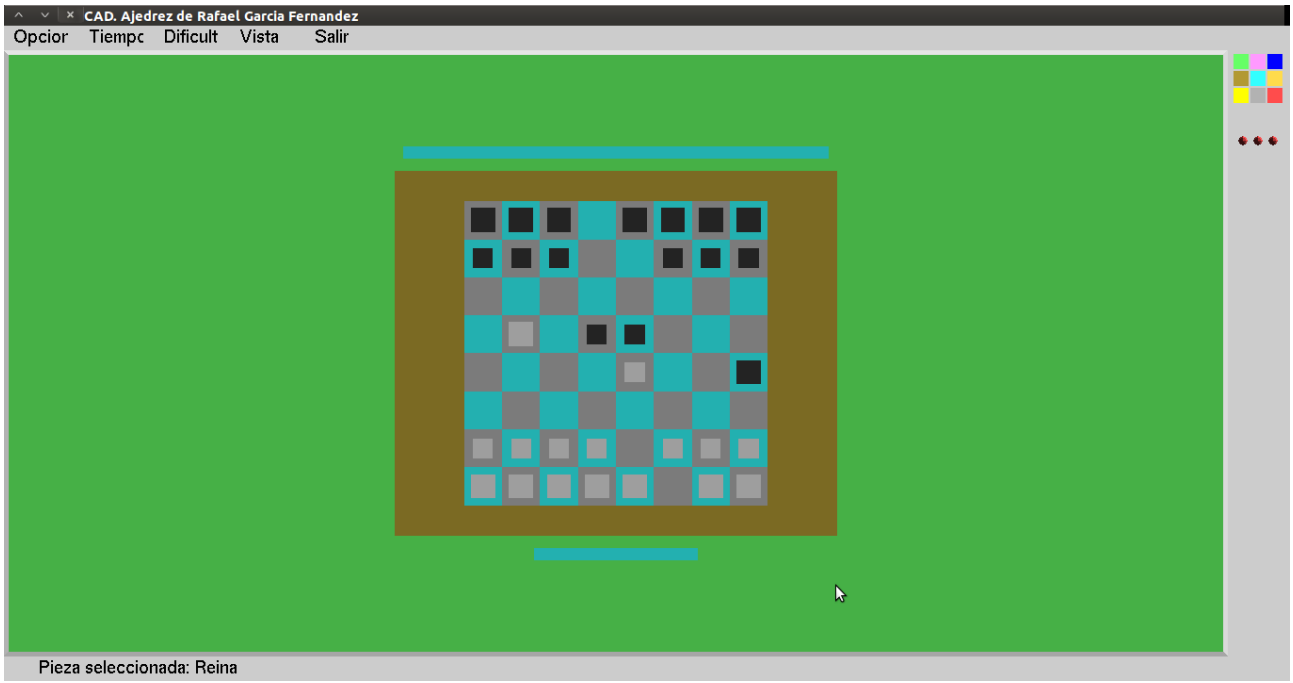
Captura 7. Modo profesional

Esta opción puede ser cambiada durante el desarrollo de la partida.

- **Vistas**

En la pestaña vistas podemos seleccionar el punto de vista desde el que queremos jugar: *2D o 3D.* Por defecto la opción elegida es 3D, ya que la aplicación ha sido desarrollada para ser jugada en 3D. Además, en 2D existe el problema de que las piezas no son reconocibles, por lo que es imposible jugar así. De todos modos, he considerado oportuno dejar la opción ya que en algunos momentos de la partida puede ser útil para el usuario tener un punto de vista de la situación actual del tablero.

Es posible tambien cambiar la vista haciendo uso del teclado. Tecla 2 para cambiar a vista 2D y tecla 3 para cambiar a vista 3D.



Captura 8. Vista 2D

- **Salir**

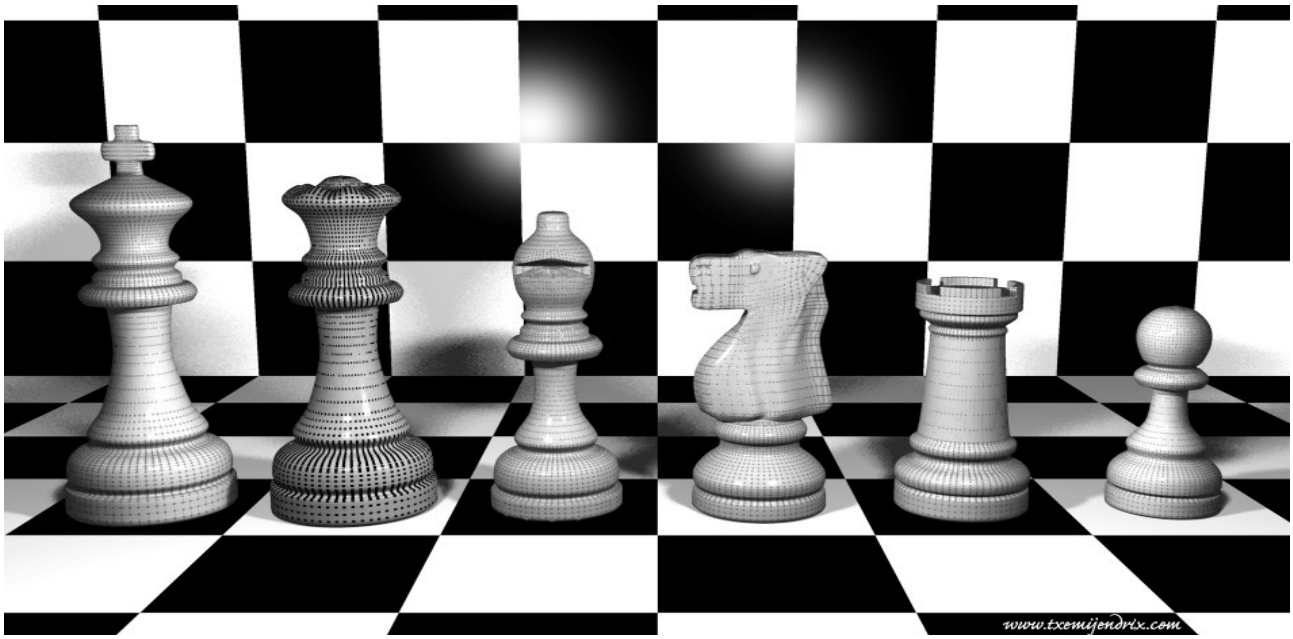
Con la pestaña salir se nos da la opción de cerrar la aplicación. Tras hacer click en esta, nos saldrá la opción de *confirmar.*

Cámara

La cámara por defecto es la que yo he considerado que es la más idónea para poder jugar al juego de forma correcta y cómoda. Sin embargo, hay numerosas teclas con las que puede cambiarse la configuración de esta. No voy a explicarlas ya que forman parte del código que se nos fue entregado por lo que ya las doy por conocidas.

DISEÑO DE PIEZAS

Para el diseño de las piezas, he cogido un modelo de una imagen que encontré en internet y he intentando recrearlo fielmente en la medida de lo posible. Este es el modelo usado:

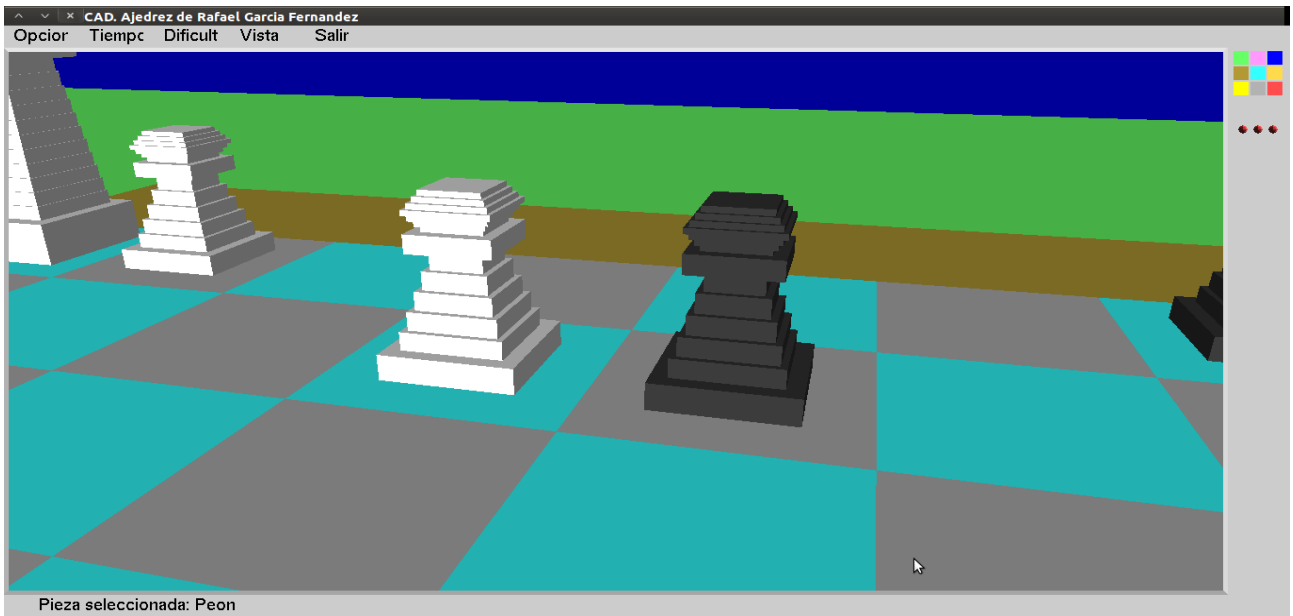


Captura 9. Modelo de piezas

Todas las piezas han sido diseñadas siguiendo un patrón parecido. Este patrón consiste en: base, cuerpo (cuerpo intermedio) y cabeza.

Analizemos pieza por pieza:

- **Peón**



Captura 10. Peón

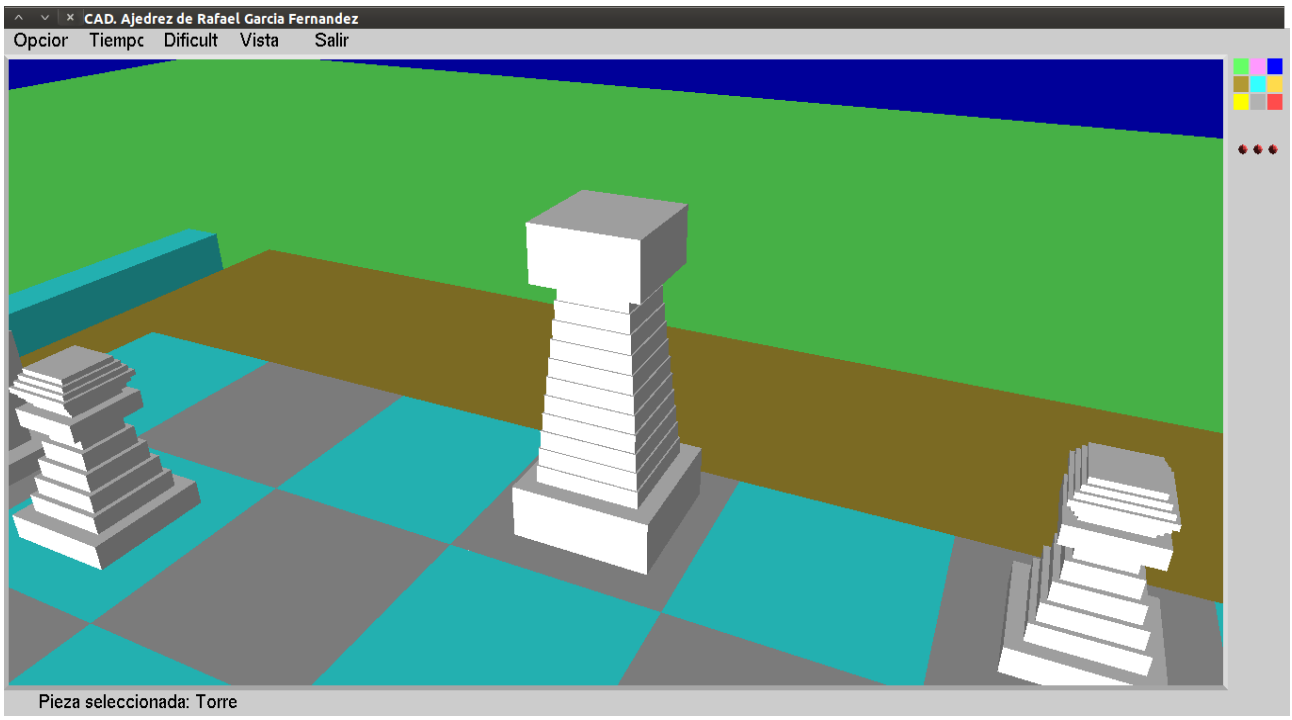
El código utilizado es sencillo. Empezamos por abajo y vamos colocando pequeñas cajas de diferente grosor y altura según nuestro modelo de peón. Una vez terminado, restablecemos el punto donde dibujamos usando un translate de la altura total del peón.

```
void dibujarPeon(int c){

	glPushMatrix();

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //base
	glTranslatef(0, 0, 0);
	caja(3.3, 0.7, 3.3);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cuerpo
	glTranslatef(0, 0.7, 0);
	caja(2.5, 0.5, 2.5);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cuerpo
	glTranslatef(0, 0.5, 0);
	caja(2.2, 0.5, 2.2);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cuerpo
	glTranslatef(0, 0.5, 0);
	caja(1.9, 0.5, 1.9);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cuerpo
	glTranslatef(0, 0.5, 0);
	caja(1.6, 0.5, 1.6);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cuerpo
	glTranslatef(0, 0.5, 0);
	caja(1.3, 0.5, 1.3);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza intermedia
	glTranslatef(0, 0.5, 0);
	caja(2.1, 0.5, 2.1);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.5, 0);
	caja(1.5, 0.15, 1.5);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.15, 0);
	caja(1.7, 0.15, 1.7);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.15, 0);
	caja(1.9, 0.15, 1.9);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.15, 0);
	caja(2.1, 0.15, 2.1);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.15, 0);
	caja(1.9, 0.15, 1.9);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.15, 0);
	caja(1.7, 0.15, 1.7);

	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color[c]); //cabeza
	glTranslatef(0, 0.15, 0);
	caja(1.5, 0.15, 1.5);

	glTranslatef(0, -4.6, 0); //Restablecemos altura

	glPopMatrix(); }
```

El código del resto de las piezas es muy parecido al del peón, solo que cambiando los traslates y las

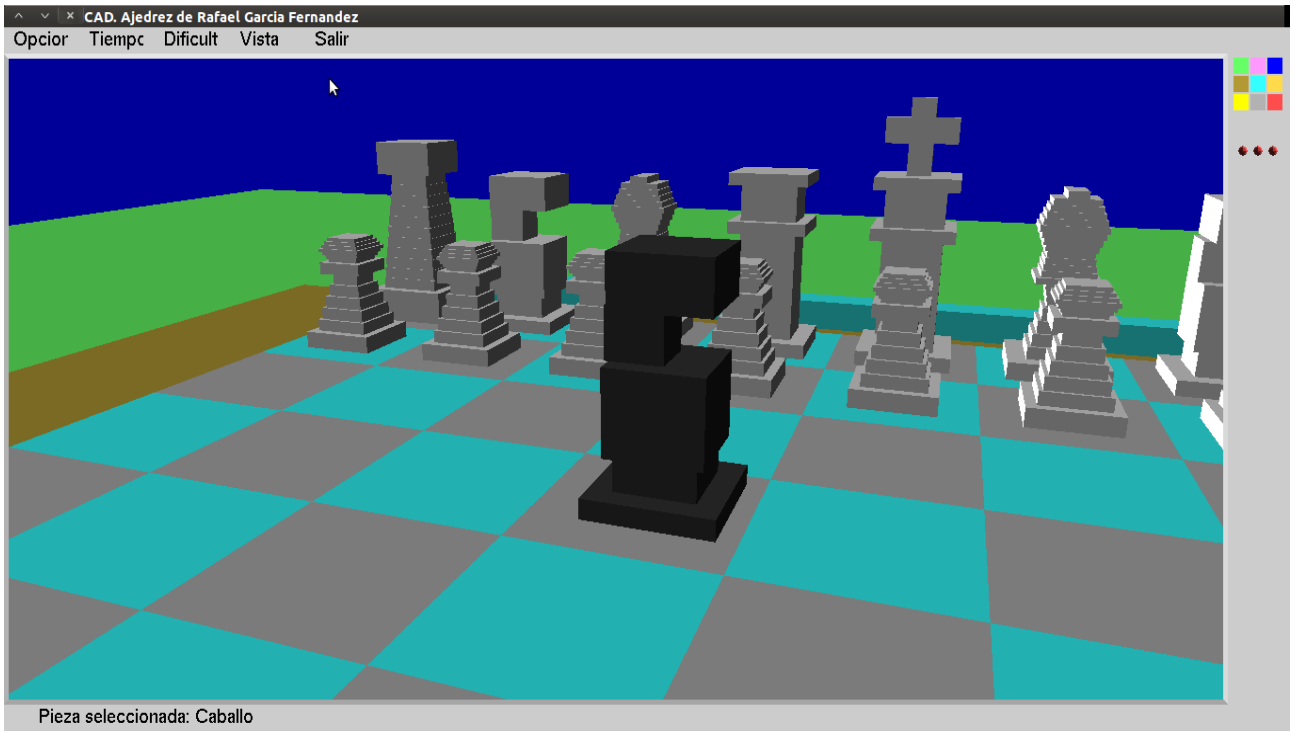
dimensiones, por lo que considero innecesario adjuntaros de aquí en adelante.

- **Torre**



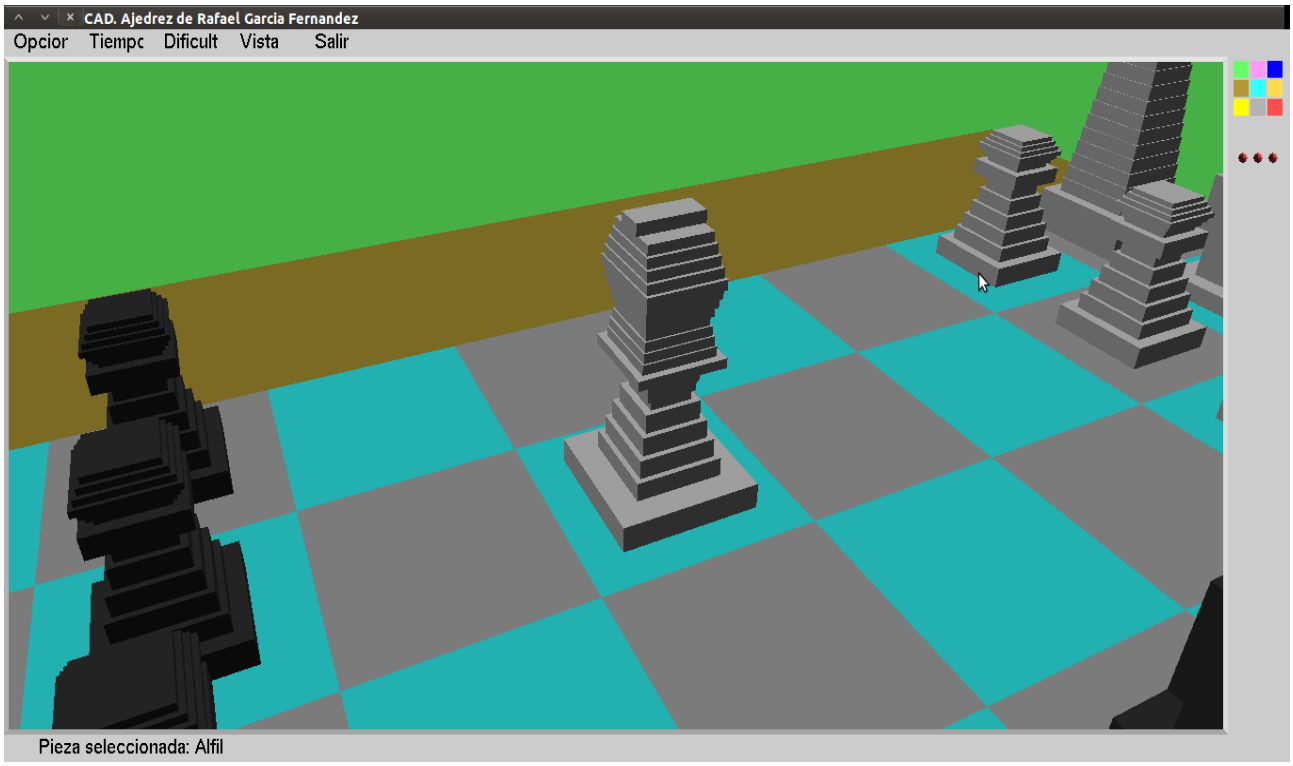
Captura 11. Torre

- **Caballo**



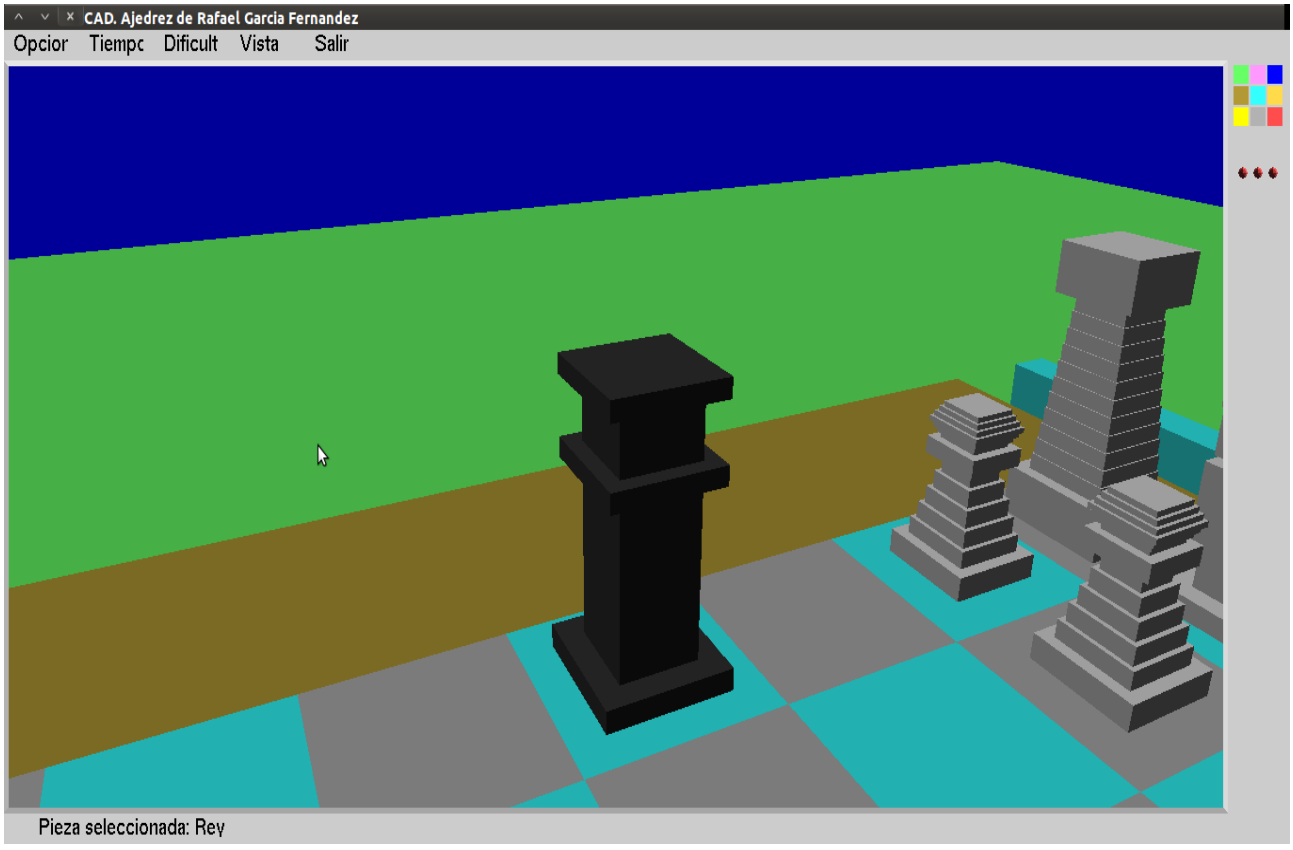
Captura 12. Caballo

- **Alfil**



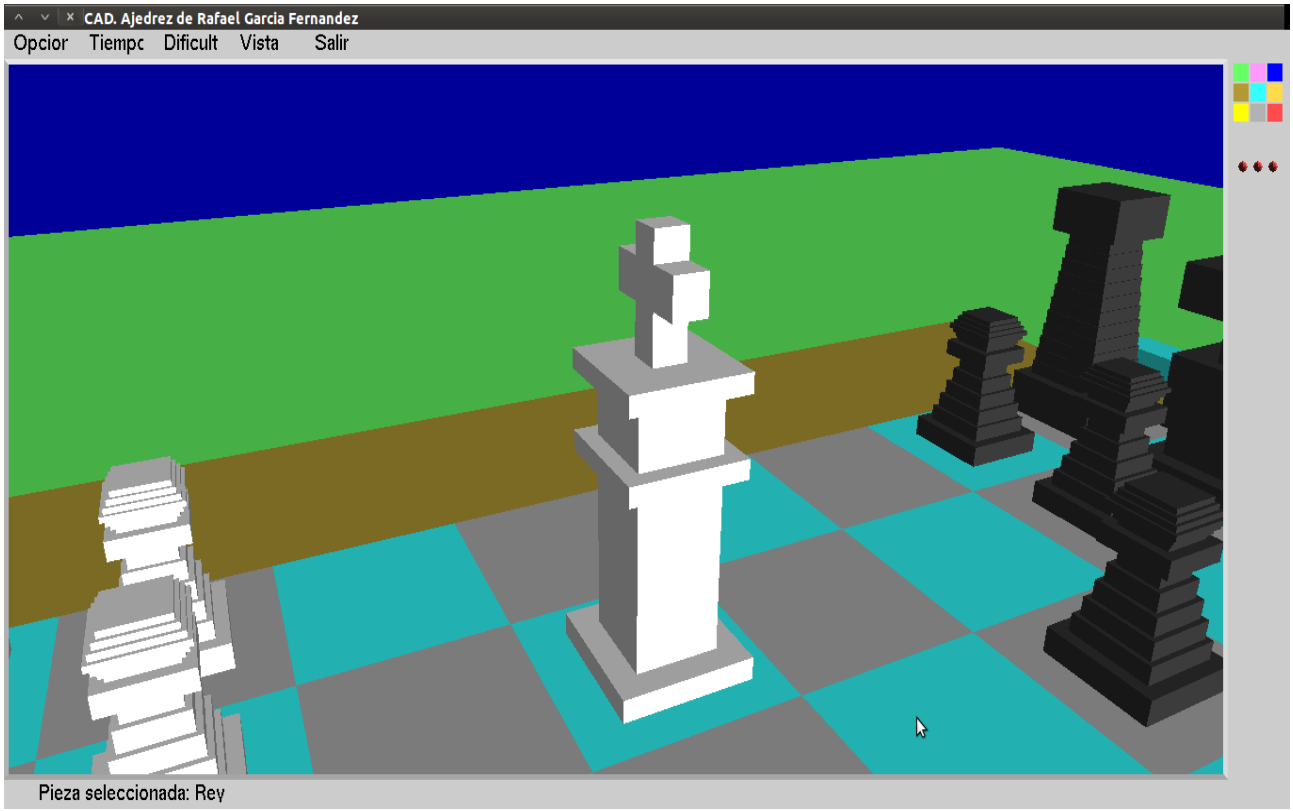
Captura 13. Alfil

- **Reina**



Captura 14. Reina

- **Rey**



Captura 15. Rey

Estructura de datos y Variables globales interesantes

Estructura Pieza

Esta estructura se encarga de todo lo relacionado con las piezas de nuestro ajedrez. Este es el código de la estructura:

```
typedef struct{

    float pospiezax;
    float pospiezay;
    float angX;
    float angY;
    int color;
    int altura;
    int estado;
    char tipo;
    int sobre_casilla;
    float angulo;
    int num_posibles;
    int posibles[50];
    float anguloAux;

} Pieza;
```

Vamos a explicar por encima que es cada uno de ellos. Pospiezax es la coordenada x de la pieza, sin embargo pospiezay no es la coordenada en el plano y, sino la del plano z. Altura es la coordenada del plano y.

Color es obviamente el color de la pieza. Estado hace referencia a si esta viva o ha sido comida. Tipo nos dice si es un peón ('p'), reina ('q'), rey('k'), caballo('c'), torre('t') o alfil ('a'). Sobre_casilla dice sobre que número de casilla esta. Num_posibles y posibles es usado para la iluminación de casillas posibles donde se puede mover la pieza.

Las restantes son atributos que se consideraron en el principio del desarrollo pero finalmente no han sido necesarias.

Estructura Casilla

Esta estructura contiene toda la información sobre cada una de las 64 casillas que tiene nuestro tablero. Vamos a ver de que esta compuesta:

```
typedef struct{

        float poscasillax;
        float poscasillaz;
        int altura;
        int color;
        int estado;
        float dimbase;
        float dimaltura;
        float dimfondo;
        int color_antiguo;

} Casilla;
```

Poscasillax, altura y poscasillaZ se encargan de las coordenadas en el plano X, Y y Z respectivamente. Color se encarga del color de la casilla, estado de si esta ocupada o libre y dimbase, dimaltura y dimfondo son atributos sobre el tamaño de las casillas. Para terminar color_antiguo es una variable utilizada en el ilumbrado de casillas posibles, para poder volver a su color por defecto cuando termine este proceso.

Variables globales interesantes

ambito Pieza piezas[Npiezas] → Vector de piezas
ambito Casilla casillas[Ncasillas] → Vector de casillas
ambito int seleccionada → Entero con número de pieza o casilla seleccionada
ambito int color_original → Color original de una pieza
ambito int comida_blancax → coordinada en plano X de donde deben situarse las piezas blancas que hayan sido comidas
ambito int comida_blancaz → coordinada en plano Z de donde deben situarse las piezas blancas que hayan sido comidas
ambito int comida_negrax → coordinada en plano X de donde deben situarse las piezas negras que hayan sido comidas
ambito int comida_negraz → coordinada en plano Z de donde deben situarse las piezas blancas que hayan sido comidas
int turno_jugador → marca el turno de jugador
int tiempo_jugador1 → tiempo restante de juego del jugador 1
int tiempo_jugador2 → tiempo restante de juego del jugador 2
int modo_tiempo → marca el modo de tiempo elegido
int modo_dificultad → marca el modo de dificultad elegido

PRINCIPALES ALGORITMOS

Veamos primero lo que hacemos dependiendo del estado en el que se encuentre el programa.

Selecciona pieza (Jugando)

Cuando el programa se encuentra en estado “Jugando”, es decir, esperando a que el jugador con turno seleccione una pieza para mover, lo que hacemos es lo siguiente: Usando la función pick, detectamos que número de pieza ha seleccionado. Una vez hecho esto, comprobamos que tipo de pieza es y llamamos a su correspondiente función rellenarPosibles*TIPOPIEZA*, la cual se ocupa de comprobar a que casillas se podría mover y las ilumina si el modo amateur está activado. Además de eso, se encarga de cambiar el color a la pieza y cambiar el estado entre otras cosas. Este es el código:

```
case jugando:

	terminado = juegoTerminado();

	if(terminado){
		gluiOutput( "PARTIDA FINALIZADA");    // Nuestra mensaje en interfaz
	}

	num_pieza = pick(x,y);
	if((num_pieza >= 1 && num_pieza <= 36) && piezas[num_pieza-1].estado != 1){

		if((turno_jugador == 0 && piezas[num_pieza-1].color == negro) || (turno_jugador == 1 && piezas[num_pieza-1].color == blanco)){

			seleccionada = num_pieza-1;

			if(piezas[seleccionada].tipo == 't'){

				gluiOutput( "Pieza seleccionada: Torre");     // Nuestra mensaje en interfaz
				rellenarPosiblesTorre(seleccionada);
			}

			if(piezas[seleccionada].tipo == 'p'){

				gluiOutput( "Pieza seleccionada: Peon");     // Nuestra mensaje en interfaz
				rellenarPosiblesPeon(seleccionada);
			}

			…..... //Igual con resto de piezas

			color_original = piezas[seleccionada].color;
			piezas[num_pieza-1].color = azul;
			xRef = x;
			zRef = y;

			if(modo_dificultad == 1){

				apagarEncendidas();
			}
			estado = piezaseleccionada;
		}
	} break;
```

Mover Pieza (Piezaseleccionada)

Se encarga de realizar otro pick para ver a que casilla quiere mover la pieza el usuario. Una vez tiene la casilla deseada, llama a movimientoPieza que será la función que se encarga realmente del movimiento. Dependiendo del éxito o no del movimiento, cambiará el estado a pieza_movida o a Jugando (entendiendo que quiere seleccionar otra pieza diferente), además de apagar las casillas encendidas (en el caso de que el modo amateur estuviese activado). Este es el código:

```
case piezaseleccionada:

        num_casilla = pick(x,y);
        int casilla_seleccionada = 0;

        if((num_casilla >= 1 && num_casilla <=36) || (num_casilla >= 40 && num_casilla <=103)){

                casilla_seleccionada = num_casilla;
                char tipo = piezas[seleccionada].tipo;
                int movida = 0;

                switch(tipo){

                        estado = jugando;

                        case 't':

                                movida = movimientoPieza(seleccionada, casilla_seleccionada,
num_casilla);
                                piezas[seleccionada].color = color_original;

                                if(movida == 1){
                                        estado = pieza_movida;
                                }

                                else{
                                        estado = jugando;
                                }
                                apagarEncendidas();
                                break;

                        case 'c':

                                movida = movimientoPieza(seleccionada, casilla_seleccionada,
num_casilla);
                                piezas[seleccionada].color = color_original;

                                if(movida == 1){
                                        estado = pieza_movida;
                                }
                                else{
                                        estado = jugando;
                                }

                                apagarEncendidas();

                        break;

                        …..... //Igual con resto de piezas
                }
        } break;
```

Cambiar Turno y cámara (Pieza_movida)

Una vez se haya movida la pieza deseada, estaremos en el estado pieza_movida. En este estado simplemente vamos a comprobar si la partida ha terminado, y si no llamamos a setCamara que se encarga de girar la cámara alrededor del tablero Y de realizar el cambio de turno de jugador. Este es el código correspondiente:

```
case pieza_movida:

        terminado = juegoTerminado();

        if(!terminado){

                estado = jugando;
                setCamara();

        }

        else{

                gluiOutput( "PARTIDA FINALIZADA");    // Muestra mensaje en interfaz

        }
break;
```

Vamos ahora a analizar las funciones más importantes.

Rellenar Posibles

Para ello vamos a comenzar con las de rellenaPosibles, que como hemos comentado previamente se encargan de ver a que casillas se podría mover la casilla seleccionada. Solo vamos a enseñar la de peón porque son funciones muy extensas y considero que lo importante es que se entienda la idea sin entrar demasiado en detalles.

– Rellenar posibles peón

Lo que vamos haciendo en esta función es lo siguiente: teniendo en cuenta los movimientos que un peón puede hacer, los cuales son: ir siempre hacía delante una posición o 2 si esta en su posición inicial, o ir uno hacia delante y otro hacia derecha o izquierda si hay alguna pieza del oponente que pueda comer, vamos mirando esas casillas en el tablero para comprobar si están disponibles. Por ejemplo para movernos una hacía delante sería como sumar/restar 8 en nuestro tablero. Para comer una pieza sería sumar/restar 7 o 9.

Estas casillas posibles son introducidas en un vector de posibles casillas, que nos será útil para determinar posteriormente el movimiento que el usuario quiere hacer o para iluminar las posibles casillas. También es importante algunas comprobaciones para no salirnos del tablero.

Este es el código:

```
void rellenarPosiblesPeon(int seleccionada){

		int t = 1;
		int sal = 0;
		int aux_casilla = piezas[seleccionada].sobre_casilla;
		piezas[seleccionada].num_posibles = 0;
		int pieza_aux = 0;

		if(turno_jugador == 0){ //NEGRAS

			if(aux_casilla + 7 < 104 && casillas[aux_casilla-40+7].estado == 1){

				pieza_aux = piezaSobreCasilla(aux_casilla+7);
				if((piezas[pieza_aux].color != piezas[seleccionada].color)&&(piezas[pieza_aux].sobre_casilla/8 != piezas[seleccionada].sobre_casilla/8)){
					piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] = aux_casilla+7;
					casillas[aux_casilla-40+7].color_antiguo = casillas[aux_casilla-40+7].color;
					casillas[aux_casilla-40+7].color = rojo;
					piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles + 1;
				}
			}

			if(aux_casilla + 9 < 104 && casillas[aux_casilla-40+9].estado == 1){

				pieza_aux = piezaSobreCasilla(aux_casilla+9);
				if((piezas[pieza_aux].color != piezas[seleccionada].color)&&(piezas[pieza_aux].sobre_casilla/8 != piezas[seleccionada].sobre_casilla/8)){
					piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] = aux_casilla+9;
					casillas[aux_casilla-40+9].color_antiguo = casillas[aux_casilla-40+9].color;
					casillas[aux_casilla-40+9].color = rojo;
					piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles + 1;
				}
			}

			if(aux_casilla + 8 < 104 && casillas[aux_casilla-40+8].estado == 0){

				piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] = aux_casilla+8;
				casillas[aux_casilla-40+8].color_antiguo = casillas[aux_casilla-40+8].color;
				casillas[aux_casilla-40+8].color = rojo;
				piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles + 1;

				if(aux_casilla <= 55 && aux_casilla >= 48 && casillas[aux_casilla-40+16].estado == 0){

					piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] = aux_casilla+16;
					casillas[aux_casilla-40+16].color_antiguo = casillas[aux_casilla-40+16].color;
					casillas[aux_casilla-40+16].color = rojo;
					piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles + 1;
				}
			}
		}

		else{

			if(aux_casilla -7 > 39 && casillas[aux_casilla-40-7].estado == 1){
```

```
                                pieza_aux = piezaSobreCasilla(aux_casilla-7);
                                if(piezas[pieza_aux].color != piezas[seleccionada].color){

                                        piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] =
aux_casilla-7;
                                        casillas[aux_casilla-40-7].color_antiguo = casillas[aux_casilla-40-7].color;
                                        casillas[aux_casilla-40-7].color = rojo;
                                        piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles +
1;

                                }
                        }

                        if(aux_casilla -9 > 39 && casillas[aux_casilla-40-9].estado == 1){

                                pieza_aux = piezaSobreCasilla(aux_casilla-9);

                                if(piezas[pieza_aux].color != piezas[seleccionada].color){

                                        piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] =
aux_casilla-9;
                                        casillas[aux_casilla-40-9].color_antiguo = casillas[aux_casilla-40-9].color;
                                        casillas[aux_casilla-40-9].color = rojo;
                                        piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles +
1;
                                }
                        }
                        if(aux_casilla - 8 > 39 && casillas[aux_casilla-40-8].estado == 0){

                                piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] = aux_casilla-8;
                                casillas[aux_casilla-40-8].color_antiguo = casillas[aux_casilla-40-8].color;
                                casillas[aux_casilla-40-8].color = rojo;
                                piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles + 1;

                                if(aux_casilla <= 95 && aux_casilla >= 88 && casillas[aux_casilla-56].estado ==
0){

                                        piezas[seleccionada].posibles[piezas[seleccionada].num_posibles] =
aux_casilla-16;
                                        casillas[aux_casilla-40-16].color_antiguo = casillas[aux_casilla-40-
16].color;
                                        casillas[aux_casilla-40-16].color = rojo;
                                        piezas[seleccionada].num_posibles = piezas[seleccionada].num_posibles +
1;
                                }
                        }
                }
}
```

Para el resto de funciones es algo más complicado pero la metodología es la misma. Por ejemplo para un alfil o torre, se hace lo mismo pero como se pueden mover a lo largo de todo el tablero, en vez de ir sumando poco a poco se realiza todo dentro de un bucle. Será importante el comprobar que no se encuentre con otras piezas en el camino, ya que si lo hiciese la pieza no podría continuar (hay que recordar que la única pieza que puede saltar es el caballo), por lo que el bucle debe acabar.

Movimiento pieza

La función movimiento pieza es llamada para hacer efectivo el movimiento de pieza deseado por el usuario. Es independiente del tipo de pieza.

El primer if que nos encontramos es simplemente para saber si el usuario ha hecho click en la casilla que se quiere comer o en una casilla hacía donde se quiere mover. El código en la parte if y else es el mismo, solo que en el del else no hace falta mirar si hay una pieza sobre la casilla ya que se sabe por lógica.

Esta función tiene 2 casos de uso diferentes: que queramos mover a una casilla libre o que queramos mover a una casilla donde exista una pieza.

Si queremos mover a una casilla libre, la metodología a seguir es más sencilla. Se comprueba si la casilla que le ha sido pasada está entre las posibles casillas donde puede moverse (información que viene de la función rellenarPosiblesTIPO). Si la casilla es posible, realizamos una serie de operaciones como cambiar el estado de la casilla en la que vamos a estar o en la que dejamos de estar, llamamos a la función que se encarga de realizar la animación del movimiento o poner el estado a “jugando”.

Si por el contrario la casilla tiene encima una pieza, habrá que hacer todo lo anterior más llamar a una función que se encarga de eliminar una pieza (con su correspondiente animación para ello).

```
int movimientoPieza(int seleccionada, int casilla_seleccionada, int num_casilla){

	int movida = 0;
	int i = 0;
	int sal = 0;
	int aux_pieza = 0;
	int pos_actual = piezas[seleccionada].sobre_casilla;

	if(num_casilla >= 40 && num_casilla <= 103){

		for(i; i < piezas[seleccionada].num_posibles && sal == 0; i++){

			if(num_casilla == piezas[seleccionada].posibles[i]){

				aux_pieza = piezaSobreCasilla(num_casilla);

				if(aux_pieza != 0){

					sal = 1;
					movida = 1;
					animacionMoverPieza(seleccionada, casilla_seleccionada);
					piezas[seleccionada].sobre_casilla = num_casilla;

					if((piezas[seleccionada].tipo == 'p') &&
((piezas[seleccionada].sobre_casilla <= 103 && piezas[seleccionada].sobre_casilla > 95) ||
(piezas[seleccionada].sobre_casilla < 48 && piezas[seleccionada].sobre_casilla >= 40))){

						piezas[seleccionada].tipo = 'q';

					}

					casillas[pos_actual-40].estado = 0;
					estado = jugando;
					eliminarPiezaTablero(aux_pieza);
```

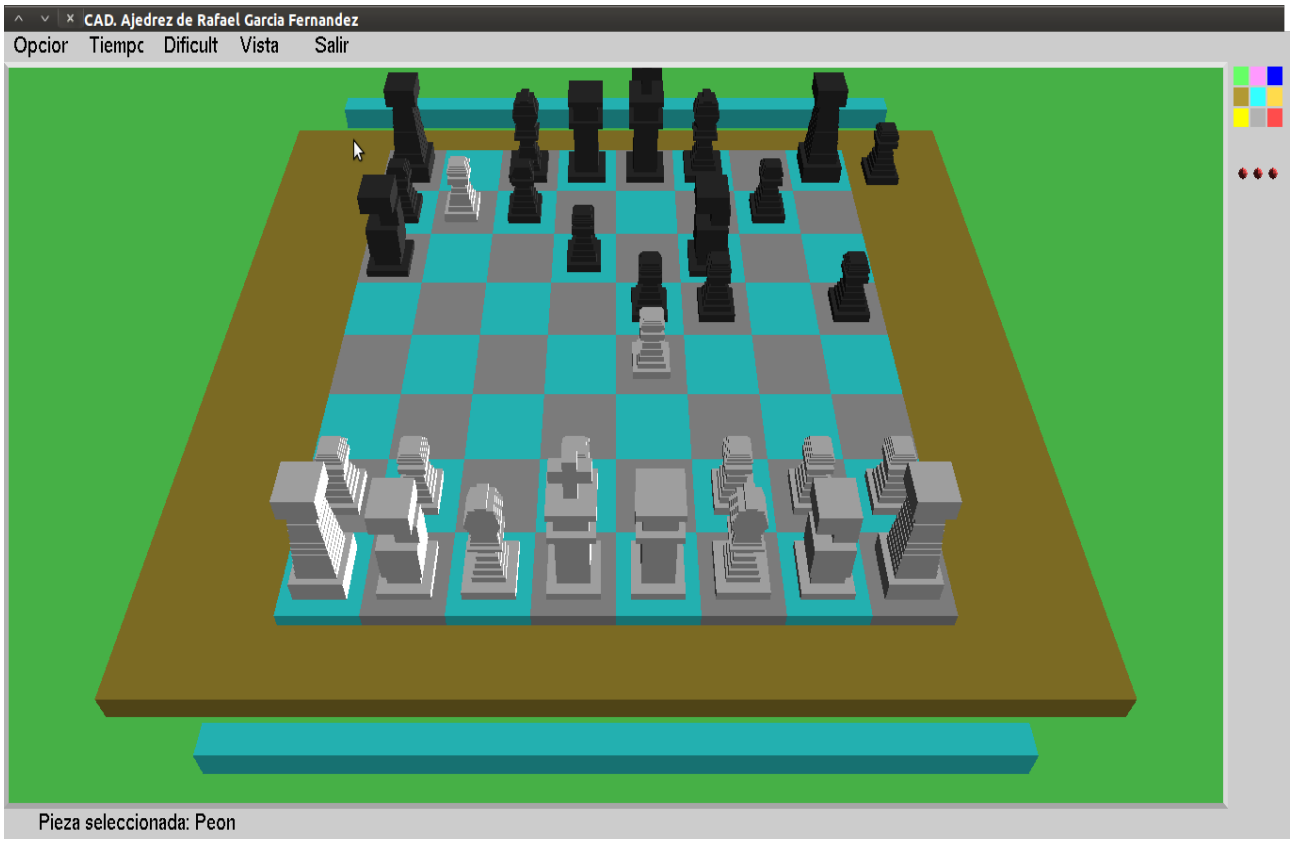
```
                                        piezas[aux_pieza].sobre_casilla = 0;
                                        piezas[aux_pieza].estado = 1;

                                }

                                else{

                                        sal = 1;
                                        movida = 1;
                                        animacionMoverPieza(seleccionada, casilla_seleccionada);
                                        piezas[seleccionada].sobre_casilla = num_casilla;

                                        if((piezas[seleccionada].tipo == 'p') &&
((piezas[seleccionada].sobre_casilla <= 103 && piezas[seleccionada].sobre_casilla > 95) ||
(piezas[seleccionada].sobre_casilla < 48 && piezas[seleccionada].sobre_casilla >= 40))){

                                                piezas[seleccionada].tipo = 'q';

                                        }


                                        casillas[casilla_seleccionada-40].estado = 1;
                                        casillas[pos_actual-40].estado = 0;
                                        estado = jugando;
                                }
                        }
                }
        }

        else{

                int nueva_num_casilla = piezas[num_casilla-1].sobre_casilla;

                for(i; i < piezas[seleccionada].num_posibles && sal == 0; i++){

                        if(nueva_num_casilla == piezas[seleccionada].posibles[i]){

                                aux_pieza = piezaSobreCasilla(nueva_num_casilla);

                                if(aux_pieza != 0){

                                        sal = 1;
                                        movida = 1;
                                        animacionMoverPieza(seleccionada, nueva_num_casilla);
                                        piezas[seleccionada].sobre_casilla = nueva_num_casilla;


                                        if((piezas[seleccionada].tipo == 'p') &&
((piezas[seleccionada].sobre_casilla <= 103 && piezas[seleccionada].sobre_casilla > 95) ||
(piezas[seleccionada].sobre_casilla < 48 && piezas[seleccionada].sobre_casilla >= 40))){

                                                piezas[seleccionada].tipo = 'q';
                                                Dibuja();
                                        }

                                        casillas[pos_actual-40].estado = 0;
                                        estado = jugando;
                                        eliminarPiezaTablero(aux_pieza);
                                        piezas[aux_pieza].sobre_casilla = 0;
                                        piezas[aux_pieza].estado = 1;

                                }
                        }
```

```
                }
        }

        return movida;

}
```

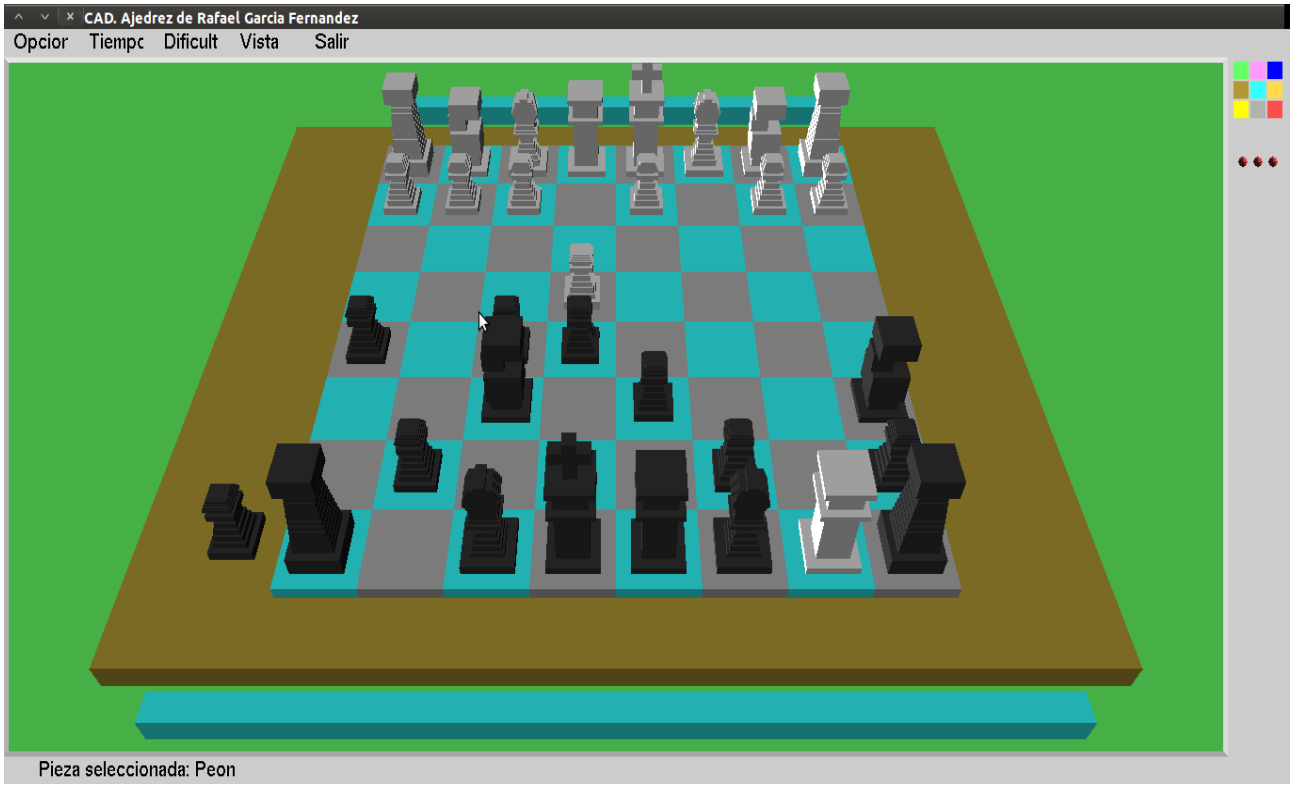
Cabe destacar el siguiente fragmento de código:

```
if((piezas[seleccionada].tipo == 'p') && ((piezas[seleccionada].sobre_casilla <= 103 &&
piezas[seleccionada].sobre_casilla > 95) || (piezas[seleccionada].sobre_casilla < 48 &&
piezas[seleccionada].sobre_casilla >= 40))){

                                                piezas[seleccionada].tipo = 'q';

                                        }
```

El cual nos vale para saber si un peón ha llegado hasta el fondo del oponente, lo cual significa en el ajedrez que se convierte en una reina. Aquí tenemos una captura donde podemos ver este hecho:



Captura 16. Peón antes de transformarse



Captura 17. Peón transformado en reina

Animación mover pieza

Esta función se encarga de mover la pieza de forma que se "deslice" por el tablero en vez de teletransportarse directamente. Para ello hay que hacer un switch, ya que cada una de las piezas se mueve de forma diferente. Voy a mostrar solo el código del peón ya que es una función muy extensa.

Lo que básicamente hace esta función es ir comprobando hacia donde en el tablero esta la casilla a la que hay que moverse. Si esta hacía recta hacía arriba (movimiento normal del peón por ejemplo), pues en un while va añadiendo 0.5 a la coordenada pospiezay (que hay que recordar que realmente es la del plano Z) y llamando a la función Dibuja() para actualizar lo que se ve. Así se irá acercando progresivamente a la casilla deseada hasta situarse en ella. Lo mismo sería en las otras direcciones: recto abajo para los peones del otro color, diagonal derecha y izquierda para comer...al igual que para el resto de piezas.

El caballo es el único que tiene una pequeña peculiaridad, y es que como ya se sabe el caballo puede saltar. Por ello, además de modificar las coordenadas del plano X y Z, también se modifica la Y al principio del movimiento (se levanta por decirlo de algun modo) y al final del movimiento (se deja caer sobre el tablero).

```
void animacionMoverPieza(int seleccionada, int casilla){

        char tipo = piezas[seleccionada].tipo;
        int pos_actual = piezas[seleccionada].sobre_casilla;

        switch(tipo){
```

```
case 'p':

        if((pos_actual < casilla) && ((pos_actual-40)%8 == (casilla-40)%8)){

                while(piezas[seleccionada].pospiezay < casillas[casilla-40].poscasillaz){

                        piezas[seleccionada].pospiezay = piezas[seleccionada].pospiezay + 0.5;
                        Dibuja();

                }

        }

        else if((pos_actual > casilla) && ((pos_actual-40)%8 == (casilla-40)%8)){

                while(piezas[seleccionada].pospiezay > casillas[casilla-40].poscasillaz){

                        piezas[seleccionada].pospiezay = piezas[seleccionada].pospiezay - 0.5;
                        Dibuja();

                }

        }

        else if((pos_actual < casilla) && ((pos_actual-40)%8 < (casilla-40)%8)){

                while((piezas[seleccionada].pospiezay < casillas[casilla-40].poscasillaz) &&
(piezas[seleccionada].pospiezax > casillas[casilla-40].poscasillax)){

                        piezas[seleccionada].pospiezay = piezas[seleccionada].pospiezay + 0.5;
                        piezas[seleccionada].pospiezax = piezas[seleccionada].pospiezax - 0.5;
                        Dibuja();

                }

        }

        else if((pos_actual > casilla) && ((pos_actual-40)%8 > (casilla-40)%8)){

                while((piezas[seleccionada].pospiezay > casillas[casilla-40].poscasillaz) &&
(piezas[seleccionada].pospiezax < casillas[casilla-40].poscasillax)){

                        piezas[seleccionada].pospiezay = piezas[seleccionada].pospiezay - 0.5;
                        piezas[seleccionada].pospiezax = piezas[seleccionada].pospiezax + 0.5;
                        Dibuja();

                }

        }

        else if((pos_actual > casilla) && ((pos_actual-40)%8 < (casilla-40)%8)){

                while((piezas[seleccionada].pospiezay > casillas[casilla-40].poscasillaz) &&
(piezas[seleccionada].pospiezax > casillas[casilla-40].poscasillax)){

                        piezas[seleccionada].pospiezay = piezas[seleccionada].pospiezay - 0.5;
                        piezas[seleccionada].pospiezax = piezas[seleccionada].pospiezax - 0.5;
                        Dibuja();

                }

        }
```

```
                    else if((pos_actual < casilla) && ((pos_actual-40)%8 > (casilla-40)%8)){

                        while((piezas[seleccionada].pospiezay < casillas[casilla-40].poscasillaz) &&
(piezas[seleccionada].pospiezax < casillas[casilla-40].poscasillax)){

                            piezas[seleccionada].pospiezay = piezas[seleccionada].pospiezay + 0.5;
                            piezas[seleccionada].pospiezax = piezas[seleccionada].pospiezax + 0.5;
                            Dibuja();

                        }

                    }

                    break;
```

La función eliminarPieza hace algo bastante parecido, solo que en este caso cuenta con coordinadas que le dicen donde exactamente tiene que colocar la pieza “comida” (dentro de su pasillo).

SetCamara (Y cambio de turno)

SetCamara se encarga tanto del movimiento de la cámara para simular el colocarse detrás del tablero como para cambiar el turno de jugador.

El movimiento de cámara lo hace de forma muy parecida a la animación de las piezas. Haciendo uso de un while va acercando progresivamente a las coordinadas predeterminadas para cada uno de los turnos. Después de cada iteración llamar a Dibuja() para actualizar lo que vemos. El cambio de turno se hace simplemente haciendo uso de un if y cambiando el valor actual que tenga la variable.

```
void setCamara(){

    if(turno_jugador == 1){

        while(z_camara > -61){

            z_camara = z_camara - 3;
            Dibuja();
        }

    while(view_roty > -180){

            view_roty = view_roty - 5;
            Dibuja();
        }
        turno_jugador = 0;
    }

    else{

        while(z_camara < 61){

            z_camara = z_camara + 3;
            Dibuja();
        }

        while(view_roty < 0){
```

```
                    view_roty=view_roty+5;
                    Dibuja();
                }
                turno_jugador = 1;
            }
}
```