

## Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2021

Versión 1: 3 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 sobrescribirlo 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 entregar 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<sup>1</sup>, 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
```

---

<sup>1</sup><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⟨seq⟨casilla⟩⟩
type posicion = tablero × jugador
type coordenada =  $\mathbb{Z} \times \mathbb{Z}$ 

```

### 5.1. Ejercicios

#### Ejercicio 1.

```

proc posicionValida (in p: tablero × jugador, out result: Bool) {
    Pre {True}
    Post {result = true ↔ esPosicionValida(p)}
}

```

#### Ejercicio 2.

```

proc posicionInicial (in :p posicion, out result: Bool) {
    Pre {esPosicionValida(p)}
    Post {piezasEnCoordenadas(p0) ∧ cantidadPiezasAlInicio(p0) ∧ jugador(p) = BLANCO}

    pred piezasEnCoordenadas (t: tablero) {
        peonesEnCoordenadas(t) ∧ torresEnCoordenadas(t) ∧ alfilesEnCoordenadas(t) ∧ reyesEnCoordenadas(t)
    }

    pred peonesEnCoordenadas (t: tablero) {
        (∀x :  $\mathbb{Z}$ )(0 ≤ x < DIM →L
            (piezaEnCoordenada(t, setCoord(1, x), PEON, NEGRO) ∧
            piezaEnCoordenada(t, setCoord(6, x), PEON, BLANCO))
        )
    }

    pred torresEnCoordenadas (t: tablero) {
        (∀c : coordenada)(c = setCoord(0, 0) ∨ c = setCoord(0, DIM − 1) →L
            piezaEnCoordenada(t, c, TORRE, NEGRO) ∧
            (∀c : coordenada)(c = setCoord(DIM − 1, 0) ∨ c = setCoord(DIM − 1, DIM − 1) →L
                piezaEnCoordenada(t, c, TORRE, BLANCO))
        )
    }

    pred alfilesEnCoordenadas (t: tablero) {

```

```

    ( $\forall c : \text{coordenada}$ )( $c = \text{setCoord}(0, 2) \vee c = \text{setCoord}(0, DIM - 2) \longrightarrow_L$ 
     $\text{piezaEnCoordenada}(t, c, ALFIL, NEGRO)$ )  $\wedge$ 
    ( $\forall c : \text{coordenada}$ )( $c = \text{setCoord}(DIM - 1, 2) \vee c = \text{setCoord}(DIM - 1, DIM - 2) \longrightarrow_L$ 
     $\text{piezaEnCoordenada}(t, c, ALFIL, BLANCO)$ )
  }
pred reyesEnCoordenadas (t: tablero) {
   $\text{piezaEnCoordenada}(t, \text{setCoord}(0, 4), REY, NEGRO) \wedge$ 
   $\text{piezaEnCoordenada}(t, \text{setCoord}(DIM - 1, 4), REY, BLANCO)$ 
}
pred cantidadPiezasAlInicio (t: tablero) {
   $\text{aparicionesEnTablero}(t, \text{setCasilla}(TORRE, NEGRO)) = 2 \wedge$ 
   $\text{aparicionesEnTablero}(t, \text{setCasilla}(TORRE, BLANCO)) = 2 \wedge$ 
   $\text{aparicionesEnTablero}(t, \text{setCasilla}(ALFIL, NEGRO)) = 2 \wedge$ 
   $\text{aparicionesEnTablero}(t, \text{setCasilla}(ALFIL, BLANCO)) = 2 \wedge$ 
   $\text{aparicionesEnTablero}(t, \text{setCasilla}(PEON, NEGRO)) = DIM \wedge$ 
   $\text{aparicionesEnTablero}(t, \text{setCasilla}(PEON, BLANCO)) = DIM$ 
}
}

```

### Ejercicio 3.

```

proc casillasAtacadas (in p: posicion, in j: jugador, out atacadas: seq<coordenada>) {
  Pre { $\text{esPosicionValida}(t) \wedge \text{jugadorValido}(j)$ }
  Post { $\text{sonCasillasAtacadas}(p_0, j, \text{atacadas})$ }

  pred sonCasillasAtacadas (t: tablero, j: jugador, atacadas: seq<coordenada>) {
    ( $\forall c : \text{coordenada}$ )( $(\text{coordenadaEnRango}(c) \wedge \#apariciones(c, \text{atacadas}) = 1) \leftrightarrow$ 
    ( $\exists o : \text{coordenada}$ )( $(\text{coordenadaEnRango}(o) \wedge o \neq c \wedge_L \text{color}(t, o) = \text{jugador}) \wedge_L \text{casillaAtacada}(t, o, c)$ ))
  }
}

```

### Ejercicio 4.

```

proc posicionSiguienteCorrecta (in p1: posicion, in p2: posicion, o: coordenada, d: coordenada, out res: Bool) {
  Pre { $\text{esPosicionValida}(p1) \wedge \text{esPosicionValida}(p2) \wedge \text{coordenadaEnRango}(o) \wedge \text{coordenadaEnRango}(d) \wedge_L$ 
   $\text{jugador}(p1) = \text{color}(p1, o)$ }
  Post { $\text{res} = \text{true} \leftrightarrow \text{posicionSiguiente}(p1, p2, o, d)$ }
}

```

### Ejercicio 5.

```

proc ordenarTablero (inout p: posicion) {
  Pre { $\text{esPosicionValida}(p) \wedge p = q$ }
  Post { $\text{esPosicionValida}(p) \wedge_L \text{jugador}(p) = \text{jugador}(q) \wedge \text{esTableroOrdenado}(p_0, q_0)$ }

  pred esTableroOrdenado (t1: tablero, t2: tablero) {
     $\text{ordenado}(t1) \wedge \text{respetarVacias}(t1, t2) \wedge ((\forall f : \mathbb{Z})(0 \leq f < |t1| \longrightarrow_L \text{mismasPiezasPorFila}(t1, t2, f)))$ 
  }
  pred ordenado (t: tablero) {

```

```

    ( $\forall f : \mathbb{Z})(0 \leq f < |t| \longrightarrow_L \text{filaOrdenada}(t, f))$ )
  }
  pred filaOrdenada (t: tablero, f:  $\mathbb{Z}$ ) {
    ( $\forall i : \mathbb{Z})(0 \leq i < |t[0]| \wedge_L (\neg \text{casillaVacía}(t, \text{setCoord}(f, i)) \wedge$ 
    ( $\exists j : \mathbb{Z})(i < j < |t[0]| \wedge_L \text{esLaSiguienteNoVacía}(t, \text{setCoord}(f, i), \text{setCoord}(f, j))) \longrightarrow_L$ 
     $\text{pieza}(t, \text{setCoord}(f, i)) \leq \text{pieza}(t, \text{setCoord}(f, j)))$ )
  }
  pred esLaSiguienteNoVacía (t: tablero, o: coordenada, d: coordenada) {
     $\neg \text{casillaVacía}(t, d) \wedge (\forall k : \mathbb{Z})(o_1 < k < d_1 \longrightarrow_L \text{casillaVacía}(t, \text{setCoord}(o_0, k)))$ 
  }
  pred mismasPiezasPorFila (t1: tablero, t2: tablero, f:  $\mathbb{Z}$ ) {
    ( $\forall c : \mathbb{Z})(0 \leq c < |t1| \wedge \neg \text{casillaVacía}(t1, \text{setCoord}(f, c)) \longrightarrow_L$ 
     $\#apariciones(t1[f][c], t1[f]) = \#apariciones(t1[f][c], t2[f])$ )
  }
  pred respetaVacías (t: tablero, ti: tablero) {
    ( $\forall c : \text{coordenada})(\text{coordenadaEnRango}(c) \longrightarrow_L$ 
     $\text{casillaVacía}(\text{obtenerCasilla}(t, c)) \leftrightarrow \text{casillaVacía}(\text{obtenerCasilla}(ti, c)))$ 
  }
}

```

**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 empate, 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(p0)  $\vee$  ( $\neg$ jugadorEnJaque(p, p1)  $\wedge$   $\neg$ 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) {
    ( $\exists o : \text{coordenada})(\exists d : \text{coordenada})((\text{coordenadaEnRango}(o) \wedge \text{coordenadaEnRango}(d)) \wedge_L$ 
     $\text{color}(p_0, o) = \text{jugador}(p) \wedge ((\text{esMovimientoValido}(p, o, d) \vee \text{esCapturaValida}(p, o, d)) \wedge$ 
     $(\exists q : \text{posicion})(\text{esPosicionValida}(q) \wedge_L (\text{posicionSiguiente}(p, q, o, d) \wedge \text{jugadorEnJaque}(q, q_1)))$ )
  }
}

```

**Ejercicio 8.**

```

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

```

```

Pre {esPosicionValida(p) ∧ |s| > 0 ∧ secuenciaDeCoordenadasValidas(s) ∧L esSecuenciaForzada(p, s) ∧ 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: ℤ) {
  Pre {esPosicionValida(p) ∧L (∃n : ℤ)(1 ≤ n ≤ 3 ∧ hayMateForzadoEn(p, n))}
  Post {hayMateForzadoEn(p, K) ∧ noHayMateAntesDe(p, K)}

  pred hayMateForzadoEn (pi: posicion, n: ℤ) {
    (∃F : seq⟨coordenada × coordenada⟩)(|F| = n ∧ secuenciaDeCoordenadasValidas(F) ∧L
    secuenciaForzadaAMate(pi, F))
  }

  pred noHayMateAntesDe (pi: posicion, n: ℤ) {
    (∀k : ℤ)(0 ≤ k < n →L ¬hayMateForzadoEn(pi, k))
  }

  pred secuenciaForzadaAMate (pi: posicion, F: seq⟨coordenada × coordenada⟩) {
    (∃pj : posicion)(∃pf : posicion)(esPosicionValida(pj) ∧ esPosicionValida(pf) ∧L
    posicionFinalDeSecuenciaForzada(pi, pj, subseq(F, 0, |F| - 1)) ∧
    posicionSiguiente(pj, pf, F[|F| - 1]0, F[|F| - 1]1) ∧ 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 : ℤ = 0;
aux BLANCO : ℤ = 1;
aux NEGRI : ℤ = 2;
aux PEON : ℤ = 1;
aux ALFIL : ℤ = 2;
aux TORRE : ℤ = 3;
aux REY : ℤ = 4;
aux DIM : ℤ = 8;

aux aparicionesEnTablero (t: tablero, p: casilla) : ℤ = ∑i=0|t|-1 ∑j=0|t[0]|-1 if t[i][j] = p then 1 else 0 fi;
aux color (t: tablero, c: coordenada) : ℤ = (t[c0][c1])1;
aux contrincante (j: ℤ) : ℤ = if j = BLANCO then NEGRO else BLANCO fi;
aux jugador (p: posicion) : ℤ = p1;

```

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) {  
  ( $\exists d : \textit{coordenada}$ )( $(\textit{coordenadaEnRango}(d) \wedge_L$   
    ( $\textit{pieza}(p_0, d) = \textit{REY} \wedge \textit{color}(p_0, d) = \textit{jugador}(p)) \wedge \textit{esCapturaValida}(p, o, d)$ )  
}  
  
pred cantidadValidaDePiezas (t: tablero) {  
   $\textit{piezasTorresValidas}(t) \wedge \textit{piezasPeonesValidas}(t) \wedge \textit{piezasAlfilesValidas}(t) \wedge \textit{piezasReyesValidas}(t)$   
}  
  
pred capturaPeonValida (t: tablero, o: coordenada, d: coordenada) {  
   $|d_0 - o_0| = 1 \wedge ((\textit{color}(t, o) = \textit{BLANCO} \wedge d_1 = o_1 - 1) \vee (\textit{color}(t, o) = \textit{NEGRO} \wedge d_1 = o_1 + 1))$   
}  
  
pred casillaAtacada (t: tablero, o: coordenada, d: coordenada) {  
   $\neg \textit{casillaVacía}(t, o) \wedge ((\textit{pieza}(t, o) \neq \textit{PEON} \wedge \textit{movimientoPiezaValido}(t, o, d)) \vee$   
    ( $\textit{pieza}(t, o) = \textit{PEON} \wedge \textit{capturaPeonValida}(t, o, d)))$   
}  
  
pred casillaVacía (t: tablero, c: coordenada) {  
   $t[c_0][c_1] = (\textit{VACIO}, \textit{VACIO})$   
}  
  
pred casillasValidas (t: tablero) {  
  ( $\forall c : \textit{coordenada}$ )( $\textit{coordenadaEnRango}(c) \longrightarrow_L$   
    ( $\textit{casillaVacía}(t, c) \vee (\textit{PEON} \leq \textit{pieza}(t, c) \leq \textit{REY} \wedge \textit{BLANCO} \leq \textit{color}(t, c) \leq \textit{NEGRO}))$ )  
}  
  
pred coordenadaEnRango (c: coordenada) {  
   $0 \leq c_0 < \textit{DIM} \wedge 0 \leq c_1 < \textit{DIM}$   
}  
  
pred enLineaFinalInicial (c: coordenada) {  
   $c_0 = 0 \vee c_0 = \textit{DIM} - 1$   
}  
  
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) {  
   $\neg \textit{casillaVacía}(p_0, o) \wedge \neg \textit{casillaVacía}(p_0, d) \wedge \textit{color}(p_0, o) \neq \textit{color}(p_0, d) \wedge \textit{casillaAtacada}(p_0, o, d)$   
}  
  
pred esJaqueMate (p: posicion) {  
   $\textit{jugadorEnJaque}(p, p_1) \wedge \neg \textit{existeMovimientoParaSalirDelJaque}(p)$   
}  
  
pred esJugadaLegal (p: posicion, o: coordenada, d: coordenada) {  
  ( $\textit{esMovimientoValido}(p, o, d) \vee \textit{esCapturaValida}(p, o, d)$ )  $\wedge_L \neg \textit{loPoneEnJaque}(p, o, d)$   
}
```

```

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

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) {
   $jugador(p) = color(p_0, o) \wedge \neg casillaVacía(p_0, o) \wedge casillaVacía(p_0, d) \wedge movimientoPiezaValido(p_0, o, d)$ 
}

pred esPosicionValida (p:posicion) {
   $esJugadorValido(jugador(p)) \wedge esTableroValido(tablero(p))$ 
}

pred esSecuenciaForzada (pi: posicion, s: seq<coordenada  $\times$  coordenada>) {
  ( $\exists P : seq<posicion>$ )( $(posicionesValidas(P) \wedge |P| = (2 * |s|) + 1) \wedge_L P[0] = pi \wedge sonPosicionesForzadas(P, s)$ )
}

pred esTableroValido (t: tablero) {
   $esMatriz(t) \wedge_L casillasValidas(t) \wedge sinPeonesNoCoronados(t) \wedge cantidadValidaDePiezas(t)$ 
}

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

pred hayMovimientosLegales (p: posicion) {
  ( $\exists movibles : seq<coordenada>$ )( $|movibles| > 0 \wedge coordenadasEnRango(movibles) \wedge_L$ 
   $piezasDeJugador(p, movibles) \wedge_L tienenMovimiento(p, movibles)$ )
}

pred jugadorEnJaque (p: posicion, j:  $\mathbb{Z}$ ) {
  ( $\exists q : posicion$ )( $\exists o : coordenada$ )( $q_0 = p_0 \wedge jugador(q) = j \wedge$ 
   $coordenadaEnRango(o) \wedge_L color(q_0, o) = contrincante(j) \wedge atacaAlRey(q, o)$ )
}

pred loPoneEnJaque (p: posicion, o: coordenada, d:coordenada) {
  ( $(\exists q : posicion)esPosicionValida(q) \wedge_L posicionSiguiente(p, q, o, d) \wedge jugador(p) = jugador(q) \wedge jugadorEnJaque(q, q_1)$ )
}

pred movimientoPiezaValido (t:tablero, o: coordenada, d: coordenada) {
  ( $pieza(t, o) = PEON \wedge movimientoPeonValido(color(t, o), o, d)$ )  $\vee$ 
  ( $pieza(t, o) = ALFIL \wedge movimientoAlfilValido(t, o, d)$ )  $\vee$ 
  ( $pieza(t, o) = TORRE \wedge movimientoTorreValido(t, o, d)$ )  $\vee$ 
  ( $pieza(t, o) = REY \wedge movimientoReyValido(o, d)$ )
}

pred movimientoAlfilValido (t: tablero, o: coordenada, d: coordenada) {
   $abs(d_0 - o_0) = abs(d_1 - o_1) \wedge$ 

```



```

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

pred movimientoForzado (pi: posicion, pm: posicion, pf: posicion, m: coordenada  $\times$  coordenada) {
  esJugadaLegal(pi, m0, m1)  $\wedge$  posicionSiguiente(pi, pm, m0, m1)  $\wedge$ 
  ( $\exists o : coordenada$ )( $\exists d : coordenada$ )(coordenadaEnRango(o)  $\wedge$  coordenadaEnRango(d)  $\wedge_L$ 
   $o \neq d \wedge esJugadaLegal(pm, o, d) \wedge esÚnicaMovidaPosibleDeJugador(pm, o, d) \wedge posicionSiguiente(pm, pf, o, d)$ )
}

pred movimientoPeonValido (color:  $\mathbb{Z}$ , o: coordenada, d: coordenada) {
   $d_1 = o_1 \wedge ((color = BLANCO \wedge d_0 = o_0 - 1) \vee (color = NEGRO \wedge d_0 = o_0 + 1))$ 
}

pred movimientoReyValido (o: coordenada, d: coordenada) {
  mueveEnDiagonal(o, d)  $\vee$  mueveEnVertical(o, d)  $\vee$  mueveEnHorizontal(o, d)
}

pred movimientoTorreValido (t: tablero, o: coordenada, d: coordenada) {
  ( $d_1 = o_1 \wedge (\forall x : \mathbb{Z})(enRango(x, o_0, d_0) \longrightarrow_L casillaVacía(t, setCoord(x, o_1)))) \vee$ 
  ( $d_0 = o_0 \wedge (\forall y : \mathbb{Z})(enRango(y, o_1, d_1) \longrightarrow_L casillaVacía(t, setCoord(o_0, y))))$ )
}

pred mueveEnHorizontal (o: coordenada, d: coordenada) {
   $abs(o_0 - d_0) = 0 \wedge abs(o_1 - d_1) = 1$ 
}

pred mueveEnDiagonal (o: coordenada, d: coordenada) {
   $abs(o_0 - d_0) = 1 \wedge abs(o_1 - d_1) = 1$ 
}

pred mueveEnVertical (o: coordenada, d: coordenada) {
   $abs(o_0 - d_0) = 1 \wedge abs(o_1 - d_1) = 0$ 
}

pred mueveJugadorEnOrigen (p: posicion, o: coordenada) {
   $color(p_0, o) = jugador(p)$ 
}

pred piezaCorrectaEnDestino (p: posicion, q: posicion, o: coordenada, d: coordenada) {
   $color(p_0, d) = color(q_0, d) \wedge ((enLineaFinalInicial(d) \wedge pieza(q_0, d) = TORRE) \vee$ 
   $(\neg enLineaFinalInicial(d) \wedge pieza(q_0, d) = pieza(p_0, o)))$ 
}

pred piezaEnCoordenada (t: tablero, c: coordenada, pza:  $\mathbb{Z}$ , col:  $\mathbb{Z}$ ) {
   $pieza(t, c) = pza \wedge color(t, c) = col$ 
}

pred piezasAlfilesValidas (t: tablero) {
   $aparicionesEnTablero(t, (ALFIL, BLANCO)) \leq 2 \wedge aparicionesEnTablero(t, (ALFIL, NEGRO)) \leq 2$ 
}

pred piezasDeJugador (p: posicion, C: seq(coordenada)) {
  ( $\forall c : coordenada$ )( $c \in C \longrightarrow_L color(p_0, c) = jugador(p)$ )
}

pred piezasPeonesValidas (t: tablero) {

```

```

    aparicionesEnTablero(t, (PEON, BLANCO)) ≤ DIM ∧ aparicionesEnTablero(t, (PEON, NEGRO)) ≤ DIM
}

pred piezasReyesValidas (t: tablero) {
    aparicionesEnTablero(t, (REY, BLANCO)) = 1 ∧ aparicionesEnTablero(t, (REY, NEGRO)) = 1
}

pred piezasTorresValidas (t: tablero) {
    aparicionesEnTablero(t, (TORRE, BLANCO)) ≤ 2 + (DIM - aparicionesEnTablero(t, (PEON, BLANCO))) ∧
    aparicionesEnTablero(t, (TORRE, NEGRO)) ≤ 2 + (DIM - aparicionesEnTablero(t, (PEON, NEGRO)))
}

pred posicionesIgualesExceptoEn (p: posicion, q: posicion, C: seq⟨coordenada⟩) {
    (∀c : coordenada)(coordenadaEnRango(c) ∧ c ∉ C →L pieza(p0, c) = pieza(q0, c) ∧ color(p0, c) = color(q0, c))
}

pred posicionFinalDeSecuenciaForzada (pi: posicion, p: posicion, s: seq⟨coordenada × coordenada⟩) {
    (∃P : seq⟨posicion⟩)((posicionesValidas(P) ∧ |P| = (2 * |s|) + 1) ∧L P[0] = pi ∧ sonPosicionesForzadas(P, s)) ∧L p =
    P[|P| - 1])
}

pred posicionesValidas (P: seq⟨posicion⟩) {
    (∀p : posicion)(p ∈ P →L esPosicionValida(p))
}

pred posicionSiguiente (p: posicion, q: posicion, o: coordenada, d: coordenada) {
    posicionesIgualesExceptoEn(p, q, < setCoord(o, d) >) ∧ casillaVacía(q0, o) ∧ (esMovimientoValido(p, o, d) ∨
    esCapturaValida(p, o, d)) ∧ piezaCorrectaEnDestino(p, q, o, d)
}

pred secuenciaDeCoordenadasValidas (S: seq⟨coordenada × coordenada⟩) {
    (∀c : coordenada × coordenada)(c ∈ S →L coordenadaEnRango(c0) ∧ coordenadaEnRango(c1))
}

pred sinPeonesNoCoronados (t: tablero) {
    (∀i : ℤ)(0 ≤ i < DIM →L (pieza(t, setCoord(0, i)) ≠ PEON ∧ pieza(t, setCoord(DIM - 1, i)) = PEON))
}

pred soloHayReyes (t: tablero) {
    (∀c : coordenada)coordenadaEnRango(c) →L casillaVacía(t, c) ∨ pieza(t, c) = REY
}

pred sonPosicionesForzadas (P: seq⟨posicion⟩, s: seq⟨coordenada × coordenada⟩) {
    (∀i : ℤ)(0 ≤ i < |s| →L (movimientoForzado(P[2 * i], P[(2 * i) + 1], P[(2 * i) + 2], s[i])))
}

pred tienenMovimiento (p: posicion, piezas: seq⟨coordenada⟩) {
    (∀c : coordenada)(c ∈ piezas →L ((∃d : coordenada)enRango(d) ∧L
    (esMovimientoValido(p0, c, d) ∨ esCapturaValida(p0, c, d))))
}

```