MEMORIA DEL PROYECTO

Asignatura

Informática Industrial y Comunicaciones

JUEGO DEL AJEDREZ

Los transistores locos:

Felipe Arriero Serrano

Alvaro Carrasco Alvarez

Jose Conejero Gonzalez

Gonzalo Morcillo Garcia

César Zabala Martín

ÍNDICE

1. ESTRUCTURA DEL CÓDIGO

Para el desarrollo de nuestro juego, hemos empleado las siguientes clases:

- Modo Juego: Es el encargado de gestionar cada uno de los modos de juego que tenemos.

Modo Juego
enum class TipoPieza { PEON, TORRE, CABALLO, ALFIL, REINA, REY }; enum class ModoJuego { NORMAL, ALAMOS, SILVERMAN };
struct ModoJuegoConfig

- Menú: Contiene los botones para seleccionar modos de juego, configuración y demás.

Menú
enum class Estado_Menu Estado_Menu Estado_Actual_Menu; void execute_Accion_Boton(int id);

void set_Menu();
void set_Estado_Menu(Estado_Menu estado);
void set_Fondo(const std::string& ruta);
const std::string& get_Fondo() const;
void draw_Menu();
void draw_Fondo(const std::string& ruta);
void keyboard_Menu(unsigned char key);
void mouse_Menu(int mouse_X, int mouse_Y);
void actualizar_Hover(int mouse_X, int mouse_Y);
Estado_Menu get_Estado_Menu() const { return Estado_Actual_Menu; }

- Botón: En esta clase se define el botón que se usara próximamente en el menú.

Botón
Color Color_Boton; int X_Boton; int Y_Boton; int Ancho_Boton; int Altura_Boton; friend class Menu;

Boton(int id, std::string texto, Color color, int x, int y, int ancho = 400, int altura = 50);
void draw_Boton();
bool contact_Boton(int mouse_X, int mouse_Y) const;
void sound_Boton(int mouse_X, int mouse_Y) const;
void set_ColorRGB(int r, int g, int b); // Metodo para Cambiar el Color
bool hovered = false;
int ID_Boton;
std::string Texto_Boton;

- Variables globales: Sirve para gestionar la ventana en la que aparecerá el juego.

Variables globales

extern int window_Width;
extern int window_Height;
const float virtual_Width = 1920.0f;
const float virtual_Height = 1080.0f;
extern int viewport_X;
extern int viewport_Y;
extern int viewport_Width;
extern int viewport_Height;

- Tablero: Contiene todas las piezas y controla los movimientos de estas de una casilla a otra.

Tablero

ModoJuego modo;
ModoJuegoConfig config;
float tamCasilla;
Vector2D origenTablero;
bool esperandoPromocion = false;
bool gameOver = false;
Vector2D posPromocion;
bool colorPromocion;
std::string mensajeFinal;
std::vector<Pieza*> piezas;
void inicializarPiezas();
bool turnoBlancas = true;
bool hayJaqueBlancas = false;
bool hayJaqueNegras = false;
void toggleCapturaAliados();

Vector2D casillaAPosicion(int col, int fila) const;
Pieza* piezaSeleccionada = nullptr;
Vector2D casillaSeleccionada;
Tablero(ModoJuego modoSeleccionado);
Tablero clonar() const;
void dibujarTablero();
void dibujarPiezas() const;
void dibujarJaque();
void manejarClick(float mouseX, float mouseY);
void promocionarPeon(char opcion);
int getFilas() const { return config.filas; }
int getColumnas() const { return config.columnas; }
bool puedeComerseAliados() const { return config.puedeComerseAliados; }
const std::vector<TipoPieza>& getPiezasPermitidas() const { return config.piezasPermitidas; }
Vector2D getOrigen() const { return origenTablero; }
float getTamCasilla() const { return tamCasilla; }
bool estaLibre(int col, int fila) const;
bool hayPiezaContraria(int col, int fila, bool blanca) const;
bool puedeMoverA(int col, int fila, bool esBlanca) const;
friend void OnKeyboard(unsigned char key, int x, int y);
bool Jaque(bool esBlancas);
void simularMovimiento(Pieza* pieza, Vector2D destino);
bool JaqueMate(bool esBlancas);
~Tablero();
bool estaTerminado() const { return gameOver; }
const std::string& getMensajeFinal() const { return mensajeFinal; }

- Funciones globales: Sirve para dibujar un texto y calcular su ancho.

Funciones Globales
void draw_BitmapText(const std::string &text, float x, float y); int calculate_Ancho_Texto(const std::string& texto, void *fuente = GLUT_BITMAP_HELVETICA_18);

- Gestor Audio: Es la clase que controlará el audio del juego.

Gestor Audio

static std::vector<Cancion> v_playlist_Musica;
static int current_Track;
static bool playing_Musica;
static bool paused_Track;
static time_t track_Start_Time;
static bool usuario_Ha_Cambiado_Pista;
static bool reproducirAleatoria_Pendiente;
static void play_Music(const std::string& track, bool repeat = true);
static void random_Track();

static void set_Gestor_Audio();
static void set_Volumen(int volumen);
static void update_Musica(Estado_Menu estado);
static void next_Track();
static void previous_Track();
static void stop_Musica();
static void pause_Musica();
static void play_Title_Screen_Music();
static int get_current_Track();
static std::string get_current_Track_Name();

- Vector2d: Representa puntos 2D, posiciones o desplazamientos en el tablero.

Vector2d

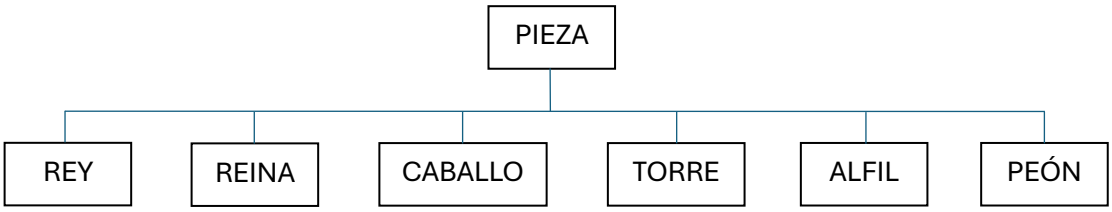
double x{};
double y{};
Vector2D() = default;
Vector2D(double x_, double y_) : x(x_), y(y_) {}
double modulo() const; //modulo del vector
double argumento() const; //argumento del vector
Vector2D unitario() const; //devuelve un vector unitario
Vector2D operator - (const Vector2D&) const; //resta de vectores
Vector2D operator + (const Vector2D&) const; //suma de vectores
double operator *(const Vector2D&) const; // producto escalar
Vector2D operator *(double) const; // producto por un escalar
Vector2D operator /(double escalar) const;
bool casiIgual(const Vector2D& otro, float epsilon = 0.01f) const;

- Pieza: Definirá la pieza (tamaño, color, tipo, posición…).

Pieza
Vector2D posicion; Vector2D velocidad; Vector2D destino; bool enMovimiento = false; bool esBlanca; TipoPieza tipo;

Pieza(Vector2D pos, bool blanca, TipoPieza tipoPieza);
virtual ~Pieza() {}
virtual void dibujar() = 0;
virtual std::vector<Vector2D> calcularMovimientosValidos(const Tablero& tablero) const = 0;
virtual Pieza* clonar() const = 0;
bool esPiezaBlanca() const { return esBlanca; }
void setPosicion(Vector2D nuevaPos) { posicion = nuevaPos; }
Vector2D getPosicion() const { return posicion; }
bool esAliada(bool blancas) const { return esBlanca == blancas; }
TipoPieza getTipo() const { return tipo; }
bool estaEnMovimiento() const { return enMovimiento; }

- Clases de las piezas: Cada pieza tiene su clase por separado, debido a que la forma de moverse es diferente para cada una de ellas.



En el caso del peón, como solo puede mover doble cuando se encuentra en la primera fila, hemos implementado una función para que solo mueva doble en ese caso.

Luego las demás se mueven según las reglas normales del juego.

Además, el rey tiene restricciones de movimiento en caso de jaque, el juego nos avisará cuando tengamos uno de los dos reyes en jaque.

2. MENU DEL JUEGO

Gracias a la clase botón, podremos seleccionar los modos de juego o los ajustes. Los modos de juego en cuestión son los siguientes:

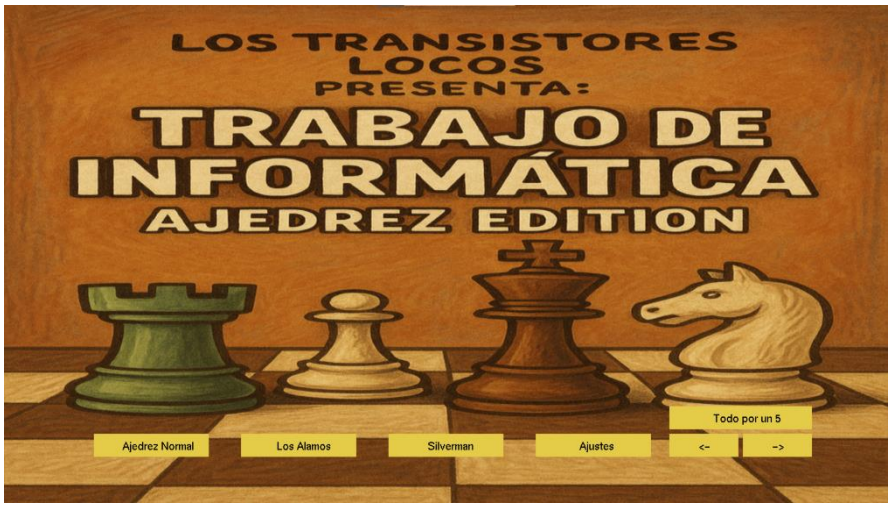
-Ajedrez normal: Ajedrez clásico de 8x8 con la distribución de piezas que se ha usado desde la creación del juego.

-Los Álamos: Variante del ajedrez que se juega en un tablero 6x6 y en la que se elimina el alfil. La variante data del año 1956.

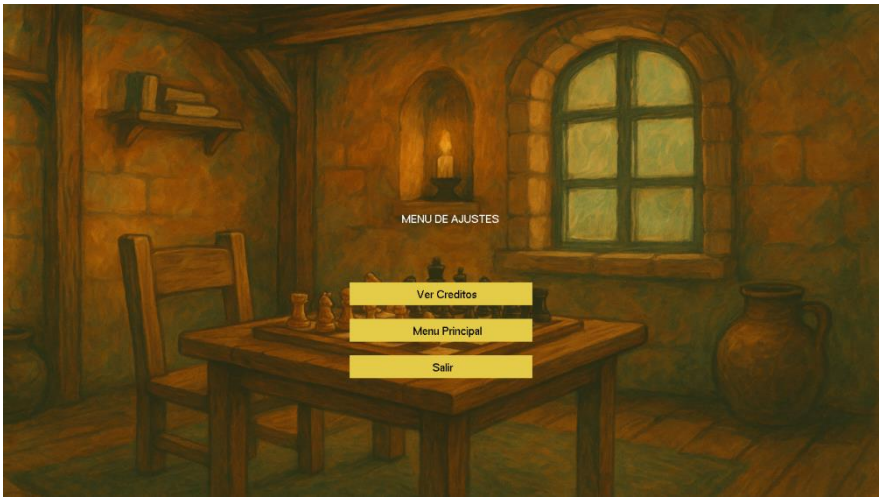
-Silverman: Variante que busca ser más rápida y sencilla que el ajedrez tradicional, donde se jugará en un 4x4 y solo se usarán peones, torres, alfiles, caballo, reina y rey.

Luego los ajustes son muy básicos, se muestran los créditos, volver al menú principal o salir del juego.

En nuestro juego la clase que controla el funcionamiento de los estados del menú se llama TrabajoInformática, la cual establece el estado en función del botón que pulsemos.



Una vez elegido el modo de juego, se pasa al juego, donde comienza la partida. Al terminar la partida si pulsamos Escape nos saldrá el menú de ajustes que nos permitirá cerrar el juego o volver al menú principal. Cuando pulsamos Escape en el juego sale lo siguiente:

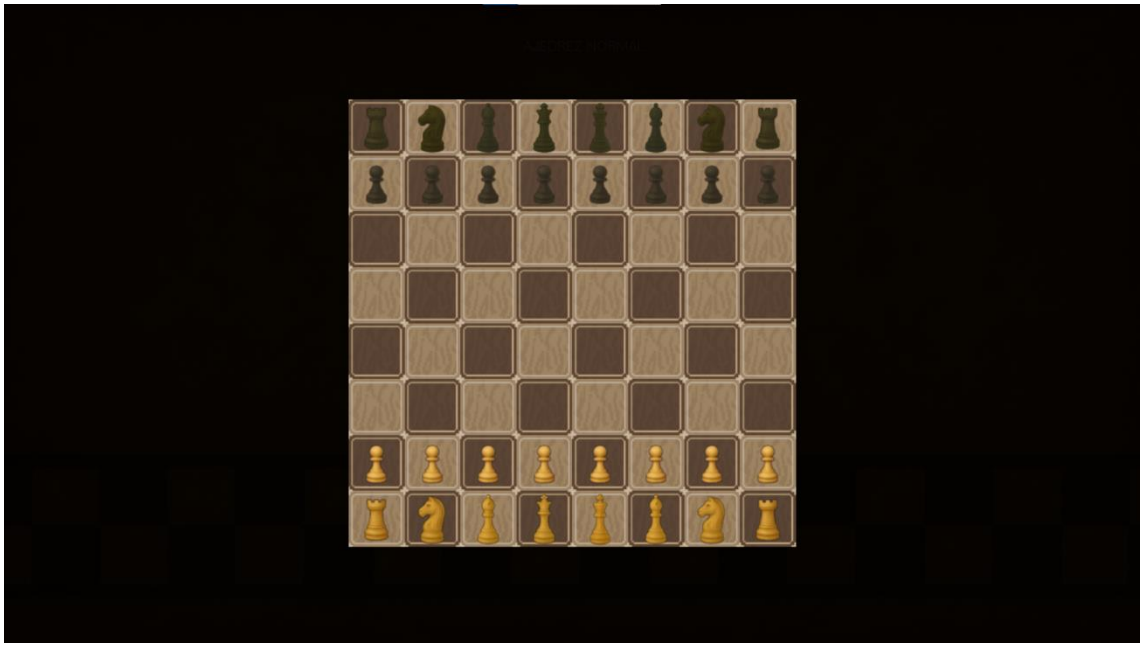


En nuestro código, cada pieza calcula el movimiento válido en su propia clase, luego esta información pasa a la clase tablero y esta ajustará y definirá si son movimientos válidos, si hay Jaque o si hay Jaque Mate.

3. GRÁFICOS

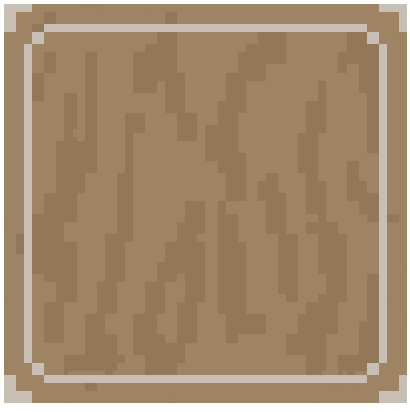
A continuación, vamos a explicar como se han desarrollado los gráficos del juego. Como hemos podido ver en el primer apartado, el juego será en 2 dimensiones. Hemos hecho uso de dos librerías clave, “freeglut.h” y “ETSIDI.h” que hemos empleado en las prácticas de laboratorio también.

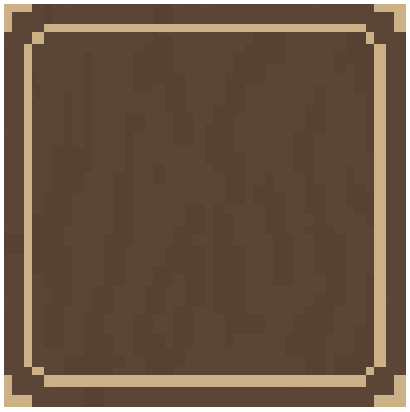
Para pintar el tablero, usaremos imágenes guardadas en formato .png en la carpeta bin/imágenes del proyecto. El aspecto del tablero que podemos ver en pantalla es el siguiente:



Para obtener este resultado, hemos empleado diferentes funciones:

-Dentro de tablero, tenemos la función dibujar tablero, esta función dibujará el tablero casilla a casilla. Toma la imagen de la casilla blanca de la carpeta mencionada anteriormente. La casilla negra ídem. Luego las alterna haciendo que quede el tablero clásico. Cada casilla tiene el siguiente aspecto:





-Las piezas, se pintan cada una dentro de su clase y luego dentro del tablero se implementan. Además, se ajusta el tamaño dependiendo del modo de juego que hemos elegido. Ejemplo de algunas piezas:

-Dentro de la clase tablero se gestionan las piezas comidas, haciendo que estas desaparezcan de la imagen y la promoción de peones, es decir, cuando llega al final un peón, si pulsas la R, este se convertirá en una reina y si pulsas la T se convertirá en una torre.

-Además, la clase tablero gestionará una variante adicional del ajedrez que hemos incluido en este trabajo llamada Kramnik en la que un jugador puede comerse sus propias piezas pulsando la letra C una vez iniciado cualquier modo de juego.

-En cuanto al sistema de comer piezas, el método que hemos elegido es mediante una función de copia que hace que el tablero se vaya actualizando todo el rato.

Dentro de los gráficos hemos integrado una función de dibujar Jaque para que distinga el color que está en jaque y aparezca en pantalla la imagen correspondiente al jaque que haya. También hemos establecido una imagen para el Jaque Mate.

4. INICIO Y REINICIO DE PARTIDAS

Antes de comenzar a explicar el control del flujo de la partida, vamos a concretar los detalles de la creación de piezas. La configuración inicial se engloba en la función inicializarPiezas() dentro de la clase Tablero, que se autoinvoca desde el constructor.

Cada pieza (peones, torres, caballos, alfiles, reinas y reyes) es creada con su color a partir de las imágenes y con su posición inicial, adaptándose a la configuración del modo de juego que hayamos seleccionado. Luego se mandan al contenedor de piezas mediante push back. Esto nos permite eliminar fácilmente piezas y su eliminación del vector, también podemos introducir nuevas piezas de forma dinámica en la promoción de peones.

Cuando queremos comenzar una partida nueva, borramos las piezas con el destructor dentro de la clase Tablero, esta libera la memoria de cada pieza mediante un bucle for y se vuelve a ejecutar inicializar piezas ().

5. AJEDREZ NORMAL

5.1. SECUENCIA DE OPERACIONES

En este apartado vamos a explicar como se controlan los movimientos dentro del programa. El programa no ayuda al jugador a mover, simplemente deja que el jugador decida los movimientos, por lo que este deberá saber previamente cómo se mueve cada una de las piezas. Además, el programa comprobará si los movimientos que quiere el jugador realizar son válidos o no.

Para ello seguirá la siguiente secuencia:

1. En cada nuevo turno, el programa comprobará la existencia de jaque mate (rey amenazado en su posición actual y en posiciones futuras). En caso de cumplirse esto, la partida terminara y no se ejecutarán los puntos siguientes. En caso contrario avanzará.

En algunas situaciones como el jaque mate, necesitamos realizar el movimiento, comprobar y volver a la situación original. De todos modos, esto no se puede realizar directamente ya que, si en la casilla de destino hay una pieza y la borramos, no podemos volver a crearla igual. Para solucionar esto, hemos creado la función que copia el tablero y se ejecutará en esta función el movimiento sin necesidad de modificar el tablero.

2. Espera a que el jugador escoja la casilla de origen y con esta información comprueba si se encuentra ocupada por una pieza de su color. Si no encuentra pieza de su color, le obliga a volver a elegir. Si la posición esta ocupada por una pieza de su color pasa al siguiente paso.

3. Espera a que el jugador elija casilla de destino y después hay varias opciones:

a. Si está ocupada por él y no está pulsada la opción de comer piezas, considera que el jugador ha cambiado de pieza de origen y volverá a empezar el punto 3.

b. Comprueba si el movimiento es válido, si lo es, avanza al punto siguiente y si no lo es vuelve repite este punto.

c. Si la pieza esta ocupada por su mismo color y tenemos pulsada la opción de comer sus propias piezas, comprobará si el movimiento es válido y si lo es avanzará al siguiente punto y si no lo es volverá al 3.

4. Una vez ha sido verificado, ejecuta el movimiento donde se mueve la pieza a la casilla de destino y comiendo la que se encontraba allí. Después de esto cambia el turno.

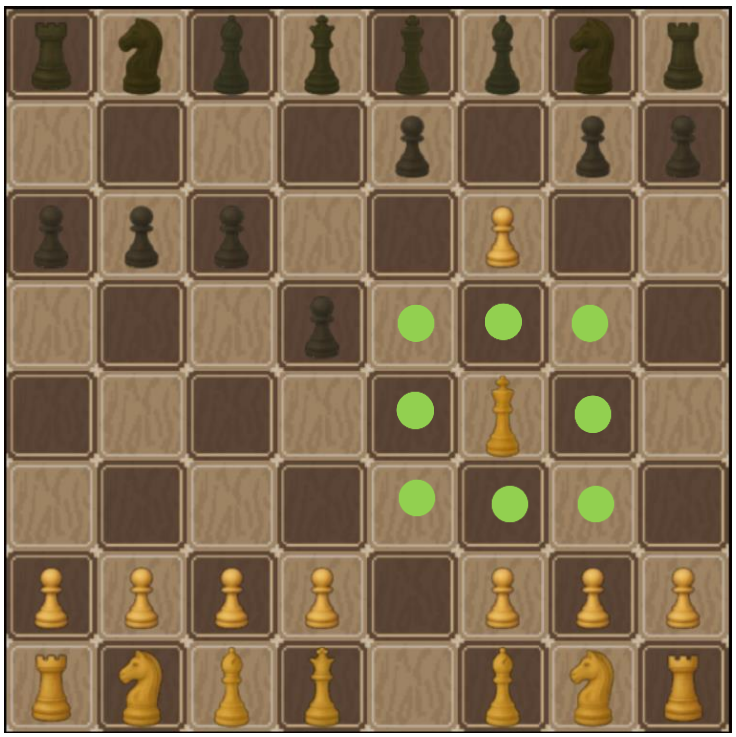
5.2. COMPROBACIÓN DE MOVIMIENTO

El punto más importante de los mencionados anteriormente, comprobar la validez del movimiento. Para ello dentro de cada pieza tenemos la función CalcularMovimientosValidos y cada uno funciona de forma diferente.

COMPROBACIÓN DEL MOVIMENTO DEL REY

Para el rey es un poco diferente, ya que es la pieza más importante de la partida. Sus pasos serán los siguientes:

1. Comprobará el movimiento correspondiente con el rey (solo podrá moverse en las casillas adyacentes). Para ello hace uso de la función mencionada anteriormente dentro de la clase Rey. El rey podrá moverse a las casillas marcadas.



2. Comprueba que la posición de destino no esta ocupada por una pieza de su mismo color (si no hemos pulsado la función de autocomerse piezas). Si estuviera la tecla pulsada, podríamos comernos una pieza nuestra.

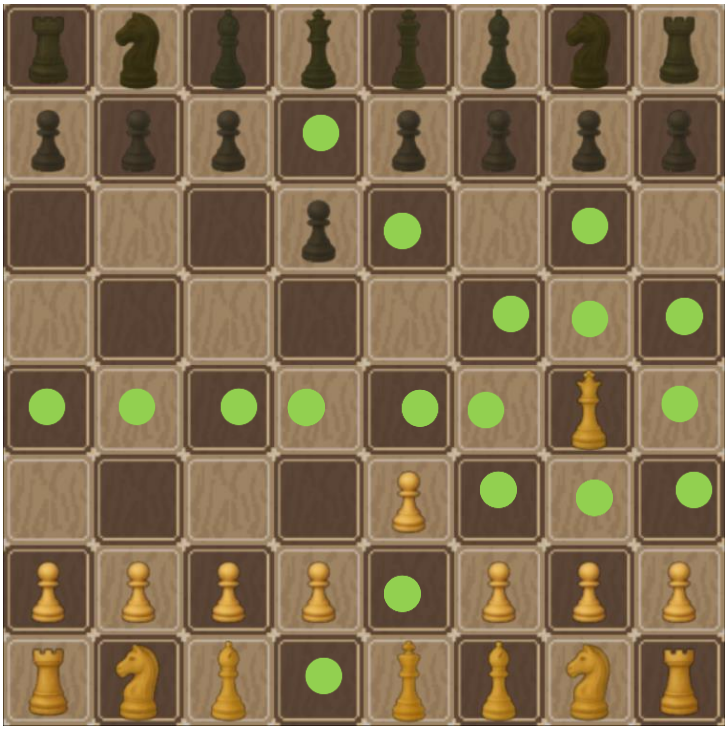
3. Comprobará también que la posición de destino no es una casilla amenazada por una pieza del jugador contrario. Para ello utilizaremos la función de Jaque dentro de la clase Tablero. Por ejemplo, en el siguiente caso, el rey no puede moverse a la casilla porque está amenazada por él. A la roja no se puede mover y a las verdes sí.

Si todo es válido, se ejecuta el movimiento.

COMPROBACIÓN DEL MOVIMENTO DE LA REINA

Para la reina simplemente comprobaremos los movimientos válidos. Sus pasos serán los siguientes:

1. Comprobará el movimiento correspondiente con la reina (Puede moverse en diagonal, horizontal y vertical). Para ello hace uso de la función CalcularMovimientosValidos dentro de la clase Reina. Los movimientos de la reina son los siguientes:



2. Comprueba que la posición de destino no está ocupada por una pieza de su mismo color (si no hemos pulsado la función de autocomerse piezas). Si estuviera la tecla pulsada, podríamos comernos una pieza nuestra.

3. Comprobará también que en la trayectoria no se encuentra ninguna pieza, ya sea de su color o del otro. Como podemos ver en la imagen, la reina no podría comerse al alfil al encontrarse en su trayectoria un peón.

En caso de que todo sea válido, se ejecutará el movimiento.

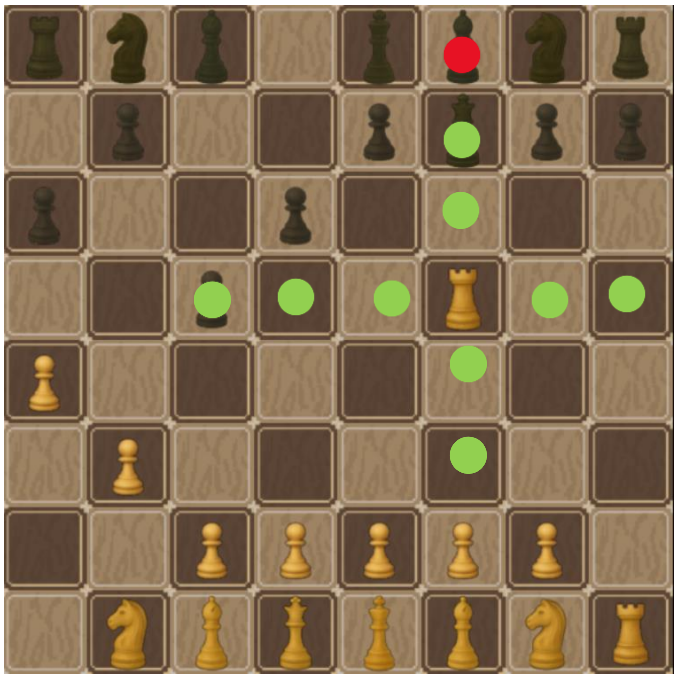
COMPROBACIÓN DEL MOVIMENTO DE LA TORRE

Para la torre comprobaremos los movimientos válidos. Sus pasos serán los siguientes:

1. Comprobará el movimiento correspondiente con la torre (Puede moverse en horizontal y vertical). Para ello hace uso de la función CalcularMovimientosValidos dentro de la clase Torre.

2. Comprueba que la posición de destino no está ocupada por una pieza de su mismo color (si no hemos pulsado la función de autocomerse piezas). Si estuviera la tecla pulsada, podríamos comernos una pieza nuestra.

3. Comprobará también que en la trayectoria no se encuentra ninguna pieza, ya sea de su color o del otro. Como podemos ver en la imagen, la torre no podría comerse al alfil al encontrarse en su trayectoria la reina.

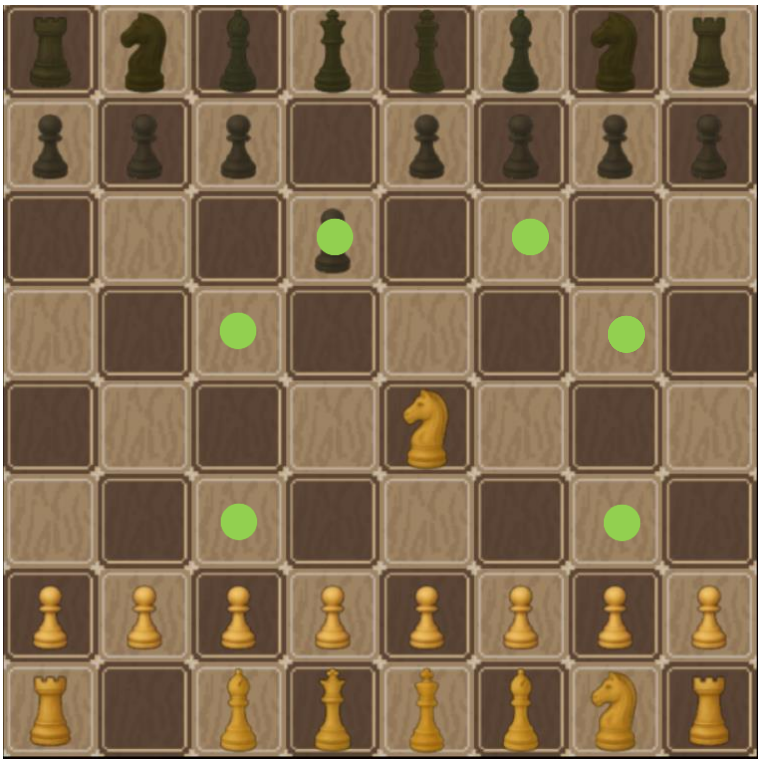


En caso de que todo sea válido, se ejecutará el movimiento.

COMPROBACIÓN DEL MOVIMENTO DEL CABALLO

Para el caballo comprobaremos los movimientos válidos. Sus pasos serán los siguientes:

1. Comprobará el movimiento correspondiente con el caballo (realiza movimientos en forma de L, desplazándose dos casillas en una dirección y una casilla en otra). Para ello hace uso de la función CalcularMovimientosValidos dentro de la clase Caballo.



2. Comprueba que la posición de destino no está ocupada por una pieza de su mismo color (si no hemos pulsado la función de autocomerse piezas). Si estuviera la tecla pulsada, podríamos comernos una pieza nuestra.

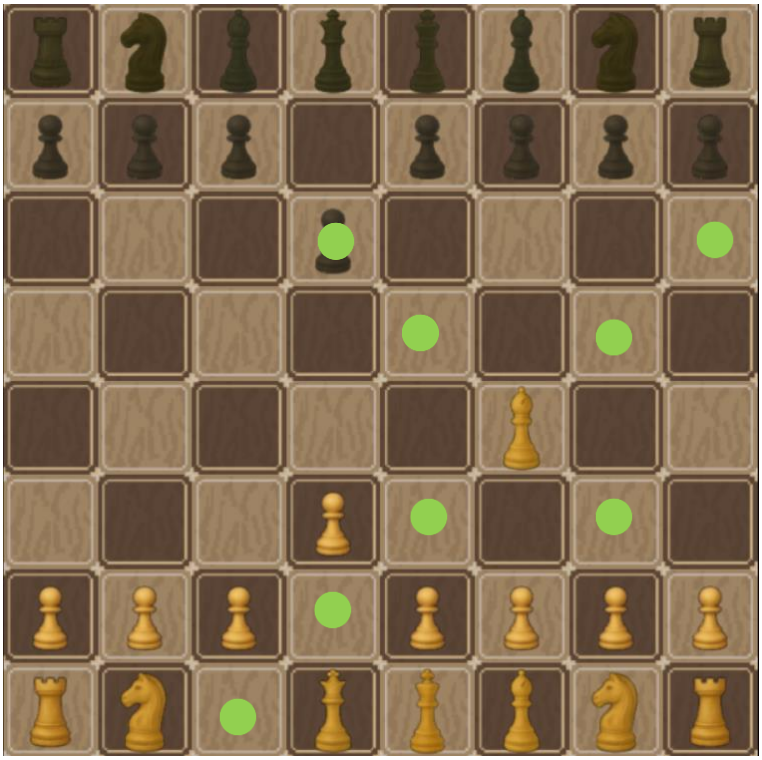
En este caso, no hace falta comprobar que no exista pieza en medio de la trayectoria, ya que el caballo puede saltar al resto de piezas, sean o no de su mismo color

En caso de que todo sea válido, se ejecutará el movimiento.

COMPROBACIÓN DEL MOVIMENTO DEL ALFIL

Para el alfil comprobaremos los movimientos válidos. Sus pasos serán los siguientes:

1. Comprobará el movimiento correspondiente con el alfil (Puede moverse en diagonal). Para ello hace uso de la función CalcularMovimientosValidos dentro de la clase Alfil.



2. Comprueba que la posición de destino no está ocupada por una pieza de su mismo color (si no hemos pulsado la función de autocomerse piezas). Si estuviera la tecla pulsada, podríamos comernos una pieza nuestra.

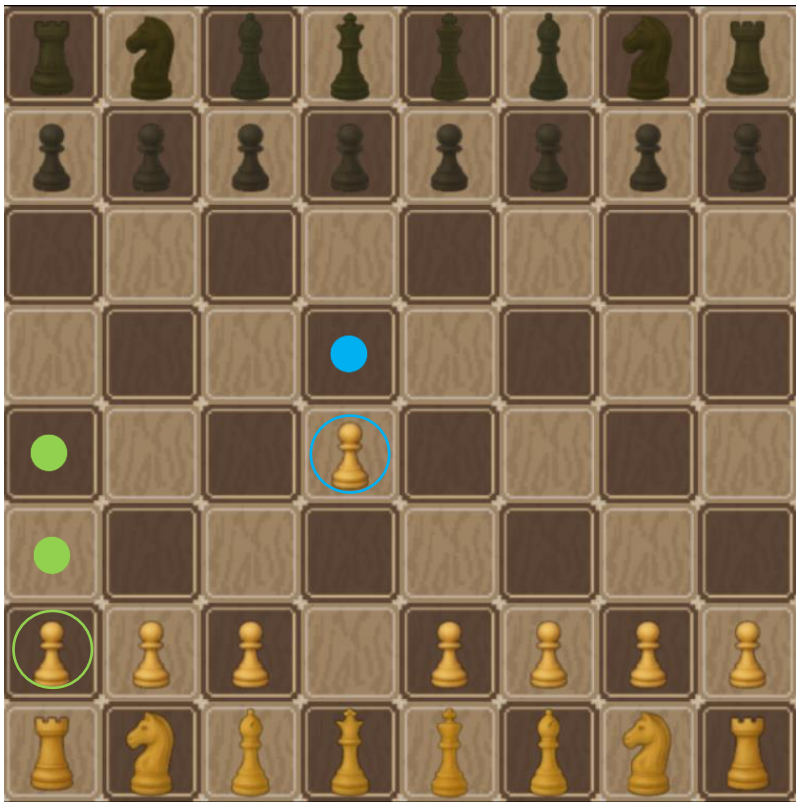
3. Comprobará también que en la trayectoria no se encuentra ninguna pieza, ya sea de su color o del otro. Como podemos ver en la imagen, el alfil no podría comerse al caballo al encontrarse en su trayectoria un peón.

En caso de que todo sea válido, se ejecutará el movimiento.

COMPROBACIÓN DEL MOVIMENTO DEL PEÓN

El movimiento del peón, a pesar de ser considerada la pieza más sencilla, puede tener algunas complicaciones en la ejecución. Si bien su movimiento normal es hacia delante, come en diagonal.

1. Comprobará el movimiento correspondiente con el peón (Puede moverse dos casillas en su fila inicial y una casilla en cualquiera de las otras filas). Para ello hace uso de la función CalcularMovimientosValidos dentro de la clase Peón.

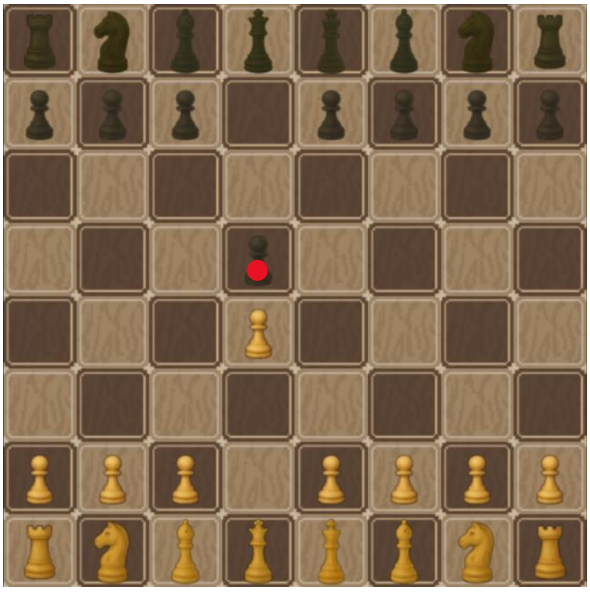


Verde – Casilla inicial, puede mover dos.

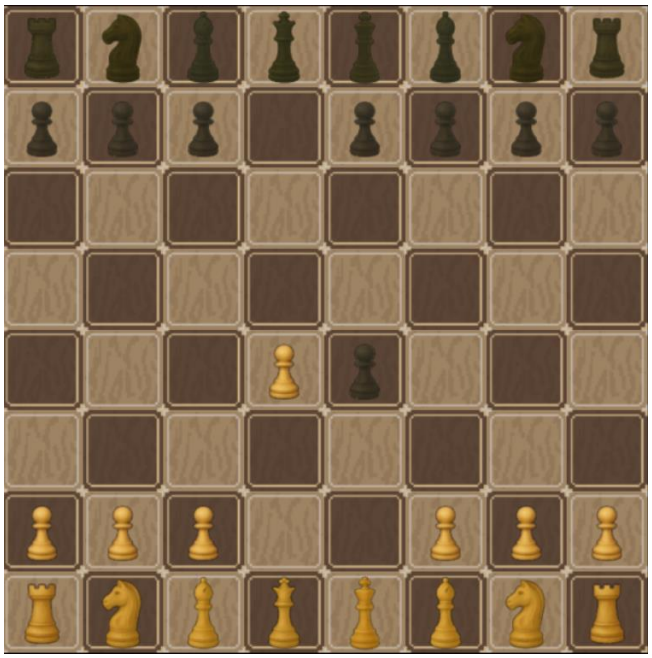
Azul – No casilla inicial, puede mover una.

2. Comprueba que la posición de destino no está ocupada por una pieza de su mismo color (si no hemos pulsado la función de autocomerse piezas). Si estuviera la tecla pulsada, podríamos comernos una pieza nuestra.

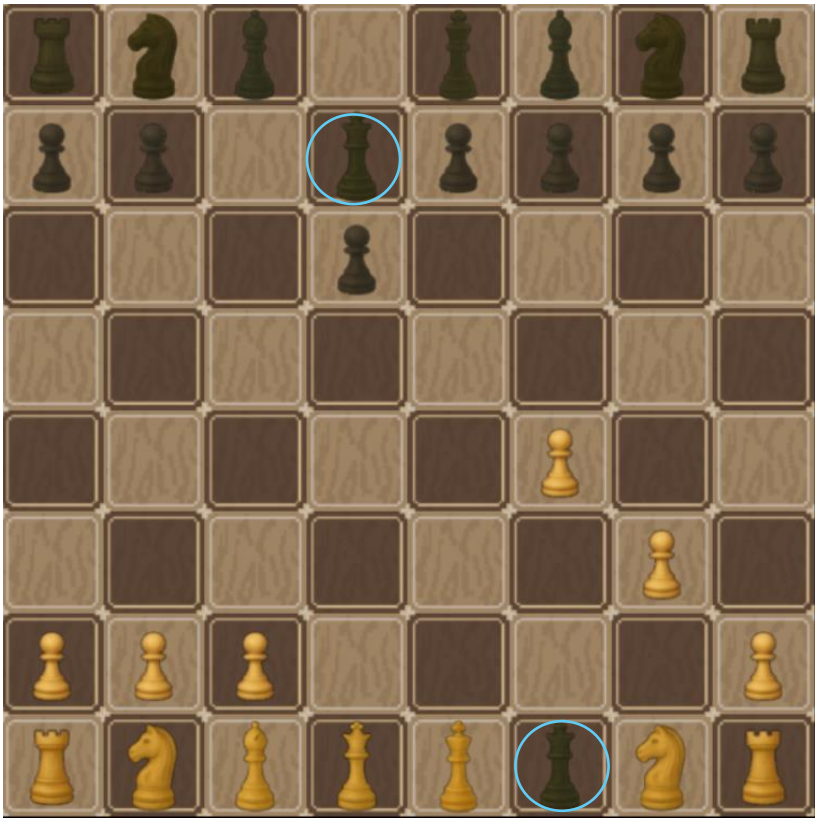
3. Comprobará también que en el destino no se encuentra ninguna pieza (moviendo en vertical), ya sea de su color o del otro. Como podemos ver en la imagen, el peón no podría avanzar al encontrarse otro peón enfrente.



4. En caso de mover en diagonal, comprobará que la posición de destino este ocupada por otra pieza, si es así puede comer. (Dependiendo del modo seleccionado podrá comer solo del otro color o de ambos).



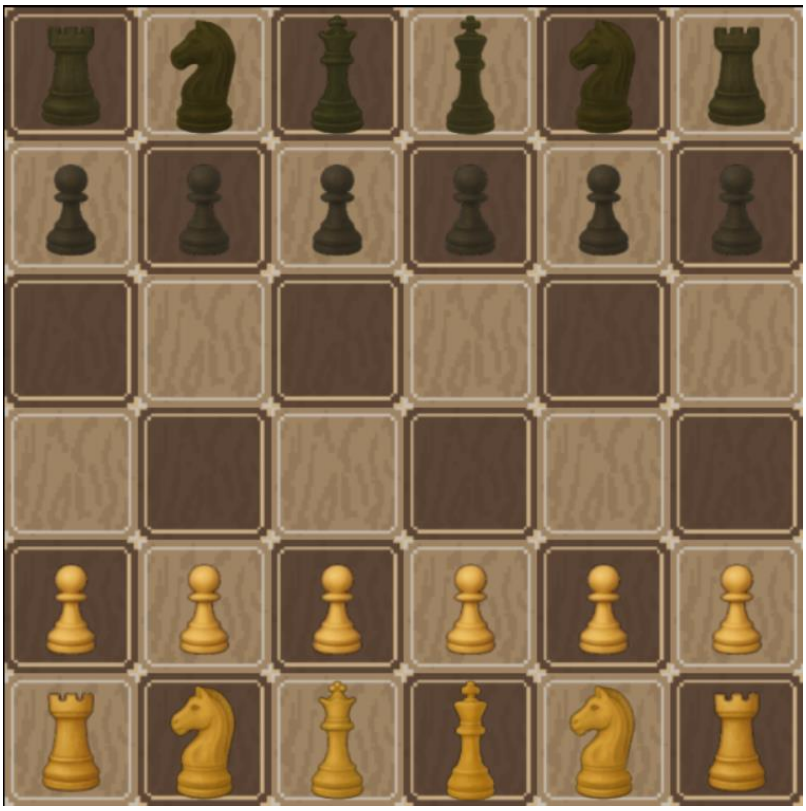
5. Cuando un peón llega al final del tablero, cambiará a una torre o una reina dependiendo de la tecla que pulse el jugador.



Aquí podemos ver 2 reinas, ya que un peón ha llegado al final.

6. AJEDREZ LOS ÁLAMOS

Todo funcionará exactamente igual que en el punto 5, con la única diferencia que el tablero es distinto. La configuración inicial del tablero sería la siguiente:



7. AJEDREZ SILVERMAN

Todo funcionará exactamente igual que en el punto 5, con la única diferencia que el tablero es distinto. La configuración inicial del tablero sería la siguiente:

8. REGLAS ESPECIALES

La única regla especial que hemos implementado es la variante Kramnik donde un jugador podría comerse sus propias piezas en caso de que este lo decida. Ya sea por estar atascado en una posición y comer una propia pieza te saca de esta o simplemente porque el jugador lo elige para obtener una ventaja. Esta variante se ejecuta pulsando la letra C.