

Taller de Computación 3 Trabajo Práctico Final

Dadas las siguientes clases, deberá crear un programa que simule un juego de ajedrez (simplificado) con 2 torres, el rey en un principio y luego agregar 2 peones.

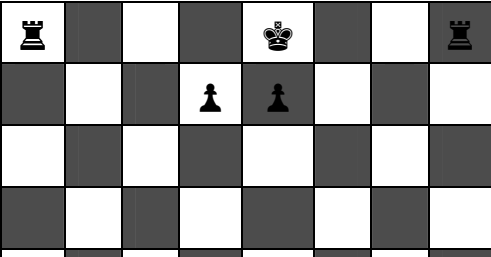
En él existen piezas de dos colores (blancas y negras) y en principio 2 tipos de piezas: rey, torre y luego de tipo peón (se podrán agregar reina, alfil y caballo como opción extra).

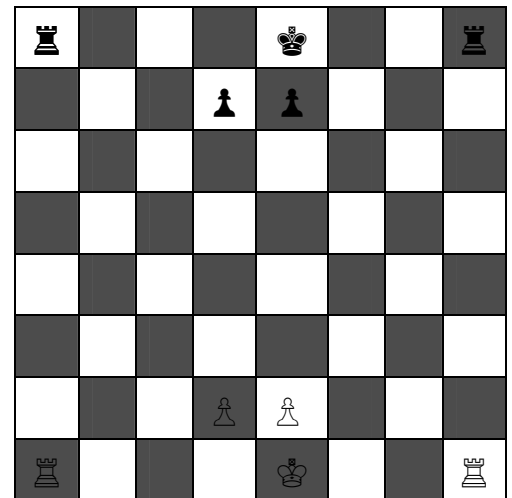
Existe un tablero compuesto por 64 celdas (8 x 8), donde las piezas se distribuirán como indica el cuadro.

El programa comienza con el método **comenzar**, luego ejecuta el método **dibujar**, el método **moverPieza**. Y por ultimo el método **Terminar**. Se debe agregar la posibilidad de grabar el juego para poder retomarlo en otro momento.

<div><div>Pieza</div><div><div>- int _posX; - int _posY - char _color - char _icono -bool puedeSaltar;</div><div><div>+ Pieza(int posX, int posY) + virtual ~Pieza() + void setPosX(int x) + void setPosY(int y) + int getPosX() + int getPosY() + void setIcono(char icono) + char getIcono() + virtual char tipoPieza() =0 + virtual void Dibujar()= 0 + virtual void Mover()= 0 + virtual bool movValido(int destX, int destY)=0</div></div></div><div>Representa una Pieza genérica</div></div>	<div><div>Torre : Pieza</div><div><div>+ Torre (int posX, int posY) + ~Torre() + void Dibujar() + void Mover() + bool movValido(int destX, int destY) + char TipoPieza()</div></div></div> <div>Representa una Torre</div>	<div><div>Rey : Pieza</div><div><div>+ Rey(int posX, int posY) + ~Rey () + void Dibujar() + void Mover() + bool movValido(int destX, int destY) + char TipoPieza()</div></div></div> <div>Representa un Rey</div>	<div><div>Juego</div><div><div>-Pieza*tablero[ALTO][ANCHO] - bool fin - char turno;</div><div><div>+ Juego() + ~Juego() + void comenzar() + void dibujar() + bool terminar() + void moverPieza() + void validar () + void borrarTablero()</div></div></div><div>Representa un Juego</div></div>
--	---	--	--

<div><div>Peon : Pieza</div><div><div>-bool primerMov; + Peon (int posX, int posY) + ~Peon() + void Dibujar() + void Mover() + bool movValido(int destX, int destY) + char TipoPieza()</div></div></div> <div>Representa un Peón</div>





Las reglas del juego son las siguientes:

- 1) El tablero comienza con 2 torres, 2 peones y un rey de cada color, tal como indica el cuadro.
- 2) El movimiento de piezas será controlado por el mouse.
- 3) Alternativamente, cada jugador mueve una pieza, un solo lugar si la pieza es un rey y un peón y todos los deseados si es torre siempre por columnas o filas. Deberá validar que mueva el jugador habilitado.
- 4) Todo movimiento debe validarse, controlando que no caiga fuera del tablero.
- 5) Para comer una pieza del contrario el lugar es reemplazo por la pieza movida siempre que el camino esté libre.
- 6) Si el rey no puede moverse sin ser atacado es "jaque mate".
- 7) El partido termina cuando es "jaque mate", es tablas o un jugador abandona.

Otras consideraciones:

- El método **constructor** de juego carga al array con piezas blancas y negras y lee de un archivo si es necesario.
- El método **destructor** de juego elimina del array la memoria pedida por el constructor y cierra el archivo.
- El método **comenzar** juego setea las posiciones de cada pieza en el tablero.
- El método **dibujar** dibuja el tablero y luego a cada pieza dentro de él.
- El método **moverPieza** pide: el jugador, que pieza y a que celda se desea mover(coordenadas); llama a **validarPieza** y luego a **mover** de pieza y se encarga de mover o comer.

El método **validar** de juego debe validar:

- 1-si se selecciono una pieza si no "no hay una pieza en este lugar"
- 2-si la pieza a mover es propia sino "no es tu pieza!"
- 3-si el destino no es el mismo que el origen "es el mismo lugar!"
- 4-si el destino tiene una pieza propia sino "no se puede comer a una pieza propia!"
- 5-si el camino está libre si no "movimiento ilegal de la pieza" (salvo el caballo)
- 6-si el movimiento es valido para este tipo de pieza sino "movimiento ilegal" y llama a **movValido** de pieza.

Taller de Computación 3 Trabajo Práctico Final

El método **movValido** de pieza se encarga de verificar si el movimiento es válido para esa clase de pieza.

- El método **mover** de pieza: modifica sus coordenadas internas.
- El método **terminar** de **Juego** verifica si hay un ganador y devuelve el valor de la variable fin en true.
- Funciones para el manejo del Mouse:
ismouseclick(WM_LBUTTONDOWN)) Esto chequea si el botón izquierdo del mouse fue presionado.
clearmouseclick(WM_LBUTTONDOWN); Limpia el evento.
- Funciones graficas: la matriz deberá dibujarse en modo gráfico, aprovechando las clases gráficas ya codificadas para otros ejercicios (círculo, rectángulo). También deberán usarse las funciones para colocar texto en la pantalla, especiales para modo gráfico (outtextxy, outtext, settextstyle, setcolor, setfillstyle, textheight, textwidth).

El programa deberá ser codificado obligatoriamente utilizando **proyectos** (ver PDF **Proyectos con Clases**).

El Juego debe cumplir con los siguientes requisitos obligatorios.

1. Cargar el tablero con 2 torres, 2 peones y 1 rey en las posiciones iniciales, poder elegir en que posición o desde un archivo.
2. Poder mover las piezas con el mouse.
3. Grabar en un archivo el estado del tablero.

Nota: se podrán extender las clases y los métodos tanto como sea necesario.

reglas ajedrez http://es.wikipedia.org/wiki/Reglamento_del_ajedrez

Mapeo

Es la forma de pasar de una representación externa a una interna y viceversa.

Para este juego la representación externa es un tablero formado por pixeles (por ejemplo 800x600) y la interna es una matriz (de 8 x 8 posiciones) de punteros a piezas. Por lo tanto al presionar con el mouse una zona del tablero se deberá obtener el pixel de coordenada x e y y mapearlo (convertirlo a) coordenadas de x, y en la matriz. Y de x, y de la matriz a pixeles.

El ciclo de un juego o Game Loop

El ciclo de juego es una representación generalizada del flujo de eventos en el mismo. El núcleo de los eventos se repite por lo que se llama un ciclo o loop. Aunque la implementación puede ser muy diferente entre un juego y otro, la estructura fundamental es la misma para casi todos los juegos de todos los géneros. Tanto para un juego de simulador espacial como un juego de rol (RPG). Por lo general se puede descomponer el juego en los mismos componentes que se repiten en ciclo o loop del juego.

Veamos las partes del ciclo de un juego:

- **Setup**. Implica los ajustes iniciales del juego, como el sonido, música y gráficos. También puede presentarse el trasfondo del juego y sus objetivos.
- **Ingreso del jugador**. Se trata de la captura del ingreso del jugador que puede provenir desde el teclado, mouse, joystick, trackball, o algún otro dispositivo de entrada.
- **Actualización del juego**. Implica la lógica del juego y las reglas que se aplican al mundo del juego, teniendo en cuenta los aportes del jugador. Puede tomar la forma de un sistema interacción física de objetos o implicar cálculos de Inteligencia Artificial del enemigo.
- **Actualización de la pantalla**. En la mayoría de los juegos, este proceso es el más exigente del hardware del equipo, ya que a menudo implica gráficos complejos. Sin embargo, este proceso puede ser tan simple como mostrar una línea de texto.
- **Comprobación de que el juego ha finalizado**. Si el juego no ha terminado (si jugador todavía está vivo y que no ha salido), el control se bifurca de nuevo a la etapa de ingreso del jugador. Si el juego ha terminado, el control se desvía a la fase de cierre.
- **Apagado**. En este punto, el juego termina. El jugador ve a menudo la información final como ser el puntaje, tiempo, etc. El programa libera todos los recursos, si es necesario, y se cierra.

