UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2021

Versión 1: 2 de junio de 2021

TPI - "Ajedrez Lite"

Entrega: 18 de Junio (hasta las 17 hs)

1. Introducción

El siguiente es el enunciado del Trabajo Práctico de Implementación (TPI) y está basado en el TPE de Ajedrez Lite. En el TPI valen las mismas definiciones presentadas en el enunciado del TPE. Hay varios ejercicios del TPE que deben ser llevados a código C++, hay algunos que cambiaron levemente su definición, y hay un nuevo ejercicio.

2. Consignas

- Implementar todas las funciones que se encuentran en el archivo `ejercicios.h`. Para ello, deberán usar la especificación que se encuentra en la sección 5 del presente enunciado. La implementación de cada ejercicio DEBE SEGUIR OBLIGATORIAMENTE ESTA ESPECIFICACIÓN.
- Extender el conjunto de casos de tests de manera tal de lograr una cobertura de líneas mayor al 95 %. En caso de no poder alcanzarla, explicar el motivo. La cobertura debe estar chequeada con herramientas que se verán en laboratorio de la materia.
- No está permitido el uso de librerías de C++ fuera de las clásicas: **math**, **vector**, **tuple**, **pair**, las de input-output, etc. Consultar con la cátedra por cualquier librería adicional.

Dentro del archivo que se descarguen desde la página de la materia van a encontrar los siguientes archivos y carpetas:

- `definiciones.h`: Aquí están las definiciones de los tipos del TPI.
- `ejercicios.cpp`: Aquí es donde van a volcar sus implementaciones.
- `ejercicios.h`: *headers* de las funciones que tienen que implementar.
- `auxiliares.cpp` y `auxiliares.h`: Donde es posible volcar funciones auxiliares.
- `main.cpp`: Punto de entrada del programa.
- `tests`: Estos son algunos Tests Suites provistos por la materia. Aquí deben completar con sus propios Tests para lograr la cobertura pedida.
- `lib`: Todo lo necesario para correr Google Tests. Aquí no deben tocar nada.
- `CMakeLists.txt`: Archivo que necesita CLion para la compilación y ejecución del proyecto. **NO** deben sobreescribirlo al importar los fuentes desde CLion. Para ello recomendamos:
  1. Lanzar el CLION.
  2. Cerrar el proyecto si hubiese uno abierto por *default*: `File->Close Project`
  3. En la ventana de Bienvenida de CLION, seleccionar `Open Project`
  4. Seleccionar la carpeta del proyecto `tpi-template-alumnos`.
  5. Si es necesario, cargar el CMakeList.txt nuevamente mediante `Tools->CMake->Reload CMake Project`
  6. No olvidarse descomprimir el GTEST.

Es importante recalcar que la especificación de los ejercicios elaborada por la materia es la guía sobre la que debe basarse el equipo a la hora de implementar los problemas.

3. Entregable

La fecha de entrega del TPI es el **18 de Junio de 2021**.

1. Entregar una implementación de los ejercicios que cumplan el comportamiento detallado en la Especificación. Los archivos obligatorios que se deben entregan son: `ejercicios.cpp`, `auxiliares.cpp`, `auxiliares.h` y el `CMakeList.txt`. Además, incluir los casos de test adicionales propuestos por el grupo que debieron desarrollar para incrementar la cobertura.
2. El proyecto debe subirse en un archivo comprimido en la solapa Trabajos Prácticos en la tarea SUBIR TPI.
3. **Importante: Utilizar la especificación diseñada para este TP.**.
4. **Importante: Es condición necesaria que la implementación pase todos los casos de tests provistos en el directorio tests. Estos casos sirven de guía para la implementación, existiendo otros TESTS SUITES *secretos* en posesión de la materia que serán usados para la corrección.**

4. Funciones C++

La declaración de las funciones a implementar es la siguiente:

```
bool posicionValida(pair<tablero, jugador> const &p);
bool posicionInicial(posicion const &p);
vector<coordenada> casillasAtacadas(posicion const &p, jugador j);
bool posicionSiguienteCorrecta(posicion const &p1,
       posicion const &p2,
       coordenada o,
       coordenada d);
void ordenarTablero(posicion &p);
bool finDeLaPartida(posicion const &p, jugador &j);
bool hayJaqueDescubierto(posicion const &p);
void ejecutarSecuenciaForzada(posicion &p, secuencia s);
int seVieneElJaqueEn(posicion const &p);
```

Donde definimos las siguientes estructuras de datos

```
typedef pair<int, int> casilla;
typedef vector<vector <casilla>> tablero;
typedef pair<tablero, jugador> posicion;
typedef pair<int, int> coordenada;
typedef vector<pair<coordenada, coordenada>> secuencia;
```

Uso de pair Vamos a utilizar el container pair que pertenece a la librería estándar del C++ y está definido en el header **utility**. Este es un container que puede poner juntos dos elementos de cualquier tipo. En nuestro caso, vamos a utilizarlo para dos valores enteros.

Para asignar u acceder al primero se utiliza la propiedad *first*, y para el otro... *second*. En el siguiente ejemplo[1], veremos varias maneras de declarar, asignar e imprimir los valores de containers pair.

```
#include <iostream>
#include <utility>
using namespace std;

int main()
{
    pair <int, char> PAIR1 ;
    pair <string, double> PAIR2 ("GeeksForGeeks", 1.23) ;
    pair <string, double> PAIR3 ;

    PAIR1.first = 100;
    PAIR1.second = 'G' ;

    PAIR3 = make_pair ("GeeksForGeeks␣is␣Best",4.56);

    cout << PAIR1.first << "␣" ; // espacio en blanco para separar los elementos
```

[1] https://www.geeksforgeeks.org/pair-in-cpp-stl/

```
    cout << PAIR1.second << endl ;

    cout << PAIR2.first << "␣" ;
    cout << PAIR2.second << endl ;

    cout << PAIR3.first << "␣" ;
    cout << PAIR3.second << endl ;

    return 0;
}
```

5. Especificación

Implementar funciones en C++ que cumplan las siguientes especificaciones respetando el **renombre** de tipo:

type *casilla* $= \mathbb{Z} \times \mathbb{Z}$

type *tablero* $= seq\langle seq\langle$*casilla*$\rangle\rangle$

type *posicion* $=$ *tablero* $\times$ *jugador*

type *coordenada* $= \mathbb{Z} \times \mathbb{Z}$

5.1. Ejercicios

Ejercicio 1.

`proc posicionValida` (in p: *tablero* $\times$ *jugador*, out result: Bool) {

`Pre` $\{True\}$

`Post` $\{result = true \leftrightarrow esPosicionValida(p)\}$

}

Ejercicio 2.

`proc posicionInicial` (in :p *posicion*, out result: Bool) {

`Pre` $\{esPosicionValida(p)\}$

`Post` $\{piezasEnCoordenadas(p_0) \land cantidadPiezasInicio(p_0) \land jugador(p) = BLANCO\}$

`pred piezasEnCoordenadas` (t: *tablero*)) {

$peonesEnCoordenadas(t) \land torresEnCoordenadas(t) \land alfilesEnCoordenadas(t) \land reyesEnCoordenadas(t)$

}

`pred peonesEnCoordenadas` (t: *tablero*)) {

$(\forall x : \mathbb{Z})(0 \leq x < DIM \longrightarrow_L$
$(piezaEnCoordenada(t, setCoord(1, x), PEON, NEGRO) \land$
$piezaEnCoordenada(t, setCoord(6, x), PEON, BLANCO))$

}

`pred torresEnCoordenadas` (t: *tablero*)) {

$(\forall c :$ *coordenada*$)(c = setCoord(0, 0) \lor c = setCoord(0, DIM - 1)) \longrightarrow_L$
$piezaEnCoordenada(t, c, TORRE, NEGRO) \land$
$(\forall c :$ *coordenada*$)(c = setCoord(DIM - 1, 0) \lor c = setCoord(DIM - 1, DIM - 1) \longrightarrow_L$
$piezaEnCoordenada(t, c, TORRE, BLANCO))$

}

`pred alfilesEnCoordenadas` (t: *tablero*)) {

$(\forall c : \textit{coordenada})(c = setCoord(0,2) \vee c = setCoord(0, DIM-2) \longrightarrow_L$
$piezaEnCoordenada(t, c, ALFIL, NEGRO)) \wedge$
$(\forall c : \textit{coordenada})(c = setCoord(DIM-1,2) \vee c = setCoord(DIM-1, DIM-2) \longrightarrow_L$
$piezaEnCoordenada(t, c, ALFIL, BLANCO))$

}

pred reyesEnCoordenadas (t: *tablero*)) {

$piezaEnCoordenada(t, setCoord(0,4), REY, NEGRO) \wedge$
$piezaEnCoordenada(t, setCoord(DIM-1,4), REY, BLANCO)$

}

pred cantidadPiezasAlInicio (t: *tablero*)) {

$apariccionesEnTablero(t, setCasilla(TORRE, NEGRO)) = 2 \wedge$
$apariccionesEnTablero(t, setCasilla(TORRE, BLANCO)) = 2 \wedge$
$apariccionesEnTablero(t, setCasilla(ALFIL, NEGRO)) = 2 \wedge$
$apariccionesEnTablero(t, setCasilla(ALFIL, BLANCO)) = 2 \wedge$
$apariccionesEnTablero(t, setCasilla(PEON, NEGRO)) = DIM \wedge$
$apariccionesEnTablero(t, setCasilla(PEON, BLANCO)) = DIM$

}

}

Ejercicio 3.

proc casillasAtacadas (in p: *posicion*, in j: *jugador*, out atacadas: $seq\langle$*coordenada*$\rangle$) {

Pre $\{esPosicionValida(t) \wedge jugadorValido(j)\}$

Post $\{sonCasillasAtacadas(p_0, j, atacadas)\}$

pred sonCasillasAtacadas (t: *tablero*, j: *jugador*, atacadas: $seq\langle$*coordenada*$\rangle$) {

$(\forall c : \textit{coordenada})((coordenadaEnRango(c) \wedge \#apariciones(c, atacadas) = 1) \leftrightarrow$
$(\exists o : \textit{coordenada})((coordenadaEnRango(o) \wedge o \neq c \wedge_L color(t, o) = jugador) \wedge_L casillaAtacada(t, o, c))$

}

}

Ejercicio 4.

proc posicionSiguienteCorrecta (in p1: *posicion*, in p2: *posicion*, o: *coordenada*, d: *coordenada*, out res: Bool) {

Pre $\{esPosicionValida(p1) \wedge esPosicionValida(p2) \wedge coordenadaEnRango(o) \wedge coordenadaEnRango(d) \wedge_L$
$jugador(p1) = color(p1, o)\}$

Post $\{res = true \leftrightarrow posicionSiguiente(p1, p2, o, d)\}$

}

Ejercicio 5.

proc ordenarTablero (inout p: *posicion*) {

Pre $\{esPosicionValida(p) \wedge p = q\}$

Post $\{esPosicionValida(p) \wedge_L jugador(p) = jugador(q) \wedge esTableroOrdenado(p_0, q_0)\}$

pred esTableroOrdenado (t1: *tablero*, t2: *tablero*) {

$ordenado(t1) \wedge respetaVacias(t1, t2) \wedge ((\forall f : \mathbb{Z})(0 \leq f < |t1| \longrightarrow_L mismasPiezasPorFila(t1, t2, f)))$

}

pred ordenado (t: *tablero*) {

$(\forall f : \mathbb{Z})(0 \leq f < |t| \longrightarrow_L filaOrdenada(t, f))$

}

pred filaOrdenada (t: *tablero*, f:$\mathbb{Z}$) {

$(\forall i : \mathbb{Z})((0 \leq i < |t[0]| \wedge_L (\neg casillaVacia(t, setCoor(f, i)) \wedge$
$(\exists j : \mathbb{Z})(i < j < |t[0]| \wedge_L esLaSiguienteNoVacia(t, setCoor(f, i), setCoor(f, j)))) \longrightarrow_L$
$pieza(t, setCoor(f, i)) \leq pieza(t, setCoor(f, j)))$

}

pred esLaSiguienteNoVacia (t: *tablero*, o: *coordenada*, d: *coordenada*) {

$\neg casillaVacia(t, d) \wedge (\forall k : \mathbb{Z})(o_1 < k < d_1 \longrightarrow_L casillaVacia(t, setCoord(o_0, k))$

}

pred mismasPiezasPorFila (t1: *tablero*, t2: *tablero*, f: $\mathbb{Z}$) {

$(\forall c : \mathbb{Z})(0 \leq c < |t1| \wedge \neg casillaVacia(t1, setCoord(f, c)) \longrightarrow_L$
$\#apariciones(t1[f][c], t1[f]) = \#apariciones(t1[f][c], t2[f]))$

}

pred respetaVacias (t: *tablero*, ti: *tablero*) {

$(\forall c : coordenada)(coordenadaEnRango(c) \longrightarrow_L$
$casillaVacia(obtenerCasilla(t, c)) \leftrightarrow casillaVacia(obtenerCasilla(ti, c)))$

}

}

Ejercicio 6. El procedimiento devuelve verdadero si la partida ha terminado. Esto significa que es un empate, o que el jugador actual está en jaque mate. Además se devuelve el jugador ganador, que en el caso de empato, la variable j toma valor cero.

proc finDeLaPartida (in p: *posicion*, out res: Bool, out j: $\mathbb{Z}$) {

Pre $\{esPosicionValida(p)\}$

Post $\{((res = true \wedge j = 0) \leftrightarrow esEmpate(p)) \vee ((res = true \wedge j = contrincante(jugador(p))) \leftrightarrow esJaqueMate(p))\}$

pred esEmpate (p: *posicion*) {

$soloHayReyes(p_0) \vee (\neg jugadorEnJaque(p) \wedge \neg hayMovimientosLegales(p)$

}

}

Ejercicio 7.

proc hayJaqueDescubierto (in p: *posicion*, out res: Bool) {

Pre $\{esPosicionValida(p) \wedge \neg jugadorEnJaque(p, contrincante(jugador(p)))\}$

Post $\{res = true \leftrightarrow alMoverQuedaEnJaque(p)\}$

pred alMoverQuedaEnJaque (p: *posicion*) {

$(\exists o : coordenada)(\exists d : coordenada)((coordenadaEnRango(o) \wedge coordenadaEnRango(d)) \wedge_L$
$color(p_0, o) = jugador(p) \wedge ((esMovimientoValido(p, o, d) \vee esCapturaValida(p, o, d)) \wedge$
$((\exists q : posicion)(esPosicionValida(q) \wedge_L (posicionSiguiente(p, q, o, d) \wedge jugadorEnJaque(q, q_1)))$

}

}

Ejercicio 8.

proc ejecutarSecuenciaForzada (inout p: *posicion*, in s:$seq\langle coordenada \times coordenada\rangle$) {

Pre $\{esPosicionValida(p) \wedge |s| > 0 \wedge secuenciaDeCoordenadasValidas(s) \wedge_L esSecuenciaForzada(p, s) \wedge p0 = p\}$

Post $\{posicionFinalDeSecuenciaForzada(p0, p, s)\}$

}

Ejercicio 9. Este procedimiento va a ayudar al jugador que tiene el turno en p, para encontrar el número de movimientos que lo van a llevar a ganar la partida por jaque mate. La posición p, de hecho, debe corresponder a una configuración del tablero que en K movimientos del jugador que tiene el turno en p llegue al Jaque siguiendo una secuencia forzada. Vamos a imponer que el programa solo funciona para $K \leq 3$ movimientos. Armarse de paciencia que este ejercicio puede tardar su tiempo. Pero pensando bien el algoritmo, se puede calcular en un lapso razonable.

proc seVieneElJaqueEn (in p: *posicion*, out K: $\mathbb{Z}$) {

Pre $\{esPosicionValida(p) \wedge_L (\exists n : \mathbb{Z})(1 \leq n \leq 3 \wedge hayMateEn(p, n))\}$

Post $\{hayMateForzadoEn(p, K) \wedge noHayMateAntesDe(p, K)\}$

pred hayMateForzadoEn (pi: *posicion*, n: $\mathbb{Z}$) {

$(\exists F : seq\langle coordenada \times coordenada\rangle)(|F| = n \wedge secuenciaDeCoordenadasValidas(F) \wedge_L secuenciaForzadaAMate(pi, F))$

}

pred noHayMateAntesDe (pi: *posicion*, n: $\mathbb{Z}$) {

$(\forall k : \mathbb{Z})(0 \leq k < n \longrightarrow_L \neg hayMateForzadoEn(pi, k))$

}

pred secuenciaForzadaAMate (pi: *posicion*, F: $seq\langle coordenada \times coordenada\rangle$) {

$(\exists pj : posicion)(\exists pf : posicion)(esPosicionValida(pj) \wedge esPosicionValida(pf) \wedge_L posicionFinalDeSecuenciaForzada(pi, pj, subseq(F, 0, |F| - 1)) \wedge posicionSiguiente(pj, pf, F[|F| - 1]_0, F[|F| - 1]_1) \wedge esJaqueMate(pf))$

}

}

5.2. Predicados y funciones auxiliares

Los auxiliares y predicados estarán ordenados alfabeticamente para agilizar su identificación.

5.2.1. Auxiliares

aux VACIO : $\mathbb{Z} = 0$;

aux BLANCO : $\mathbb{Z} = 1$;

aux NEGRI : $\mathbb{Z} = 2$;

aux PEON : $\mathbb{Z} = 1$;

aux ALFIL : $\mathbb{Z} = 2$;

aux TORRE : $\mathbb{Z} = 3$;

aux REY : $\mathbb{Z} = 4$;

aux DIM : $\mathbb{Z} = 8$;

aux aparicionesEnTablero (t: *tablero*, p: *casilla*) : $\mathbb{Z} = \sum_{i=0}^{|t|-1} \sum_{j=0}^{|t[0]|-1}$ if $t[i][j] = p$ then 1 else 0 fi;

aux cuantasAtacanAlRey (p: *posicion*) : $\mathbb{Z} = \sum_{i=0}^{|p_0|-1} \sum_{j=0}^{|p_0[0]|-1}$ if $esDelOponente(p, (i, j)) \wedge ataca AlRey(p, (i, j))$ then 1 else 0 fi;

aux color (t: *tablero*, c: *coordenada*) : $\mathbb{Z} = (t[c_0][c_1])_1$;

aux contrincante (j: $\mathbb{Z}$) : $\mathbb{Z}$ = if $j = BLANCO$ then $NEGRO$ else $BLANCO$ fi ;

aux jugador (p: *posicion*) : $\mathbb{Z}$ = p_1 ;

aux pieza (t: *tablero*, c: *coordenada*) : $\mathbb{Z}$ = $(t[c_0][c_1])_0$;

aux setCoord (i: $\mathbb{Z}$, j: $\mathbb{Z}$) : *coordenada* = (i, j) ;

aux setCasilla (i: $\mathbb{Z}$, j: $\mathbb{Z}$) : *casilla* = (i, j) ;

5.2.2. Predicados

pred atacaAlRey (p: *posicion*, o: *coordenada*) {

$(\exists d : coordenada)((coordenadaEnRango(d) \wedge_L$
$(pieza(p_0, d) = REY \wedge color(p_0, d) = jugador(p)) \wedge esCapturaValida(p, o, d))$

}

pred cantidadValidaDePiezas (t: *tablero*) {

$piezasTorresValidas(t) \wedge piezasPeonesValidas(t) \wedge piezasAlfilesValidas(t) \wedge piezasReyesValidas(t)$

}

pred capturaPeonValida (t: *tablero*, o: *coordenada*, d: *coordenada*) {

$|d_0 - o_0| = 1 \wedge ((color(t, o) = BLANCO \wedge d_1 = o_1 - 1) \vee (color(t, o) = NEGRO \wedge d_1 = o_1 + 1))$

}

pred casillaAtacada (t: *tablero*, o: *coordenada*, d: *coordenada*) {

$\neg casillaVacia(t, o) \wedge ((pieza(t, o) \neq PEON \wedge movimientoPiezaValido(t, o, d)) \vee$
$(pieza(t, o) = PEON \wedge capturaPeonValida(t, o, d)))$

}

pred casillaVacia (t:*tablero*, c: *coordenada*) {

$t[c_0][c_1] = (VACIO, VACIO)$

}

pred casillasValidas (t: *tablero*) {

$(\forall c : coordenada)(coordenadaEnRango(c) \longrightarrow_L$
$(casillaVacia(t, c) \vee (PEON \leq pieza(t, c) \leq REY \wedge BLANCO \leq color(t, c) \leq NEGRO)))$

}

pred coordenadaEnRango (c:*coordenada*) {

$0 \leq c_0 < DIM \wedge 0 \leq c_1 < DIM$

}

pred delOponente (p: *posicion*, c: *coordenada*) {

$\neg casillaVacia(p_0, c) \wedge color(p_0, c) \neq jugador(p)$

}

pred enLineaFinalInicial (c: *coordenada*) {

$c_0 = 0 \vee c_0 = DIM - 1$

}

pred enRango (x: $\mathbb{Z}$, m1: $\mathbb{Z}$, m2: $\mathbb{Z}$)) {

$m1 < x < m2 \vee m2 < x < m1$

}

pred esCapturaValida (p:*posicion*, o: *coordenada*, d: *coordenada*) {

$\neg casillaVacia(p_0, o) \wedge \neg casillaVacia(p_0, d) \wedge color(p_0, o) \neq color(p_0, d) \wedge casillaAtacada(p_0, o, d))$

}

pred esDelOponente (p: *posicion*, c: *coordenada*) {

$\neg casillaVacia(p_0, c) \wedge color(p_0, c) \neq jugador(p)$

}

`pred esJaqueMate` (p: *posicion*) {

$jugadorEnJaque(p) \wedge \neg existeMovimientoParaSalirDelJaque(p)$

}

`pred esJugadaLegal` (p: *posicion*, o: *coordenada*, d:*coordenada*) {

$(esMovimientoValido(p, o, d) \vee esCapturaValida(p, o, d)) \wedge_L \neg loPoneEnJaque(p, o, d)$

}

`pred existeMovimientoParaSalirDelJaque` (p: *posicion*) {

$(\exists o : \textit{coordenada})(\exists d : \textit{coordenada})(coordenadaEnRango(o) \wedge coordenadaEnRango(d) \wedge_L$
$color(p_0, o) = jugador(p) \wedge esJugadaLegal(p, o, d))$

}

`pred esJugadorValido` (j: $\mathbb{Z}$) {

$j = BLANCO \vee j = NEGRO$

}

`pred esMatriz` (t: *tablero*) {

$|t| = DIM \wedge_L (\forall i : \mathbb{Z})(0 \leq i < DIM \longrightarrow_L |t[i]| = |t|)$

}

`pred esMovimientoValido` (p:*posicion*, o: *coordenada*, d: *coordenada*) {

$jugador(p) = color(p_0, o) \wedge \neg casillaVacia(p_0, o) \wedge casillaVacia(p_0, d) \wedge movimientoPiezaValido(p_0, o, d))$

}

`pred esPosicionValida` (p:*posicion*) {

$esJugadorValido(jugador(p)) \wedge esTableroValido(tablero(p))$

}

`pred esSecuenciaForzada` (pi: *posicion*, s: $seq\langle$*coordenada* $\times$ *coordenada*$\rangle$) {

$(\exists P : seq\langle \textit{posicion}\rangle)((posicionesValidas(P) \wedge |P| = (2 * |s|) + 1) \wedge_L P[0] = pi \wedge sonPosicionesForzadas(P, s))$

}

`pred esTableroValido` (t: *tablero*) {

$esMatriz(t) \wedge_L casillasValidas(t) \wedge sinPeonesNoCoronados(t) \wedge cantidadValidaDePiezas(t)$

}

`pred esUnicaMovidaPosibleDeJugador` (p: *posicion*, o: *coordenada*, d: *coordenada*) {

$(\forall o1 : \textit{coordenada})(\forall d1 : \textit{coordenada})(coordenadaEnRango(o1) \wedge coordenadaEnRango(d1) \wedge_L$
$esJugadaLegal(p, o1, d1) \longrightarrow o = o1 \wedge d = d1)$

}

`pred hayMovimientosLegales` (p: *posicion*) {

$(\exists movibles : seq\langle \textit{coordenada}\rangle)(|movibles| > 0 \wedge coordenadasEnRango(movibles) \wedge_L$
$pizasDeJugador(jugador(p), movibles)) \wedge_L tienenMovimiento(p, movibles)$

}

`pred loPoneEnJaque` (p: *posicion*, o: *coordenada*, d:*coordenada*) {

$((\exists q : \textit{posicion})esPosicionValida(q) \wedge_L posicionSiguiente(p, q, o, d) \wedge jugador(p) = jugador(q) \wedge jugadorEnJaque(q))$

}

`pred movimientoPiezaValido` (t:*tablero*, o: *coordenada*, d: *coordenada*) {

$(pieza(t, o) = PEON \wedge movimientoPeonValido(color(t, o), o, d)) \vee$
$(pieza(t, o) = ALFIL \wedge movimientoAlfilValdo(t, o, d)) \vee$

$(pieza(t, o) = TORRE \land movimientoTorreValido(t, o, d)) \lor$
$(pieza(t, o) = REY \land movimientoReyValido(o, d))$

}

`pred movimientoAlfilValido` (t: *tablero*, o: *coordenada*, d: *coordenada*) {

$abs(d_0 - o_0) = abs(d_1 - o_1) \land$
$(\forall x : \mathbb{Z})(enRango(x, 0, d_0 - o_0) \longrightarrow_L$
$((\forall y : \mathbb{Z})((enRango(y, 0, d_1 - o_1) \land abs(x) = abs(y)) \longrightarrow_L casillaVacia(t, setCoord(o_0 + x, o_1 + y)))$

}

`pred movimientoForzado` (pi: *posicion*, pm: *posicion*, pf: *posicion*, m: *coordenada* $\times$ *coordenada*) {

$esJugadaLegal(pi, m_0, m_1) \land posicionSiguiente(pi, pm, m_0, m_1) \land$
$(\exists o : \textit{coordenada})(\exists d : \textit{coordenada})(coordenadaEnRango(o) \land coordenadaEnRango(d) \land_L$
$o \neq d \land esJugadaLegal(pm, o, d) \land esUnicaMovidaPosibleDeJugador(pm, o, d) \land posicionSiguiente(pm, pf, o, d))$

}

`pred movimientoPeonValido` (color: $\mathbb{Z}$, o: *coordenada*, d: *coordenada*) {

$d_1 = o_1 \land ((color = BLANCO \land d_0 = o_0 - 1) \lor (color = NEGRO \land d_0 = o_0 + 1))$

}

`pred movimientoReyValido` (o: *coordenada*, d: *coordenada*)) {

$mueveEnDiagonal(o, d) \lor mueveEnVertical(o, d) \lor mueveEnHorizontal(o, d)$

}

`pred movimientoTorreValido` (t: *tablero*, o: *coordenada*, d: *coordenada*)) {

$(d_1 = o_1 \land (\forall x : \mathbb{Z})(enRango(x, o_0, d_0) \longrightarrow_L casillaVacia(t, setCoord(x, o_1)))) \lor$
$(d_0 = o_0 \land (\forall y : \mathbb{Z})(enRango(y, o_1, d_1) \longrightarrow_L casillaVacia(t, setCoord(o_0, y))))$

}

`pred mueveEnHorizontal` (o: *coordenada*, d: *coordenada*)) {

$abs(o_0 - d_0) = 0 \land abs(o_1 - d_1) = 1$

}

`pred mueveEnDiagonal` (o: *coordenada*, d: *coordenada*)) {

$abs(o_0 - d_0) = 1 \land abs(o_1 - d_1) = 1$

}

`pred mueveEnVertical` (o: *coordenada*, d: *coordenada*)) {

$abs(o_0 - d_0) = 1 \land abs(o_1 - d_1) = 0$

}

`pred mueveJugadorEnOrigen` (p: *posicion*, o:*coordenada*) {

$color(p_0, o) = jugador(p)$

}

`pred piezaCorrectaEnDestino` (p: *posicion*, q: *posicion*, o: *coordenada*, d:*coordenada*) {

$color(p_0, d) = color(q_0, d) \land ((enLineaFinalInicial(d) \land pieza(q_0, d) = TORRE) \lor$
$(\neg enLineaFinalInicial(d) \land pieza(q_0, d) = pieza(p_0, o)))$

}

`pred piezaEnCoordenada` (t:*tablero*, c: *coordenada*, pza: $\mathbb{Z}$, col: $\mathbb{Z}$) {

$pieza(t, c) = pza \land color(t, c) = col$

}

`pred piezasAlfilesValidas` (t: *tablero*) {

$aparicionesEnTablero(t, (ALFIL, BLANCO)) \leq 2 \land aparicionesEnTablero(t, (ALFIL, NEGRO)) \leq 2$

}

pred piezasPeonesValidas (t: *tablero*) {

$aparicionesEnTablero(t, (PEON, BLANCO)) \leq DIM \land aparicionesEnTablero(t, (PEON, NEGRO)) \leq DIM$

}

pred piezasReyesValidas (t: *tablero*) {

$aparicionesEnTablero(t, (REY, BLANCO)) = 1 \land aparicionesEnTablero(t, (REY, NEGRO)) = 1$

}

pred piezasTorresValidas (t: *tablero*) {

$aparicionesEnTablero(t, (TORRE, BLANCO)) \leq 2 + (DIM - aparicionesEnTablero(t, (PEON, BLANCO))) \land$
$aparicionesEnTablero(t, (TORRE, NEGRO)) \leq 2 + (DIM - aparicionesEnTablero(t, (PEON, NEGRO)))$

}

pred piezaValida (t:*tablero*, c: *coordenada*) {

$0 \leq pieza(t, c) \leq 4 \land 0 \leq color(t, c) \leq 2 \land (pieza(t, c) = VACIO \leftrightarrow color(t, c) = VACIO)$

}

pred posicionesIgualesExceptoEn (p: *posicion*, q: *posicion*, C: $seq\langle$*coordenada*$\rangle$) {

$(\forall c : coordenada)(coordenadaEnRango(c) \land c \notin C \longrightarrow_L pieza(p_0, c) = pieza(q_0, c) \land color(p_0, c) = color(q_0, c))$

}

pred posicionFinalDeSecuenciaForzada (pi: *posicion*, p: *posicion*, s: $seq\langle$*coordenada* $\times$ *coordenada*$\rangle$) {

$(\exists P : seq\langle posicion\rangle)(((posicionesValidas(P) \land |P| = (2 * |s|) + 1) \land_L P[0] = pi \land sonPosicionesForzadas(P, s)) \land_L p = P[|P| - 1])$

}

pred posicionesValidas (P: $seq\langle$*posicion*$\rangle$) {

$(\forall p : posicion)(p \in P \longrightarrow_L esPosicionValida(p))$

}

pred posicionSiguiente (p: *posicion*, q: *posicion*, o: *coordenada*, d:*coordenada*) {

$posicionesIgualesExceptoEn(p, q, < setCoord(o, d) >) \land casillaVacia(q_0, o) \land (esMovimientoValido(p, o, d) \lor esCapturaValida(p, o, d)) \land piezaCorrectaEnDestino(p, q, o, d)$

}

pred secuenciaDeCoordenadasValidas (S:$seq\langle$*coordenada* $\times$ *coordenada*$\rangle$) {

$(\forall c : coordenada \times coordenada)(c \in S \longrightarrow_L coordenadaEnRango(c_0) \land coordenadaEnRango(c_1))$

}

pred sinPeonesNoCoronados (t: *tablero*) {

$(\forall i : \mathbb{Z})(0 \leq i < DIM \longrightarrow_L (pieza(t, setCoord(0, i)) \neq PEON \land pieza(t, setCoord(DIM - 1, i)) = PEON))$

}

pred soloHayReyes (t: *tablero*) {

$(\forall c : coordenada)coordenadaEnRango(c) \longrightarrow_L casillaVacia(t, c) \lor pieza(t, c) = REY$

}

pred sonPosicionesForzadas (P: $seq\langle$*posicion*$\rangle$, s: $seq\langle$*coordenada* $\times$ *coordenada*$\rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L (movimientoForzado(P[2 * i], P[(2 * i) + 1], P[(2 * i) + 2], s[i])))$

}

pred tienenMovimiento (p: *posicion*, piezas:$seq\langle$*coordenada*$\rangle$) {

$(\forall c : coordenada)(c \in piezas \longrightarrow_L ((\exists d : coordenada)enRango(d) \land_L$
$(esMovimientoValido(p_0, c, d) \lor esCapturaValida(p_0, c, d)))$

}