OBLIGATORIO PROGRAMACIÓN

Segundo Semestre – 2017

Universidad ORT
Facultad de Ingeniería

Grupo M2A IZ YI IE

Nombre completo:
Ángela Belén Martínez Antonini

C.I.:
5108186 -1

Nº de estudiante:
199811

Nombre completo:
Allen Federico Hermida Ventoso

C.I.:
4.716.364-7

Nº de estudiante:
219611

ÍNDICE

Prueba de datos 2
Prueba de datos: Menú 2
Prueba de datos: Partida 4
Ejemplo de partida en tablero 3x3 4
Ejemplo de partida en tablero 5x5 7
Ganar, perder y empatar 9
Opciones válidas para ambos tableros 9
Opciones válidas para las fichas 10
Casos de uso 12
Comparacion con competencia 13
Evidencia de Testing 13
Código del programa 14
Ficha 14
Jugador 15
Juego 17
Interfaz 18
ObligatorioP2 32
Partida 32
UML 44

PRUEBA DE DATOS

PRUEBA DE DATOS: MENÚ

Los siguientes comandos son ingresados desde el menú principal del juego, el cual es el siguiente:

```
  ===> ¡Bienvenido a Inversiones! <===
----------------------------------------

===-- Menu --===

1 -   Registro de Jugador
2 -   Jugar
3 -   Ranking
4 -   Salir

----------------------------------------

---> Ingrese el numero de la opcion deseada:
```

Se hace	Se espera	Que pasa
Se ingresa la opción 1: Registro de jugador en el menú del juego.	Que el programa acepte la opción y se pueda registrar un jugador.	Se cumple lo esperado.
Se ingresa un numero seguido por un espacio y una combinación de letras o números, por ejemplo “1 asjej” (opción Registrar un Jugador), en el menú del juego.	Que el programa acepte la opción y se pueda registrar un jugador.	Se cumple lo esperado.
En el menú del juego se ingresa una letra, por ejemplo “hola”.	Que el programa no acepte el comando y que pida nuevamente un comando al usuario.	Se cumple lo esperado.
En la opción 2: Registro de Jugador se ingresa un número o una letra para el nombre del jugador, este caso se ingresa “Hola”	Que el programa la acepte y prosiga el registro.	Se cumple lo esperado.
En la opción 2: Registro de Jugador se ingresa un Enter para registrar el usuario, el nombre y la edad del jugador.	Que el programa siga pidiendo el comando hasta que se ingrese.	Se cumple lo esperado.
En la opción 2: Registro de Jugador se ingresa un número o una letra para el alias del jugador, en este caso se ingresa “Hola”.	Que el programa la acepte y prosiga el registro.	Se cumple lo esperado.

En la opción 2: Registro de Jugador se ingresa un número fuera del rango para la edad del jugador. Se probó con -1.	Que el programa no acepte lo ingresado y pida nuevamente la edad	Se cumple lo esperado.
En la opción 2: Registro de Jugador se ingresa una letra para la edad del jugador. Se probó con Angie.	Que el programa no acepte lo ingresado y pida nuevamente la edad	Se cumple lo esperado.
En la opción 2: Registro de Jugador se ingresa un número dentro del rango para la edad del jugador. Se probó con 19.	Que el programa la acepte y prosiga el registro.	Se cumple lo esperado.
Se ingresa la opción 2: Jugar habiendo ingresado menos de dos jugadores.	Que el programa no permita jugar al usuario ya que no se han registrado jugadores suficientes.	Se cumple lo esperado.
Se ingresa la opción 2: Jugar.	Que el programa permita la configuración para poder jugar, es decir que se muestre la lista de jugadores y el tamaño del tablero a seleccionar.	Se cumple lo esperado.
En la opción 2: Jugar, al mostrar la lista de jugadores se seleccionan dos diferentes.	Que el programa lo permita y prosiga con la selección del tablero.	Se cumple lo esperado.
En la opción 2: Jugar, al mostrar la lista de jugadores se seleccionan dos iguales.	Que el programa no valide esta acción y que pida nuevamente la selección de un jugador.	Se cumple lo esperado.
En la opción 2: Jugar, al mostrar la lista de jugadores se ingresa un numero fuera del rango, por ejemplo 0.	Que el programa no valide esta acción y que pida nuevamente la selección de un jugador.	Se cumple lo esperado.
En la opción 2: Jugar, al mostrar la lista de jugadores se ingresa un numero fuera del rango, por ejemplo, habiendo dos jugadores ingresados el usuario ingresa el número 3.	Que el programa no valide esta acción y que pida nuevamente la selección de un jugador.	Se cumple lo esperado.
En la opción 2: Jugar, luego de haber seleccionado los dos jugadores se selecciona el tablero 3x3.	Que el programa permita jugar con el tamaño del tablero seleccionado.	Se cumple lo esperado.
En la opción 2: Jugar, luego de haber seleccionado los dos jugadores se selecciona el tablero 5x5.	Que el programa permita jugar con el tamaño del tablero seleccionado.	Se cumple lo esperado.
En la opción 2: Jugar, luego de haber seleccionado los dos jugadores se selecciona un numero fuera del rango, por ejemplo 0.	Que el programa no permita jugar y que pida una opción correcta al usuario.	Se cumple lo esperado.
En la opción 2: Jugar, luego de haber seleccionado los dos jugadores se selecciona un número fuera del rango, por ejemplo 3.	Que el programa no permita jugar y que pida una opción correcta al usuario.	Se cumple lo esperado.
Se selecciona la opción 3: Ranking.	Que el programa muestre la lista de jugadores ordenada por cantidad de partidas ganadas, además de la cantidad de partidas empatadas y perdidas.	Se cumple lo esperado.

Se selecciona la opción 3: Ranking, pero sin jugadores previamente registrados.	Que el programa no muestre ninguna lista de jugadores y el usuario vuelva al menú.	Se cumple lo esperado.
Se selecciona la opción 3: Ranking, con jugadores pero que aún no han ganado.	Que el programa diga no aún no hay jugadores rankeados.	Se cumple lo esperado.
Se selecciona la opción 4: Fin. Se ponen opciones dentro del rango.	Que el programa le pregunte al usuario si está seguro. En el caso de que el usuario lo esté el programa se termina. De lo contrario el usuario sigue en el menú.	Se cumple lo esperado.
Se selecciona la opción 4: Fin. Se ponen opciones fuera del rango, en este caso 0.	Que el programa siga pidiendo un comando válido al usuario hasta que lo ingrese.	Se cumple lo esperado.

PRUEBA DE DATOS: PARTIDA

EJEMPLO DE PARTIDA EN TABLERO 3X3

Una vez que el programa valide los requisitos para jugar se puede empezar una partida. En este caso el usuario elige un tablero 3x3.

3	T	T	T
2			A
1	T	T	
	A	B	C

Turno del jugador Azul:
C1 C2

Imagen 1

3	T	T	T
2			A
1	T	T	
	A	B	C

Turno del jugador Rojo:
D1 D2

Imagen 2

3	T		T
2			A
1	T	A	
	A	B	C

Turno del jugador Rojo:
B3 B1

Imagen 3

3	T		T
2			A
1	T	A	
	A	B	C

Turno del jugador Azul:

0

Imagen 4

3	T		T
2			A
1	T	A	
	A	B	C

Turno del jugador Azul:

A1 A2

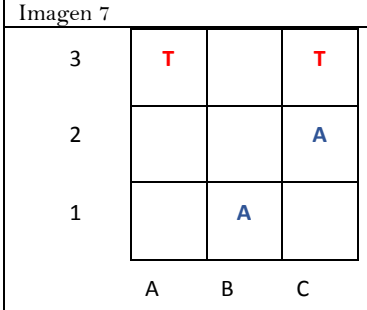
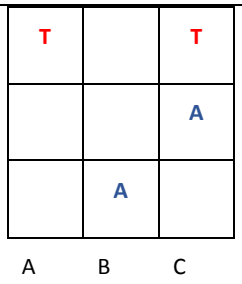
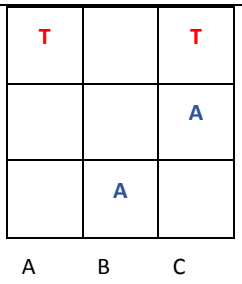
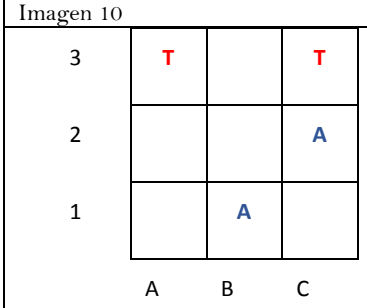
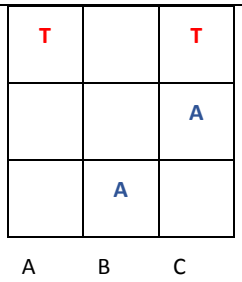
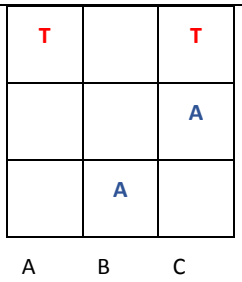
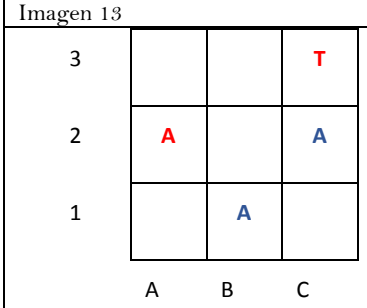
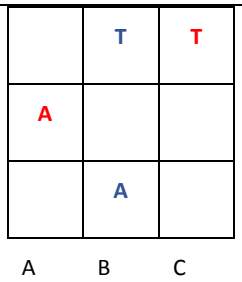
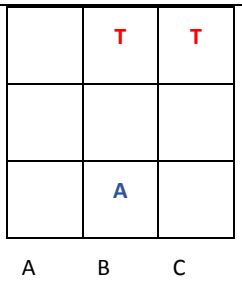
Imagen 5

3	T		T
2			A
1		A	
	A	B	C

Turno del jugador Azul:

A1 B1

Imagen 6

A B C Turno del jugador Rojo: B1 B3 Imagen 7	A B C Turno del jugador Rojo: AA AA Imagen 8	A B C Turno del jugador Rojo: 11 11 Imagen 9
 Turno del jugador Rojo: A3 A1 asasdf Imagen 10	 Turno del jugador Rojo: D Imagen 11	 Turno del jugador Rojo: A3 C1 Imagen 12
 Turno del jugador Rojo: Imagen 13	 Turno del jugador Rojo: B3 B2 Imagen 14	 Turno del jugador Rojo: C3 C1 Imagen 15
 Turno del jugador Rojo: A3 A2 Imagen 16	 Turno del jugador Azul: C2 B3 Imagen 17	 Turno del jugador Rojo: A2 B3 Imagen 18

3 T T 2 T 1 A B C Turno del jugador Azul: B1 C1 Imagen 19	3 T 2 T 1 A A B C Turno del jugador Rojo: B3 B1 Imagen 20	

Se hace	Se espera	Que pasa
Como en la Imagen 1 el jugador mueve su ficha.	Que el programa valide el movimiento del jugador y mueve la ficha, convirtiendo en este caso la torre en alfil.	Se cumple
Como en la Imagen 2 el jugador ingresa una jugada inválida que excede los límites del tablero.	Que el programa no valide el movimiento y que lo pida nuevamente.	Se cumple
Como en la Imagen 3 el jugador mueve su torre hacia el arco de su oponente (ataque).	Que el programa valide el movimiento de ataque, y que su torre pase a ser alfil.	Se cumple
Como en la Imagen 4 el jugador ingresa un 0.	Que el programa no valide el movimiento y que le pida ingresar nuevamente al jugador.	Se cumple
Como en la Imagen 5 el jugador ingresa un movimiento mientras su arco esta capturado.	Que el programa no valide el movimiento y que le pida ingresar nuevamente al jugador ya que debe defender su arco por reglas de jugadas.	Se cumple
Como en la Imagen 6 el jugador defiende su arco.	Que le programa valide el movimiento y que su torre pase a ser un alfil.	Se cumple
Como en la Imagen 7 el jugador ingresa un movimiento intentando tomar una ficha contraria.	Que le programa no valide el movimiento ya que no le pertenece esa ficha al jugador de ese turno. Le pide nuevamente un movimiento.	Se cumple
Como en la Imagen 8 el jugador ingresa solo letras en su turno.	Que le programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple
Como en la Imagen 9 el jugador ingresa solo números en su turno.	Que el programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple
Como en la Imagen 10 el jugador ingresa un movimiento válido, pero además letras arbitrarias.	Que el programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple

Como en la Imagen 11 el jugador ingresa una letra o un número. En el caso de la letra esta no es ninguna de las posibles para ingresar al submenú (Y, X, E, H o R)	Que el programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple
Como en la Imagen 12 el jugador intenta un movimiento característico de otro tipo de ficha (en este caso siendo torre intenta moverse como alfil).	Que el programa no valide el movimiento ya que la ficha no puede moverse de otra manera que la suya. Le pide nuevamente un movimiento.	Se cumple
Como en la Imagen 13, el jugador deja un espacio en vacío.	Que el programa siga esperando por el comando.	Se cumple
Como en la Imagen 14, el jugador intenta referirse a un espacio vacío.	Que el programa no valide el comando y pide nuevamente un movimiento al jugador.	Se cumple
Como en la Imagen 15, el jugador intenta moverse teniendo a una ficha en su camino.	Que el programa no valide el comando ya que una ficha está en el camino y pida nuevamente un comando.	Se cumple
Como en la Imagen 16, el jugador mueve su ficha (en este caso una torre).	Que el programa valide el movimiento y que cambie la torre por el alfil.	Se cumple
Como en la Imagen 17, el jugado se mueve al arco contrario (en este caso un alfil).	Que el programa valide el movimiento y que cambie el alfil por la torre.	Se cumple
Como en la Imagen 18, el jugador defiende su arco.	Que el programa valide el movimiento, es obligatorio defender el arco propio.	Se cumple
Como en la Imagen 19, el jugador azul, se mueve hacia arriba a la derecha.	Que el programa valide el movimiento y que cambie el alfil por la torre.	Se cumple

EJEMPLO DE PARTIDA EN TABLERO 5X5

Una vez que el programa valide los requisitos para jugar se puede empezar una partida. En este caso el usuario elige un tablero 5x5.

Se hace	Se espera	Que pasa
El jugador mueve su torre (E1 E4).	Que la torre se mueva a su lugar de destino.	Se cumple lo esperado.
El jugador intenta comer una ficha de su oponente que no está en un arco (A5 A1)	Que el programa no valide el movimiento y que lo pida nuevamente.	Se cumple lo esperado.
El jugador ingresa una jugada inválida que excede los límites del tablero (A1 A6).	Que el programa no valide el movimiento y que lo pida nuevamente.	Se cumple lo esperado.
El jugador mueve su torre hacia el arco de su oponente (C5 C1)	Que el programa valide el movimiento de ataque, y que su torre pase a ser alfil.	Se cumple lo esperado.
El jugador ingresa un 0.	Que el programa no valide el movimiento y que le pida ingresar nuevamente al jugador.	Se cumple lo esperado.

El jugador ingresa un movimiento mientras su arco esta capturado (Jugador Azul: A1 A5).	Que el programa no valide el movimiento y que le pida ingresar nuevamente al jugador ya que debe defender su arco por reglas de jugadas.	Se cumple lo esperado.
El jugador defiende su arco (Jugador Azul: B1 C1).	Que le programa valide el movimiento y que su torre pase a ser un alfil.	Se cumple lo esperado.
El jugador ingresa un movimiento intentando tomar una ficha contraria (Jugador Rojo: A1 A5).	Que le programa no valide el movimiento ya que no le pertenece esa ficha al jugador de ese turno. Le pide nuevamente un movimiento.	Se cumple lo esperado.
El jugador ingresa solo letras en su turno (en este caso ingresa 'hola')	Que le programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple lo esperado.
El jugador ingresa solo números en su turno (en este caso '123').	Que el programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple lo esperado.
El jugador ingresa un movimiento válido, pero además letras arbitrarias (en este caso 'B5 B1 asd').	Que el programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple lo esperado.
El jugador ingresa una letra o un número. La letra esta no es ninguna de las posibles para ingresar al submenú (Y,X,E, H o R), en este caso ('A')	Que el programa no valide el movimiento ya que no representa ningún lugar en el tablero. Le pide nuevamente un movimiento.	Se cumple lo esperado.
El jugador intenta un movimiento característico de otro tipo de ficha (en este caso siendo torre intenta moverse como alfil Jugador Rojo: A5 E1).	Que el programa no valide el movimiento ya que la ficha no puede moverse de otra manera que la suya. Le pide nuevamente un movimiento.	Se cumple lo esperado.
El jugador deja un espacio en vacío.	Que el programa siga esperando por el comando.	Se cumple lo esperado.
El jugador intenta referirse a un espacio vacío (C5 C4).	Que el programa no valide el comando y pide nuevamente un movimiento al jugador.	Se cumple lo esperado.
El jugador intenta moverse teniendo a una ficha en su camino (E5 E3).	Que el programa no valide el comando ya que una ficha está en el camino y pida nuevamente un comando.	Se cumple lo esperado.

GANAR, PERDER Y EMPATAR

Los jugadores deberán procurar ganar, para ello existen diferentes condiciones en las cuales los jugadores pierden, ganan o empatan. Estas sirven para ambos tableros.

Se hace	Se espera	Que pasa
En una partida el jugador escribe el comando “X” para abandonar.	Que el programa le pregunte al jugador si está seguro. En caso de que el jugador lo esté, que permita al jugador abandonar y que este pierda el partido, mientras su oponente gana. En el caso de que el jugador no esté seguro la partida prosigue	Se cumple lo esperado.
En una partida el jugador se queda sin movimientos. Por ejemplo, al tener capturado el arco y no poder defenderse.	Que el programa interprete que dicho jugador perdió, mientras que su oponente gana.	Se cumple lo esperado.
En una partida el jugador pide empate.	Si su oponente acepta entonces ambos empatan.	Se cumple lo esperado.

OPCIONES VÁLIDAS PARA AMBOS TABLEROS

En el juego existen comandos que pueden ingresarse por diferentes motivos. Dichas opciones se encuentran en ambos tableros y se comportan de la misma manera.

Se hace	Se espera	Que pasa
En una partida se ingresa “E” para pedir empate. El jugador contrario tiene dos posibilidades: a) Lo acepta. b) No lo acepta.	Que el programa le pregunte al otro jugador si quiere terminar la partida en un empate. Si: a) La partida termina y se queda en empate, se sale al menú. b) La partida prosigue en el mismo turno en el que se dejó	a) Se cumple lo esperado. b) Se cumple lo esperado.
En una partida se ingresa “X” para abandonar.	Que el programa le pregunte al jugador si está seguro. En caso de que el jugador lo esté, que permita al jugador abandonar y que este pierda el partido, mientras su oponente gana. En el caso de que el jugador no esté seguro la partida prosigue	Se cumple lo esperado.
En una partida se ingresa “Y” para obtener ayuda.	Que el programa muestre una lista de jugadas posibles. Primero deben aparecer las jugadas de defensa, luego las de ataque al arco contrario y hasta 5 jugadas posibles de otras fichas propias del jugador. En el caso de que no haya jugadas disponibles se le avisa al usuario.	Se cumple lo esperado.
En una partida se ingresa “R” para rotar la matriz.	Que el programa rote el tablero 180 grados y que se mantenga por los turnos.	Se cumple lo esperado.

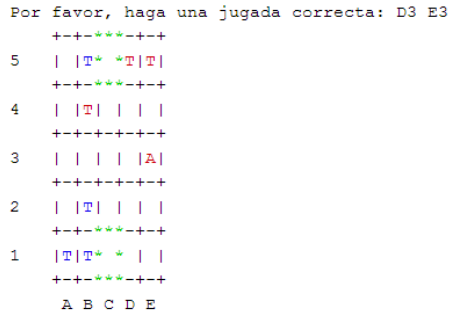
En una partida se ingresa “H” para mostrar el historial.	Que el programa muestre los movimientos anteriores de cada jugador. Si no hay ningún movimiento aún que se muestre que no hay movimientos en el historial.	Se cumple lo esperado.

OPCIONES VÁLIDAS PARA LAS FICHAS

Como no es posible corroborar todos los movimientos de ambas fichas en una sola partida a continuación se contemplarán estos por separado.

Se hace	Se espera	Que pasa
Se mueve un alfil en la diagonal hacia abajo a la derecha.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada: C2 D1 +-+-***-+-+ 5 | |T* *T|T| +-+-***-+-+ 4 | | | | | | +-+-+-+-+-+ 3 | |A| | |A| +-+-+-+-+-+ 2 | | | | | | +-+-***-+-+ 1 |T|T* *T| | +-+-***-+-+ A B C D E
Se mueve un alfil en la diagonal hacia arriba a la derecha.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada: A3 B4 +-+-***-+-+ 5 | | * *T|T| +-+-***-+-+ 4 | |T|A| | | +-+-+-+-+-+ 3 | | | |T| | +-+-+-+-+-+ 2 | |T| | | | +-+-***-+-+ 1 |T|T* * | | +-+-***-+-+ A B C D E
Se mueve un alfil en la diagonal hacia abajo a la izquierda.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada: D2 C1 +-+-***-+-+ 5 |T|T* * |T| +-+-***-+-+ 4 |A|A| | | | +-+-+-+-+-+ 3 | | | | | | +-+-+-+-+-+ 2 | | | | | | +-+-***-+-+ 1 | | *T* |T| +-+-***-+-+ A B C D E

Se mueve un alfil en la diagonal hacia arriba a la izquierda.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada correcta: C4 B5 +-+-***-+-+ 5 | |T* *T|T| +-+-***-+-+ 4 | |T| | | | +-+-+-+-+-+ 3 | | | |T| | +-+-+-+-+-+ 2 | |T| | | | +-+-***-+-+ 1 |T|T* * | | +-+-***-+-+ A B C D E
Se mueve una torre de abajo hacia arriba.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada: B1 B4 +-+-***-+-+ 5 |T|T* * |T| +-+-***-+-+ 4 |A|A| | | | +-+-+-+-+-+ 3 | | | | | | +-+-+-+-+-+ 2 | | | |A| | +-+-***-+-+ 1 | | *A* |T| +-+-***-+-+ A B C D E
Se mueve una torre de arriba hacia abajo.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada correcta: A5 A3 +-+-***-+-+ 5 | |T* *T|T| +-+-***-+-+ 4 | | | | |A| +-+-+-+-+-+ 3 |A| | | | | +-+-+-+-+-+ 2 | | | | | | +-+-***-+-+ 1 |T| *A*T| | +-+-***-+-+ A B C D E
Se mueve una torre de derecha a izquierda.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. Por favor, haga una jugada: E1 C1 +-+-***-+-+ 5 |T|T* * |T| +-+-***-+-+ 4 |A|A| | | | +-+-+-+-+-+ 3 | | | | | | +-+-+-+-+-+ 2 | | | | | | +-+-***-+-+ 1 | | *A* | | +-+-***-+-+ A B C D E

Se mueve una torre de izquierda a derecha.	Que se mueva en la dirección que se marcó.	Se cumple lo esperado. 

CASOS DE USO

1. Título: Registro de Jugador
   Actor: Usuario

   Curso Normal:

   1. Se selecciona la opción 2 del menú
   2. Se ingresa nombre del jugador
   3. Se ingresa alias del jugador
   4. Se ingresa edad del jugador
   5. Se vuelve al menú

   Curso Alternativo:

   3.1. el alias ya está registrado en el programa (debe de ser único)

   4.1 se ingresa una edad incorrecta

2. Título: Ranking
   Actor: Usuario

   Curso Normal:

   1. Se selecciona opción 3 del menú
   2. Se imprime una lista de jugadores ordenados por partidas ganadas, además de empates y partidas perdidas
   3. Se vuelve al menú

   Curso Alternativo:

   2.1. No hay jugadores ingresados. Fin CU

COMPARACION CON COMPETENCIA

Se presentarán dos apps para la comparación de nuestro programa; primero el juego *Damas* y luego el juego *Reversi*. Ambos comparten características similares a *Inversiones*, las cuales se analizarán a continuación.

Para empezar los juegos *Reversi* y *Damas* son juegos tradicionales, por lo que son reconocidos mundialmente o bien, populares en ciertas culturas. Mientras que *Inversiones* no presenta popularidad en ninguna cultura, esto podría implicar que los jugadores tengan que realizar un esfuerzo mayor para aprender a jugar.
Sin embargo estos juegos comparten cualidades muy parecidas entre ellos, aunque con leves diferencias. Un ejemplo de esto son los tableros, los cuales si bien en *Reversi* y en *Damas* son definidos (no se pueden elegir las dimensiones), todos los juegos se ven limitados por estos entornos físicos, los cuales definen una parte importante del juego. Por otra parte, dichos juegos presentan turnos entre los jugadores, sin embargo, en *Reversi* si el jugador no puede mover ninguna ficha está obligado a ceder su turno, mientras que en los demás juegos eso no es posible. Otro pilar de estos juegos es el comportamiento de las fichas. En *Inversiones* los movimientos del alfil son similares a los de las fichas de *Damas*, mientras que en el *Reversi* las fichas pueden posicionarse en casi cualquier lugar, sin reglas de movimientos más que las de poder crear una línea recta para poder 'comer' las fichas del oponente. Por último, se propone analizar la cantidad de fichas que obtiene cada jugador. En *Reversi* cada jugador empieza su partida con solo dos, pero a medida que la jugada avanza se suman las fichas de cada jugador, llegando a tener hasta 64 fichas. En *Damas* e *Inversiones* las fichas disminuyen ya que son 'comidas' por el oponente hasta no tener ninguna más.

Fuente: https://www.coolmath-games.com/0-reversi
http://www.ludoteka.com/juegos?juego=damas-espa%C3%B1olas&jatorri=gp_ches_wrld2&gclid=Cj0KCQjwprbPBRCHARIsAF_7gDaPlXSgddtUr3dRQjP74WQS7ZmRiqDQC3zjxr7-rQ5ySbeSXG2VSQUaAn9sEALw_wcB

EVIDENCIA DE TESTING

```
== Turno del jugador Rojo ==
Por favor, haga una jugada: B5 C5
    +-+-***-+-+
5   | | *A*T|T|
    +-+-***-+-+
4   | | | | | |
    +-+-+-+-+-+
3   | | | | | |
    +-+-+-+-+-+
2   | |T| | | |
    +-+-***-+-+
1   |T| *A* |T|
    +-+-***-+-+
     A B C D E
```

CÓDIGO DEL PROGRAMA

FICHA

```
package obligatoriop2;

public class Ficha {

   private boolean esAlfil;
   private boolean esAzul;
   private int fila;
   private int columna;
   private Jugador jugador;

   public void setEsAlfil(boolean unIndicador) {
      this.esAlfil = unIndicador;
   }
   public void setFila(int unaFila) {
      this.fila = unaFila;
   }
   public void setColumna(int unaColumna) {
      this.columna = unaColumna;
   }
   public void setJugador(Jugador unJugador){
      this.jugador = unJugador;
   }
      public void setEsAzul(boolean unIndicadorColor){
      this.esAzul = unIndicadorColor;
   }


   public int getFila() {
      return this.fila;
   }

   public int getColumna() {
      return this.columna;
   }

   public boolean getEsAlfil() {
      return this.esAlfil;
   }

   public boolean getEsAzul(){
      return this.esAzul;
   }

   public Jugador getJugador(){
      return this.jugador;
   }

   public char getEstado() {
      if (this.esAlfil) {
         return 'A';
      } else {
         return 'T';
      }
   }
```

```
    public Ficha() {
        this.setEsAlfil(false);
        this.setEsAzul(false);
        this.setFila(0);
        this.setColumna(0);
        this.setJugador(null);
    }

    public Ficha(boolean unIndicador, boolean unIndicadorColor, int unaFila, int unaColumna, Jugador unJugador) {
        this.setEsAlfil(unIndicador);
        this.setEsAzul(unIndicadorColor);
        this.setFila(unaFila);
        this.setColumna(unaColumna);
        this.setJugador(unJugador);
    }

    @Override
    public String toString() {
        //Azul- reset color
        if (this.esAzul) {
            if (this.esAlfil) {
                return "\u001B[34mA\u001B[0m";
            } else {
                return "\u001B[34mT\u001B[0m";
            }
        } //Rojo- reset color
        else {
            if (this.esAlfil) {
                return "\u001B[31mA\u001B[0m";
            } else {
                return "\u001B[31mT\u001B[0m";
            }
        }
    }

}
```

JUGADOR

```
package obligatoriop2;

public class Jugador implements Comparable<Jugador> {

    private String nombre;
    private String alias;
    private boolean esAzul;
    private int partidasGanadas;
    private int partidasEmpatadas;
    private int partidasPerdidas;
    private int edad;

    public void setNombre(String unNombre) {
        this.nombre = unNombre;
    }

    public void setAlias(String unAlias) {
        this.alias = unAlias;
    }

    public void setEdad(int unaEdad) {
        this.edad = unaEdad;
    }
```

```
public void setPartidasGanadas(int unaPartidaGanada) {
   this.partidasGanadas = unaPartidaGanada;
}

public void setPartidasEmpatadas(int unaPartidaEmpatada) {
   this.partidasEmpatadas = unaPartidaEmpatada;
}

public void setPartidasPerdidas(int unaPartidaPerdida) {
   this.partidasPerdidas = unaPartidaPerdida;
}

public void setEsAzul(boolean unIndicadorColor){
   this.esAzul = unIndicadorColor;
}



public String getNombre() {
   return this.nombre;
}

public String getAlias() {
   return this.alias;
}

public int getEdad() {
   return this.edad;
}

public int getPartidasGanadas() {
   return this.partidasGanadas;
}

public int getPartidasEmpatadas() {
   return this.partidasEmpatadas;
}

public int getPartidasPerdidas() {
   return this.partidasPerdidas;
}

public boolean getEsAzul(){
   return this.esAzul;
}



public Jugador(String unNombre, String unAlias, int unaEdad) {
   this.setNombre(unNombre);
   this.setAlias(unAlias);
   this.setEdad(unaEdad);
   this.setEsAzul(false);
   this.setPartidasPerdidas(0);
   this.setPartidasEmpatadas(0);
   this.setPartidasGanadas(0);

}
```

```
public Jugador() {
      this.setNombre(" ");
      this.setAlias(" ");
      this.setEdad(0);
      this.setEsAzul(false);
      this.setPartidasPerdidas(0);
      this.setPartidasEmpatadas(0);
      this.setPartidasGanadas(0);
   }



   @Override
   public boolean equals(Object obj) {
      return this.getAlias().equals(((Jugador) obj).getAlias());
   }

   public int compareTo(Jugador obj) {
      int ret = this.getPartidasGanadas() - obj.getPartidasGanadas();
      if (ret == 0) {
         return obj.getPartidasPerdidas() - this.getPartidasPerdidas();
      }
      return ret;
   }

   public String toString() {
      return "Nombre del Jugador: " + this.getNombre() + "\n"
            + "Alias del Jugador: " + this.getAlias() + "\n"
            + "Edad del Jugador: " + this.getEdad() + " años \n"
            + "Partidas ganadas/empatadas/perdidas: "
            + this.getPartidasGanadas() + "/"
            + this.getPartidasEmpatadas() + "/"
            + this.getPartidasPerdidas() + "\n";
   }
}
```

JUEGO

```
package obligatoriop2;

import java.util.ArrayList;
import java.util.Collections;

public class Juego {

   private ArrayList<Jugador> listaJugadores = new ArrayList<Jugador>();

   public ArrayList<Jugador> getListaJugadores(){
      return this.listaJugadores;
   }
   public void agregarJugador(Jugador unJugador) {
      this.listaJugadores.add(unJugador);
   }
   public boolean aliasValido(Jugador unJugador){
      boolean aux = true;
      if (this.listaJugadores.contains(unJugador)) {
         aux = false;
      }
      return aux;
   }
```

```
  public ArrayList<Jugador> ranking() {
    ArrayList<Jugador> listaAux = new ArrayList<Jugador>();
    listaAux = this.listaJugadores;
    Collections.sort(listaAux);
    return listaAux;
  }

}
```

INTERFAZ

```
package obligatoriop2;

import java.util.*;

public class Interfaz {

  private Juego juego;

  public Interfaz(Juego unJuego) {
    this.juego = unJuego;
  }

  public int ingresarNumero(String mensaje) {

    return ingresarNumeroConRango(mensaje, Integer.MAX_VALUE, 0);
  }

  public String ingresarTexto(String mensaje) {
    String txt = null;
    Scanner in = new Scanner(System.in);

    System.out.print(mensaje);
    txt = in.nextLine();

    return txt;

  }

  public int ingresarNumeroConRango(String mensaje, int supRango, int infRango) {
    System.out.print(mensaje);
    Scanner in = new Scanner(System.in);
    int num = 0;
    try {
      num = in.nextInt();
    } catch (InputMismatchException e) {
      System.out.print("Por favor, ingrese un numero: ");
      in.nextLine();
    }
    /*comprobamos que el numero este dentro del rango,
    si no lo está lo pdeimos de nuevo*/

    while (num > supRango || infRango > num) {

      System.out.print(mensaje);
      try {
        num = in.nextInt();
      } catch (InputMismatchException e) {
        System.out.print("Por favor, ingrese un numero: ");
        in.nextLine();
      }
```

```
      // num = ingresarNumeroConRango("Por favor ingrese un nuevo numero: ", supRango, infRango);
      if (num > supRango || infRango > num) {
        System.out.println("El numero ingresado esta fuera del rango. \n");
      }
    }
    return num;
  }

  public void imprimirLista(ArrayList lista) {

    for (int i = 0; i < lista.size(); i++) {
      System.out.println((i + 1) + ") " + lista.get(i));
    }
  }

  public void imprimirRanking(ArrayList<Jugador> listaAux) {
    System.out.println("===-- Ranking de jugadores: --====");
    for (int i = 0; i < listaAux.size(); i++) {
      System.out.println((i + 1) + "º)" + listaAux.get(i));
    }
  }

  public void registroJugador(Juego unJuego) {
    System.out.println("===-- Bienvenido a registro de jugador --===");
    Jugador unJugador = new Jugador();
    unJugador.setNombre(ingresarTexto("--- Por favor, ingrese un nombre: "));
    while (unJugador.getNombre().trim().equals("")) {
      unJugador.setNombre(ingresarTexto("Debe ingresar un nombre: "));
    }

    unJugador.setAlias(ingresarTexto("--- Por favor, ingrese un alias, recuerde que este debe ser unico: "));
    while (unJugador.getAlias().trim().equals("")) {
      unJugador.setAlias(ingresarTexto("Debe ingresar un alias: "));
    }
    while (!unJuego.aliasValido(unJugador)) {
      unJugador.setAlias(ingresarTexto("Este alias ya ha sido tomado, por favor, ingrese uno nuevo: "));
    }
    unJugador.setEdad(ingresarNumero("--- Por favor, ingrese su edad: "));
    unJuego.agregarJugador(unJugador);
  }

  public String mostrarFicha(Ficha[][] unaMatrizFichas, int fila, int columna) {
    if (unaMatrizFichas[fila][columna] == null) {
      return " ";
    } else {
      return unaMatrizFichas[fila][columna].toString();
    }
  }
```

```
public void displayTablero(Ficha[][] unaMatrizFichas) {
    if (unaMatrizFichas.length == 5) {
        String matriz = "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "5   |" + mostrarFicha(unaMatrizFichas, 0, 0) + "|" + mostrarFicha(unaMatrizFichas, 0, 1) + "\u001B[32m*\u001B[0m"
+ mostrarFicha(unaMatrizFichas, 0, 2) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 0, 3) + "|" +
mostrarFicha(unaMatrizFichas, 0, 4) + "| \n"
                + "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "4   |" + mostrarFicha(unaMatrizFichas, 1, 0) + "|" + mostrarFicha(unaMatrizFichas, 1, 1) + "|" +
mostrarFicha(unaMatrizFichas, 1, 2) + "|" + mostrarFicha(unaMatrizFichas, 1, 3) + "|" + mostrarFicha(unaMatrizFichas, 1, 4) + "|
\n"
                + "    +-+-+-+-+-+ \n"
                + "3   |" + mostrarFicha(unaMatrizFichas, 2, 0) + "|" + mostrarFicha(unaMatrizFichas, 2, 1) + "|" +
mostrarFicha(unaMatrizFichas, 2, 2) + "|" + mostrarFicha(unaMatrizFichas, 2, 3) + "|" + mostrarFicha(unaMatrizFichas, 2, 4) + "|
\n"
                + "    +-+-+-+-+-+ \n"
                + "2   |" + mostrarFicha(unaMatrizFichas, 3, 0) + "|" + mostrarFicha(unaMatrizFichas, 3, 1) + "|" +
mostrarFicha(unaMatrizFichas, 3, 2) + "|" + mostrarFicha(unaMatrizFichas, 3, 3) + "|" + mostrarFicha(unaMatrizFichas, 3, 4) + "|
\n"
                + "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "1   |" + mostrarFicha(unaMatrizFichas, 4, 0) + "|" + mostrarFicha(unaMatrizFichas, 4, 1) + "\u001B[32m*\u001B[0m"
+ mostrarFicha(unaMatrizFichas, 4, 2) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 4, 3) + "|" +
mostrarFicha(unaMatrizFichas, 4, 4) + "| \n"
                + "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "     A B C D E  \n";
        System.out.println(matriz);
    } else {
        String matriz = "    +-\u001B[32m***\u001B[0m-+ \n"
                + "3   |" + mostrarFicha(unaMatrizFichas, 0, 0) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 0, 1) +
"\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 0, 2) + "| \n"
                + "    +-\u001B[32m***\u001B[0m-+ \n"
                + "2   |" + mostrarFicha(unaMatrizFichas, 1, 0) + "|" + mostrarFicha(unaMatrizFichas, 1, 1) + "|" +
mostrarFicha(unaMatrizFichas, 1, 2) + "| \n"
                + "    +-\u001B[32m***\u001B[0m-+ \n"
                + "1   |" + mostrarFicha(unaMatrizFichas, 2, 0) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 2, 1) +
"\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 2, 2) + "| \n"
                + "    +-\u001B[32m***\u001B[0m-+ \n"
                + "     A B C  \n";
        System.out.println(matriz);
    }
  }
```

```
public void displayTableroRotado(Ficha[][] unaMatrizFichas) {
    if (unaMatrizFichas.length == 5) {
        String matriz = "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "1   |" + mostrarFicha(unaMatrizFichas, 4, 4) + "|" + mostrarFicha(unaMatrizFichas, 4, 3) + "\u001B[32m*\u001B[0m"
+ mostrarFicha(unaMatrizFichas, 4, 2) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 4, 1) + "|" +
mostrarFicha(unaMatrizFichas, 4, 0) + "| \n"
                + "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "2   |" + mostrarFicha(unaMatrizFichas, 3, 4) + "|" + mostrarFicha(unaMatrizFichas, 3, 3) + "|" +
mostrarFicha(unaMatrizFichas, 3, 2) + "|" + mostrarFicha(unaMatrizFichas, 3, 1) + "|" + mostrarFicha(unaMatrizFichas, 3, 0) + "|
\n"
                + "    +-+-+-+-+-+ \n"
                + "3   |" + mostrarFicha(unaMatrizFichas, 2, 4) + "|" + mostrarFicha(unaMatrizFichas, 2, 3) + "|" +
mostrarFicha(unaMatrizFichas, 2, 2) + "|" + mostrarFicha(unaMatrizFichas, 2, 1) + "|" + mostrarFicha(unaMatrizFichas, 2, 0) + "|
\n"
                + "    +-+-+-+-+-+ \n"
                + "4   |" + mostrarFicha(unaMatrizFichas, 1, 4) + "|" + mostrarFicha(unaMatrizFichas, 1, 3) + "|" +
mostrarFicha(unaMatrizFichas, 1, 2) + "|" + mostrarFicha(unaMatrizFichas, 1, 1) + "|" + mostrarFicha(unaMatrizFichas, 1, 0) + "|
\n"
                + "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "5   |" + mostrarFicha(unaMatrizFichas, 0, 4) + "|" + mostrarFicha(unaMatrizFichas, 0, 3) + "\u001B[32m*\u001B[0m"
+ mostrarFicha(unaMatrizFichas, 0, 2) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 0, 1) + "|" +
mostrarFicha(unaMatrizFichas, 0, 0) + "| \n"
                + "    +-+-\u001B[32m***\u001B[0m-+-+ \n"
                + "     E D C B A  \n";
        System.out.println(matriz);
    } else {
        String matriz = "    +-\u001B[32m***\u001B[0m-+ \n"
                + "1   |" + mostrarFicha(unaMatrizFichas, 2, 2) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 2, 1) +
"\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 2, 0) + "| \n"
                + "    +-\u001B[32m***\u001B[0m-+ \n"
                + "2   |" + mostrarFicha(unaMatrizFichas, 1, 2) + "|" + mostrarFicha(unaMatrizFichas, 1, 1) + "|" +
mostrarFicha(unaMatrizFichas, 1, 0) + "| \n"
                + "    +-\u001B[32m***\u001B[0m-+ \n"
                + "3   |" + mostrarFicha(unaMatrizFichas, 0, 2) + "\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 0, 1) +
"\u001B[32m*\u001B[0m" + mostrarFicha(unaMatrizFichas, 0, 0) + "| \n"
                + "    +-\u001B[32m***\u001B[0m-+ \n"
                + "     C B A  \n";
        System.out.println(matriz);
    }
  }

  public void displayMenu() {
    String menu = "\n"
          + "===-- Menu --=== \n"
          + " \n"
          + "1 -   Registro de Jugador \n"
          + "2 -   Jugar \n"
          + "3 -   Ranking \n"
          + "4 -   Salir \n";

    System.out.println(menu);
  }
```

```
public int[] recibirJugada(Ficha[][] unTablero, String unaJugada) {
      int[] coordenadas = new int[4];
      String jugada = null;

      jugada = unaJugada;

      while (jugada == null || !jugadaValida(unTablero, jugada)) {
         jugada = ingresarTexto("** Por favor, ingrese correctamente la jugada. ** \n"
               + "Recuerde que esta debe tener solo letras mayúsculas, \n "
               + "primero se indica la fila de origen, seguida por la columna de origen, \n"
               + "espacio, fila y columna de destino. \n");
      }
      if (unTablero.length == 3) {
         coordenadas[0] = jugada.charAt(0) - 65;
         coordenadas[1] = numeroCorrespondiente3(jugada.charAt(1) - 49);
         coordenadas[2] = jugada.charAt(3) - 65;
         coordenadas[3] = numeroCorrespondiente3(jugada.charAt(4) - 49);
         return coordenadas;
      } else {
         coordenadas[0] = jugada.charAt(0) - 65;
         coordenadas[1] = numeroCorrespondiente5(jugada.charAt(1) - 49);
         coordenadas[2] = jugada.charAt(3) - 65;
         coordenadas[3] = numeroCorrespondiente5(jugada.charAt(4) - 49);

         return coordenadas;
      }
   }

   public int numeroCorrespondiente5(int unIndice) {
      int indice = unIndice;
      int[] correspondiente = new int[]{4, 3, 2, 1, 0};
      indice = correspondiente[indice];
      return indice;
   }

   public int numeroCorrespondiente3(int unIndice) {
      int indice = unIndice;
      int[] correspondiente = new int[]{2, 1, 0};
      indice = correspondiente[indice];
      return indice;
   }
```

```
public boolean jugadaValida(Ficha[][] unTablero, String jugada) {
      boolean esValida = true;
      if (unTablero.length == 5) {
         if (jugada.length() != 5) {
            esValida = false;
         } else if (jugada.charAt(0) - 65 > 4
               || jugada.charAt(1) - 49 > 4
               || jugada.charAt(3) - 65 > 4
               || jugada.charAt(4) - 49 > 4
               || jugada.charAt(0) - 65 < 0
               || jugada.charAt(1) - 49 < 0
               || jugada.charAt(3) - 65 < 0
               || jugada.charAt(4) - 49 < 0) {
            esValida = false;
         }
      } else {
         if (jugada.length() != 5) {
            esValida = false;
         } else if (jugada.charAt(0) - 65 > 2
               || jugada.charAt(1) - 49 > 2
               || jugada.charAt(3) - 65 > 2
               || jugada.charAt(4) - 49 > 2
               || jugada.charAt(0) - 65 < 0
               || jugada.charAt(1) - 49 < 0
               || jugada.charAt(3) - 65 < 0
               || jugada.charAt(4) - 49 < 0) {
            esValida = false;
         }

      }
      return esValida;
   }

   public void seleccionarJugadores(ArrayList<Jugador> unaListaJugadores, Partida unaPartida) {

      imprimirLista(unaListaJugadores);

      Jugador jAzul = unaListaJugadores.get((ingresarNumeroConRango("-- Elija el jugador Azul por favor: ",
(unaListaJugadores.size()), 1)) - 1);

      imprimirLista(unaListaJugadores);

      Jugador jRojo = unaListaJugadores.get((ingresarNumeroConRango("-- Elija el jugador Rojo por favor: ",
(unaListaJugadores.size()), 1)) - 1);

      while (jAzul.equals(jRojo)) {
         jRojo = unaListaJugadores.get((ingresarNumeroConRango(" No puedes elegir el mismo jugador,"
               + " por favor, selecciona otro: ", (unaListaJugadores.size() - 1), 1)) - 1);

      }

      unaPartida.setJugadorAzul(jAzul);
      unaPartida.getJugadorAzul().setEsAzul(true);
      unaPartida.setJugadorRojo(jRojo);
      unaPartida.getJugadorRojo().setEsAzul(false);
   }
```

```
public void jugarUnaPartida() {
    Partida unaPartida = new Partida();
    String[] opciones = {"e", "x", "y", "r", "h"};
    System.out.println("==-- Seleccion de jugadores: --==");
    seleccionarJugadores(this.juego.getListaJugadores(), unaPartida);
    int opc;
    boolean valido = false;
    while (!valido) {
      opc = ingresarNumero("==-- Seleccione el tamaño de tablero: --== \n"
          + "1) Tablero 5x5 \n"
          + "2) Tablero 3x3 \n");
      switch (opc) {

        case 1:
          unaPartida.crearTablero(5);
          System.out.println("--> Ha seleccionado: tablero 5x5");
          valido = true;
          break;

        case 2:
          unaPartida.crearTablero(3);
          System.out.println("--> Ha seleccionado: tablero 3x3");
          valido = true;
          break;

        default:
          System.out.println("* La opcion no es valida *");
          break;
      }
    }

    //empieza la partida
    boolean termino = false;
    boolean fueRotado = false;
    while (!termino) {
      if (unaPartida.getTurno() == 0 && !fueRotado) {
        displayTablero(unaPartida.getTablero());
      }

      if (unaPartida.getTurno() % 2 == 0) {
        // Turno jugador Azul
        // Se arma la lista de jugadas disponibles antes de pedir la jugada
        unaPartida.jugadasDefensaDisponibles();
        unaPartida.jugadasAtaqueDisponibles();
        unaPartida.otrasJugadasDisponibles();

        //checkea si el jugador se queda sin movimientos
        if (unaPartida.getListaJugadasDefensa().isEmpty()
            && unaPartida.getListaJugadasAtaque().isEmpty()
            && unaPartida.getListaOtrasJugadas().isEmpty()) {
          System.out.println("-- El jugador Azul no tiene mas movimientos. El jugador rojo gana. --");

          unaPartida.getJugadorAzul().setPartidasPerdidas(unaPartida.getJugadorAzul().getPartidasPerdidas() + 1);
          unaPartida.getJugadorRojo().setPartidasGanadas(unaPartida.getJugadorRojo().getPartidasGanadas() + 1);

          termino = true;

        } else if (unaPartida.getListaJugadasDefensa().isEmpty() && unaPartida.getTablero().length == 3
            && unaPartida.getTablero()[2][1] != null
            && !unaPartida.getTablero()[2][1].getEsAzul()) {
          System.out.println("-- El jugador Azul no puede defender su arco. El jugador rojo gana. --");
          unaPartida.getJugadorAzul().setPartidasPerdidas(unaPartida.getJugadorAzul().getPartidasPerdidas() + 1);
```

```
            unaPartida.getJugadorRojo().setPartidasGanadas(unaPartida.getJugadorRojo().getPartidasGanadas() + 1);
            termino = true;

        } else if (unaPartida.getListaJugadasDefensa().isEmpty() && unaPartida.getTablero().length == 5
                && unaPartida.getTablero()[4][2] != null
                && !unaPartida.getTablero()[4][2].getEsAzul()) {
            System.out.println("-- El jugador Azul no puede defender su arco. El jugador rojo gana. --");
            unaPartida.getJugadorAzul().setPartidasPerdidas(unaPartida.getJugadorAzul().getPartidasPerdidas() + 1);
            unaPartida.getJugadorRojo().setPartidasGanadas(unaPartida.getJugadorRojo().getPartidasGanadas() + 1);
            termino = true;

        } else {
            System.out.println("== Turno del jugador Azul ==");

            String unaJugada = ingresarTexto("Por favor, haga una jugada: ");

            boolean esta = false;
            for (int i = 0; i < opciones.length; i++) {
                if (opciones[i].equals(unaJugada.trim().toLowerCase())) {
                    esta = true;
                }
            }
            if (esta) {
                int opc1 = Arrays.asList(opciones).indexOf(unaJugada.trim().toLowerCase());
                switch (opc1) {

                    case 0:
                        boolean abandonar = false;
                        while (!abandonar) {
                            int opc2 = ingresarNumero("--- Jugador Rojo: ¿Acepta el empate? --- \n"
                                    + "Si lo hace, no le sera acreditada ninguna partida ganada. \n"
                                    + "1) Si \n"
                                    + "2) No \n");
                            switch (opc2) {
                                case 1:
                                    abandonar = true;
                                    termino = true;
                                    unaPartida.getJugadorAzul().setPartidasEmpatadas(unaPartida.getJugadorAzul()
                                                                                    .getPartidasEmpatadas() + 1);
                                    unaPartida.getJugadorRojo().setPartidasEmpatadas(unaPartida.getJugadorRojo()
                                                                                    .getPartidasEmpatadas() + 1);
                                    break;
                                case 2:
                                    abandonar = true;
                                    break;
                                default:
                                    System.out.println("* La opcion no es valida. *");
                                    break;
                            }

                        }
                        break;

                    case 1:
                        boolean abandonar1 = false;
                        while (!abandonar1) {
                            int opc2 = ingresarNumero("--- ¿Esta seguro de que quiere abandonar la partida? --- \n"
                                    + "Si lo hace perdera el juego. \n"
                                    + "1) Si \n"
                                    + "2) No \n");
                            switch (opc2) {
                                case 1:
```

```
                    abandonar1 = true;
                    termino = true;
                    unaPartida.getJugadorAzul().setPartidasPerdidas(unaPartida.getJugadorAzul().getPartidasPerdidas() + 1);
                    unaPartida.getJugadorRojo().setPartidasGanadas(unaPartida.getJugadorRojo().getPartidasGanadas() + 1);
                    break;
                  case 2:
                    abandonar1 = true;
                    break;
                  default:
                    System.out.println("* La opcion no es valida. *");
                    break;
                }

              }
              break;

            case 2:
              if (unaPartida.getListaJugadasDefensa().isEmpty()) {
                System.out.println("* No hay movimientos de defensa. *");
              } else {
                System.out.println("--- Lista de posibles jugadas defensivas ---");
                imprimirLista(unaPartida.getListaJugadasDefensa());
              }
              if (unaPartida.getListaJugadasAtaque().isEmpty()) {
                System.out.println("* No hay movimientos de ataque disponibles. *");
              } else {
                System.out.println("--- Lista de posibles jugadas ofensivas ---");
                imprimirLista(unaPartida.getListaJugadasAtaque());
              }
              if (unaPartida.getListaOtrasJugadas().isEmpty()) {
                System.out.println("* No hay otros movimientos disponibles. *");
              } else {
                System.out.println("--- Lista de otras jugadas posibles ---");
                imprimirLista(unaPartida.getListaOtrasJugadas());
              }
              break;

            case 3:
              fueRotado = true;
              unaPartida.setTableroRotado(!unaPartida.getTableroRotado());
              if (unaPartida.getTableroRotado()) {
                displayTableroRotado(unaPartida.getTablero());
              }
              if (!unaPartida.getTableroRotado()) {
                displayTablero(unaPartida.getTablero());
              }

              break;

            case 4:
              if (unaPartida.getHistorialJugadas().isEmpty()) {
                System.out.println("* Todavia no se han hecho jugadas. *");
              } else {
                System.out.println("=== Historial de jugadas de la Partida ===");
                imprimirLista(unaPartida.getHistorialJugadas());
              }
              break;
          }

        } else {
          int[] coordAux = recibirJugada(unaPartida.getTablero(), unaJugada);
```

```
            boolean debeDefender = false;

            if (unaPartida.getTablero().length == 3 && !unaPartida.getListaJugadasDefensa().isEmpty() && (coordAux[2] != 1 ||
coordAux[3] != 2)) {
              debeDefender = true;
            }
            if (unaPartida.getTablero().length == 5 && !unaPartida.getListaJugadasDefensa().isEmpty() && (coordAux[2] != 2 ||
coordAux[3] != 4)) {
              debeDefender = true;
            }

            while (!unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                unaPartida.getJugadorAzul(), coordAux, unaPartida.getTablero()) || debeDefender) {

              if (!unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                  unaPartida.getJugadorAzul(), coordAux, unaPartida.getTablero())) {
                System.out.println("* El movimiento no es valido. *");
                coordAux = recibirJugada(unaPartida.getTablero(), ingresarTexto("Por favor, haga una jugada correcta: "));
              } else if (debeDefender) {
                System.out.println("* Debe defender su arco. *");
                coordAux = recibirJugada(unaPartida.getTablero(), ingresarTexto("Por favor, haga una jugada correcta: "));

                if (unaPartida.getTablero().length == 3 &&
unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                    unaPartida.getJugadorAzul(), coordAux, unaPartida.getTablero()) && coordAux[2] == 1
                    && coordAux[3] == 2) {
                  debeDefender = false;
                }
                if (unaPartida.getTablero().length == 5
                    && unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                    unaPartida.getJugadorAzul(), coordAux, unaPartida.getTablero()) && coordAux[2] == 2
                    && coordAux[3] == 4) {
                  debeDefender = false;
                }

              }

            }

            unaPartida.mover(coordAux);

            if (unaPartida.getTableroRotado()) {
              displayTableroRotado(unaPartida.getTablero());
            } else {
              displayTablero(unaPartida.getTablero());
            }
          }

        }

      } else {
        //turno de jugador rojo
        unaPartida.jugadasDefensaDisponibles();
        unaPartida.jugadasAtaqueDisponibles();
        unaPartida.otrasJugadasDisponibles();

        //checkea si el jugador se queda sin movimientos
        if (unaPartida.getListaJugadasDefensa().isEmpty()
            && unaPartida.getListaJugadasAtaque().isEmpty()
            && unaPartida.getListaOtrasJugadas().isEmpty()) {
          System.out.println("-- El jugador Rojo no tiene mas movimientos. El jugador Azul gana. --");
```

```
    unaPartida.getJugadorAzul().setPartidasGanadas(unaPartida.getJugadorAzul().getPartidasGanadas() + 1);
    unaPartida.getJugadorRojo().setPartidasPerdidas(unaPartida.getJugadorRojo().getPartidasPerdidas() + 1);

    termino = true;

} else if (unaPartida.getListaJugadasDefensa().isEmpty() && unaPartida.getTablero().length == 3
        && unaPartida.getTablero()[0][1] != null
        && unaPartida.getTablero()[0][1].getEsAzul()) {
    System.out.println("-- El jugador Rojo no puede defender su arco. El jugador Azul gana. --");
    unaPartida.getJugadorAzul().setPartidasGanadas(unaPartida.getJugadorAzul().getPartidasGanadas() + 1);
    unaPartida.getJugadorRojo().setPartidasPerdidas(unaPartida.getJugadorRojo().getPartidasPerdidas() + 1);
    termino = true;

} else if (unaPartida.getListaJugadasDefensa().isEmpty() && unaPartida.getTablero().length == 5
        && unaPartida.getTablero()[0][2] != null
        && unaPartida.getTablero()[0][2].getEsAzul()) {
    System.out.println("-- El jugador Rojo no puede defender su arco. El jugador Azul gana. --");
    unaPartida.getJugadorAzul().setPartidasGanadas(unaPartida.getJugadorAzul().getPartidasGanadas() + 1);
    unaPartida.getJugadorRojo().setPartidasPerdidas(unaPartida.getJugadorRojo().getPartidasPerdidas() + 1);
    termino = true;
} else {
    System.out.println("== Turno del jugador Rojo ==");

    String unaJugada = ingresarTexto("Por favor, haga una jugada: ");

    boolean esta = false;
    for (int i = 0; i < opciones.length; i++) {
        if (opciones[i].equals(unaJugada.trim().toLowerCase())) {
            esta = true;
        }
    }
    if (esta) {
        int opc1 = Arrays.asList(opciones).indexOf(unaJugada.trim().toLowerCase());
        switch (opc1) {

            case 0:
                boolean abandonar = false;
                while (!abandonar) {
                    int opc2 = ingresarNumero("--- Jugador Azul: ¿Acepta el empate? --- \n"
                            + "Si lo hace, no le sera acreditada ninguna partida ganada. \n"
                            + "1) Si \n"
                            + "2) No \n");
                    switch (opc2) {
                        case 1:
                            abandonar = true;
                            termino = true;
                            unaPartida.getJugadorAzul().setPartidasEmpatadas(unaPartida.getJugadorAzul()
                                    .getPartidasEmpatadas() + 1);
                            unaPartida.getJugadorRojo().setPartidasEmpatadas(unaPartida.getJugadorRojo()
                                    .getPartidasEmpatadas() + 1);
                            break;
                        case 2:
                            abandonar = true;
                            break;
                        default:
                            System.out.println("* La opcion no es valida. * ");
                            break;
                    }

                }
                break;
```

```
case 1:
   boolean abandonar1 = false;
   while (!abandonar1) {
      int opc2 = ingresarNumero("--- ¿Esta seguro de que quiere abandonar la partida? --- \n"
            + "Si lo hace perdera el juego. \n"
            + "1) Si \n"
            + "2) No \n");
      switch (opc2) {
         case 1:
            abandonar1 = true;
            termino = true;
            unaPartida.getJugadorRojo().setPartidasPerdidas(unaPartida.getJugadorRojo().getPartidasPerdidas() + 1);
            unaPartida.getJugadorAzul().setPartidasGanadas(unaPartida.getJugadorAzul().getPartidasGanadas() + 1);
            break;
         case 2:
            abandonar1 = true;
            break;
         default:
            System.out.println("* La opcion no es valida. *");
            break;
      }

   }
   break;

case 2:
   if (unaPartida.getListaJugadasDefensa().isEmpty()) {
      System.out.println("* No hay movimientos de defensa. *");
   } else {
      System.out.println("--- Lista de posibles jugadas defensivas ---");
      imprimirLista(unaPartida.getListaJugadasDefensa());
   }
   if (unaPartida.getListaJugadasAtaque().isEmpty()) {
      System.out.println("* No hay movimientos de ataque disponibles. *");
   } else {
      System.out.println("--- Lista de posibles jugadas ofensivas ---");
      imprimirLista(unaPartida.getListaJugadasAtaque());
   }
   if (unaPartida.getListaOtrasJugadas().isEmpty()) {
      System.out.println("* No hay otros movimientos disponibles. *");
   } else {
      System.out.println("--- Lista de otras jugadas posibles ---");
      imprimirLista(unaPartida.getListaOtrasJugadas());
   }
   break;

case 3:

   unaPartida.setTableroRotado(!unaPartida.getTableroRotado());
   if (unaPartida.getTableroRotado()) {
      displayTableroRotado(unaPartida.getTablero());
   }
   if (!unaPartida.getTableroRotado()) {
      displayTablero(unaPartida.getTablero());
   }

   break;

case 4:
   if (unaPartida.getHistorialJugadas().isEmpty()) {
      System.out.println("* Todavia no se han hecho jugadas. *");
   } else {
```

```
                    System.out.println("=== Historial de jugadas de la Partida === ");
                    imprimirLista(unaPartida.getHistorialJugadas());
                }
                break;
            }
        } else {
            int[] coordAux = recibirJugada(unaPartida.getTablero(), unaJugada);
            boolean debeDefender = false;

            if (unaPartida.getTablero().length == 3 && !unaPartida.getListaJugadasDefensa().isEmpty()
                    && coordAux[2] != 1 && coordAux[3] != 0) {
                debeDefender = true;
            }
            if (unaPartida.getTablero().length == 5 && !unaPartida.getListaJugadasDefensa().isEmpty()
                    && coordAux[2] != 2 && coordAux[3] != 0) {
                debeDefender = true;
            }
            while (!unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                unaPartida.getJugadorRojo(), coordAux, unaPartida.getTablero()) || debeDefender) {

                if (!unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                    unaPartida.getJugadorRojo(), coordAux, unaPartida.getTablero())) {
                    System.out.println("* El movimiento no es valido. *");
                    coordAux = recibirJugada(unaPartida.getTablero(), ingresarTexto("Por favor, haga una jugada correcta: "));
                } else if (debeDefender) {
                    System.out.println("* Debe defender su arco. *");
                    coordAux = recibirJugada(unaPartida.getTablero(), ingresarTexto("Por favor, haga una jugada correcta: "));

                    if (unaPartida.getTablero().length == 3
                        && unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                        unaPartida.getJugadorRojo(), coordAux, unaPartida.getTablero())
                        && (coordAux[2] == 1 || coordAux[3] == 0)) {
                        debeDefender = false;
                    }
                    if (unaPartida.getTablero().length == 5
                        && unaPartida.movimientoValido(unaPartida.getTablero()[coordAux[1]][coordAux[0]],
                        unaPartida.getJugadorRojo(), coordAux, unaPartida.getTablero())
                        && (coordAux[2] == 2 || coordAux[3] == 0)) {
                        debeDefender = false;
                    }

                }

            }
            unaPartida.mover(coordAux);

            if (unaPartida.getTableroRotado()) {
                displayTableroRotado(unaPartida.getTablero());
            } else {
                displayTablero(unaPartida.getTablero());
            }
        }
    }
  }
}

public void Empezar(Juego unJuego) {
  System.out.println("  ===> ¡Bienvenido a Inversiones! <===");
  System.out.println("----------------------------------------");
  displayMenu();
  System.out.println("----------------------------------------");
```

```
int opc1;

boolean quedarse = true;

while (quedarse) {
   System.out.println("");
   opc1 = ingresarNumeroConRango("---> Ingrese el numero de la opcion deseada: ", Integer.MAX_VALUE
                                                                                    , Integer.MIN_VALUE);
   System.out.println("");

   switch (opc1) {

      case 1:
         registroJugador(unJuego);

         break;

      case 2:
         if (unJuego.getListaJugadores().size() >= 2) {
            this.jugarUnaPartida();
         } else {
            System.out.println("Para jugar debe haber al menos dos jugadores registrados.");
         }

         break;

      case 3:
         if (unJuego.getListaJugadores().isEmpty()) {
            System.out.println("- Aun no hay jugadores registrados. -");
         } else {
            boolean noJugadoresRankeados = false;
            for (int i = 0; i < unJuego.getListaJugadores().size(); i++) {
               if (unJuego.getListaJugadores().get(i).getPartidasGanadas() == 0
                     && unJuego.getListaJugadores().get(i).getPartidasPerdidas() == 0
                     && unJuego.getListaJugadores().get(i).getPartidasEmpatadas() == 0) {
                  noJugadoresRankeados = true;
               }
            }
            if (noJugadoresRankeados) {
               System.out.println("- Aun no hay jugadores rankeados. -");
            } else {
               imprimirRanking(unJuego.ranking());
            }
         }

         break;

      case 4:
         boolean quieroSalir = false;
         while (!quieroSalir) {
            int opcion = ingresarNumero("----- ¿Esta seguro de que desea salir? ----- \n"
                  + "1) Si \n"
                  + "2) No \n");
            switch (opcion) {

               case 1:
                  quieroSalir = true;
                  quedarse = false;
                  break;

               case 2:
                  quieroSalir = true;
```

```
                    break;

                default:
                    System.out.println("* La opcion no es valida *");
                    break;
                }
            }
            break;
        default:
            System.out.println("* La opcion no es valida *");
            break;

      }
      if (quedarse) {
        displayMenu();
        System.out.println("--------------------------");
      }
    }
  }
}
```

OBLIGATORIOP2

```
package obligatoriop2;

public class ObligatorioP2 {

  public static void main(String[] args) {
    Juego juego = new Juego();
    Interfaz interfaz = new Interfaz(juego);
    interfaz.Empezar(juego);
  }
}
```

PARTIDA

```
package obligatoriop2;

import java.util.ArrayList;

public class Partida {

  private Ficha[][] tablero;
  private Jugador jugadorRojo;
  private Jugador jugadorAzul;
  private int turno;
  private boolean tableroRotado;
  private ArrayList<String> historialJugadas = new ArrayList<String>();
  private ArrayList<String> listaJugadasDefensa = new ArrayList<String>();
  private ArrayList<String> listaJugadasAtaque = new ArrayList<String>();
  private ArrayList<String> listaOtrasJugadas = new ArrayList<String>();

  public void setJugadorAzul(Jugador unJugadorAzul) {
    this.jugadorAzul = unJugadorAzul;
  }

  public void setJugadorRojo(Jugador unJugadorRojo) {
    this.jugadorRojo = unJugadorRojo;
  }
```

```
public void setTurno(int unTurno) {
      this.turno = unTurno;
   }

   public void setTableroRotado(boolean estaRotado) {
      this.tableroRotado = estaRotado;
   }

   public int getTurno() {
      return this.turno;
   }

   public Jugador getJugadorAzul() {
      return this.jugadorAzul;
   }

   public Jugador getJugadorRojo() {
      return this.jugadorRojo;
   }

   public Ficha[][] getTablero() {
      return this.tablero;
   }

   public boolean getTableroRotado() {
      return this.tableroRotado;
   }

   public ArrayList<String> getListaJugadasDefensa() {
      return this.listaJugadasDefensa;
   }

   public ArrayList<String> getListaJugadasAtaque() {
      return this.listaJugadasAtaque;
   }

   public ArrayList<String> getListaOtrasJugadas() {
      return this.listaOtrasJugadas;
   }

   public ArrayList<String> getHistorialJugadas() {
      return this.historialJugadas;
   }

   public Partida() {
      this.tablero = null;
      this.setJugadorRojo(null);
      this.setJugadorAzul(null);
      this.setTurno(0);
      this.setTableroRotado(false);
   }
```

```
public void jugadasDefensaDisponibles() {
    this.listaJugadasDefensa.clear(); //borra la lista anterior
    //Para tableros 3x3
    if (this.getTablero().length == 3) {
      //Para el jugador Azul
      if (this.getTablero()[2][1] != null && !this.getTablero()[2][1].getEsAzul() && this.getTurno() % 2 == 0) {
        // Si el arco del jugador azul esta ocupado, si esta ocupado por el jugador rojo y es el turno del azul:

        for (int i = 0; i < this.getTablero().length; i++) {
          for (int j = 0; j < this.getTablero().length; j++) {
            if (this.getTablero()[j][i] != null && this.getTablero()[j][i].getEsAzul()) {
              int[] auxCoordenadas = new int[4];
              auxCoordenadas[0] = i;
              auxCoordenadas[1] = j;
              auxCoordenadas[2] = 1;
              auxCoordenadas[3] = 2;
              // auxCoordenadas[0] e [1] son de origen
              //DX=[2]-COLUMNAS
              //DY=[3]-FILAS

              if (movimientoValido(this.getTablero()[j][i], this.getJugadorAzul(),
                  auxCoordenadas, this.getTablero())) {
                //Si el movimiento es valido:

                String a = String.valueOf((char) (auxCoordenadas[0] + 65)); //letra que pasa a Ascii
                String b = String.valueOf((char) (numeroCorrespondiente3(auxCoordenadas[1]) + 49)); // num a
                this.listaJugadasDefensa.add(a + b + " "
                    + ((char) (auxCoordenadas[2] + 65))
                    + ((char) (numeroCorrespondiente3(auxCoordenadas[3]) + 49)));
              }
            }
          }
        }
      }
      //Para el jugador Rojo
      if (this.getTablero()[0][1] != null && this.getTablero()[0][1].getEsAzul() && this.getTurno() % 2 != 0) {

        for (int i = 0; i < this.getTablero().length; i++) {
          for (int j = 0; j < this.getTablero().length; j++) {
            if (this.getTablero()[j][i] != null && !this.getTablero()[j][i].getEsAzul()) {
              int[] auxCoordenadas = new int[4];
              auxCoordenadas[0] = i;
              auxCoordenadas[1] = j;
              auxCoordenadas[2] = 1;
              auxCoordenadas[3] = 0;

              if (movimientoValido(this.getTablero()[j][i], this.getJugadorRojo(),
                  auxCoordenadas, this.getTablero())) {

                String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                String b = String.valueOf((char) (numeroCorrespondiente3(auxCoordenadas[1]) + 49));
                this.listaJugadasDefensa.add(a + b + " "
                    + ((char) (auxCoordenadas[2] + 65))
                    + ((char) (numeroCorrespondiente3(auxCoordenadas[3]) + 49)));
              }
            }
          }
        }
      }
    }
    //Para tablero 5x5
    else {
```

```
        if (this.getTablero().length == 5) {
            //Para el jugador Azul
            if (this.getTablero()[4][2] != null && !this.getTablero()[4][2].getEsAzul() && this.getTurno() % 2 == 0) {
                for (int i = 0; i < this.getTablero().length; i++) {
                    for (int j = 0; j < this.getTablero().length; j++) {
                        if (this.getTablero()[j][i] != null && this.getTablero()[j][i].getEsAzul()) {
                            int[] auxCoordenadas = new int[4];
                            auxCoordenadas[0] = i;
                            auxCoordenadas[1] = j;
                            auxCoordenadas[2] = 2;
                            auxCoordenadas[3] = 4;

                            if (movimientoValido(this.getTablero()[j][i], this.getJugadorAzul(),
                                    auxCoordenadas, this.getTablero())) {

                                String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                                String b = String.valueOf((char) (numeroCorrespondiente5(auxCoordenadas[1]) + 49));
                                this.listaJugadasDefensa.add(a + b + " "
                                        + ((char) (auxCoordenadas[2] + 65))
                                        + ((char) (numeroCorrespondiente5(auxCoordenadas[3]) + 49)));
                            }
                        }
                    }
                }
            }
            //Para el jugador Rojo
            if (this.getTablero()[0][2] != null && this.getTablero()[0][2].getEsAzul() && this.getTurno() % 2 != 0) {
                for (int i = 0; i < this.getTablero().length; i++) {
                    for (int j = 0; j < this.getTablero().length; j++) {
                        if (this.getTablero()[j][i] != null && !this.getTablero()[j][i].getEsAzul()) {
                            int[] auxCoordenadas = new int[4];
                            auxCoordenadas[0] = i;
                            auxCoordenadas[1] = j;
                            auxCoordenadas[2] = 2;
                            auxCoordenadas[3] = 0;

                            if (movimientoValido(this.getTablero()[j][i], this.getJugadorRojo(),
                                    auxCoordenadas, this.getTablero())) {

                                String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                                String b = String.valueOf((char) (numeroCorrespondiente5(auxCoordenadas[1]) + 49));
                                this.listaJugadasDefensa.add(a + b + " "
                                        + ((char) (auxCoordenadas[2] + 65))
                                        + ((char) (numeroCorrespondiente5(auxCoordenadas[3]) + 49)));
                            }
                        }
                    }
                }
            }
        }
    }
}

public void jugadasAtaqueDisponibles() {
    this.listaJugadasAtaque.clear();
    //Para tableros 3x3
    if (this.getTablero().length == 3) {
        //Para el jugador Azul
        if ((this.getTablero()[0][1] == null || !this.getTablero()[0][1].getEsAzul()) && this.getTurno() % 2 == 0) {
            for (int i = 0; i < this.getTablero().length; i++) {
                for (int j = 0; j < this.getTablero().length; j++) {
                    if (this.getTablero()[j][i] != null && this.getTablero()[j][i].getEsAzul()) {
```

```
                int[] auxCoordenadas = new int[4];
                auxCoordenadas[0] = i;
                auxCoordenadas[1] = j;
                auxCoordenadas[2] = 1;
                auxCoordenadas[3] = 0;

                if (movimientoValido(this.getTablero()[j][i], this.getJugadorAzul(),
                    auxCoordenadas, this.getTablero())) {
                  String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                  String b = String.valueOf((char) (numeroCorrespondiente3(auxCoordenadas[1]) + 49));
                  this.listaJugadasAtaque.add(a + b + " "
                      + ((char) (auxCoordenadas[2] + 65))
                      + ((char) (numeroCorrespondiente3(auxCoordenadas[3]) + 49)));
                }
              }
            }
          }
        }
        //Para el jugador Rojo
        if ((this.getTablero()[2][1] == null || this.getTablero()[2][1].getEsAzul()) && this.getTurno() % 2 != 0) {
          for (int i = 0; i < this.getTablero().length; i++) {
            for (int j = 0; j < this.getTablero().length; j++) {
              if (this.getTablero()[j][i] != null && !this.getTablero()[j][i].getEsAzul()) {
                int[] auxCoordenadas = new int[4];
                auxCoordenadas[0] = i;
                auxCoordenadas[1] = j;
                auxCoordenadas[2] = 1;
                auxCoordenadas[3] = 2;

                if (movimientoValido(this.getTablero()[j][i], this.getJugadorRojo(),
                    auxCoordenadas, this.getTablero())) {

                  String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                  String b = String.valueOf((char) (numeroCorrespondiente3(auxCoordenadas[1]) + 49));
                  this.listaJugadasAtaque.add(a + b + " "
                      + ((char) (auxCoordenadas[2] + 65))
                      + ((char) (numeroCorrespondiente3(auxCoordenadas[3]) + 49)));
                }
              }
            }
          }
        }
      } //Para tablero 5x5
      else {
        if (this.getTablero().length == 5) {
          //Para el jugador Azul
          if ((this.getTablero()[0][2] == null || !this.getTablero()[0][2].getEsAzul()) && this.getTurno() % 2 == 0) {
            for (int i = 0; i < this.getTablero().length; i++) {
              for (int j = 0; j < this.getTablero().length; j++) {
                if (this.getTablero()[j][i] != null && this.getTablero()[j][i].getEsAzul()) {
                  int[] auxCoordenadas = new int[4];
                  auxCoordenadas[0] = i;
                  auxCoordenadas[1] = j;
                  auxCoordenadas[2] = 2;
                  auxCoordenadas[3] = 0;

                  if (movimientoValido(this.getTablero()[j][i], this.getJugadorAzul(),
                      auxCoordenadas, this.getTablero())) {

                    String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                    String b = String.valueOf((char) (numeroCorrespondiente5(auxCoordenadas[1]) + 49));
                    this.listaJugadasAtaque.add(a + b + " "
```

```
                        + ((char) (auxCoordenadas[2] + 65))
                        + ((char) (numeroCorrespondiente5(auxCoordenadas[3]) + 49)));
                }
              }
            }
          }
        }
        //Para el jugador Rojo
        if ((this.getTablero()[4][2] != null && this.getTablero()[4][2].getEsAzul()) && this.getTurno() % 2 != 0) {
          for (int i = 0; i < this.getTablero().length; i++) {
            for (int j = 0; j < this.getTablero().length; j++) {
              if (this.getTablero()[j][i] != null && !this.getTablero()[j][i].getEsAzul()) {
                int[] auxCoordenadas = new int[4];
                auxCoordenadas[0] = i;
                auxCoordenadas[1] = j;
                auxCoordenadas[2] = 2;
                auxCoordenadas[3] = 4;

                if (movimientoValido(this.getTablero()[j][i], this.getJugadorRojo(),
                    auxCoordenadas, this.getTablero())) {

                  String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                  String b = String.valueOf((char) (numeroCorrespondiente5(auxCoordenadas[1]) + 49));
                  this.listaJugadasAtaque.add(a + b + " "
                      + ((char) (auxCoordenadas[2] + 65))
                      + ((char) (numeroCorrespondiente5(auxCoordenadas[3]) + 49)));
                }
              }
            }
          }
        }
      }
    }
  }

  public void otrasJugadasDisponibles() {
    this.listaOtrasJugadas.clear();
    //Para tableros 3x3
    if (this.getTablero().length == 3) {
      //Para el jugador Azul
      if (this.getTurno() % 2 == 0) {
        for (int i = 0; i < this.getTablero().length; i++) {
          for (int j = 0; j < this.getTablero().length; j++) {
            if (this.getTablero()[j][i] != null && this.getTablero()[j][i].getEsAzul()) {
              for (int k = 0; k < this.getTablero().length; k++) {
                for (int l = 0; l < this.getTablero().length; l++) {
                  //Recorre el tablero y por cada elemento (espacio en el tablero). Si esta ocupado y
                  // es por una ficha azul entonces recorre los tableros nuevamente.
                  int[] auxCoordenadas = new int[4];
                  auxCoordenadas[0] = i;
                  auxCoordenadas[1] = j;
                  auxCoordenadas[2] = k;
                  auxCoordenadas[3] = l;
                  if (this.getTablero()[l][k] == null
                      && movimientoValido(this.getTablero()[j][i], this.getJugadorAzul(),
                          auxCoordenadas, this.getTablero())
                      && !esArco(l, k, this.getTablero()) && this.listaOtrasJugadas.size() < 6) {
                    //Si el destino esta vacio, el mov es valido y el arco esta desocupado entonces se puede mover
                    // a su destino.
```

```
                        String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                        String b = String.valueOf((char) (numeroCorrespondiente3(auxCoordenadas[1]) + 49));
                        this.listaOtrasJugadas.add(a + b + " "
                                + ((char) (auxCoordenadas[2] + 65))
                                + ((char) (numeroCorrespondiente3(auxCoordenadas[3]) + 49)));
                    }
                }
            }
        }
    }
}
//Para el jugador Rojo
if (this.getTurno() % 2 != 0) {
    for (int i = 0; i < this.getTablero().length; i++) {
        for (int j = 0; j < this.getTablero().length; j++) {
            if (this.getTablero()[j][i] != null && !this.getTablero()[j][i].getEsAzul()) {
                for (int k = 0; k < this.getTablero().length; k++) {
                    for (int l = 0; l < this.getTablero().length; l++) {
                        int[] auxCoordenadas = new int[4];
                        auxCoordenadas[0] = i;
                        auxCoordenadas[1] = j;
                        auxCoordenadas[2] = k; // destino x
                        auxCoordenadas[3] = l; // destino y
                        if (this.getTablero()[l][k] == null
                                && movimientoValido(this.getTablero()[j][i], this.getJugadorRojo(),
                                        auxCoordenadas, this.getTablero())
                                && !esArco(l, k, this.getTablero()) && this.listaOtrasJugadas.size() < 6) {

                            String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                            String b = String.valueOf((char) (numeroCorrespondiente3(auxCoordenadas[1]) + 49));
                            this.listaOtrasJugadas.add(a + b + " "
                                    + ((char) (auxCoordenadas[2] + 65))
                                    + ((char) (numeroCorrespondiente3(auxCoordenadas[3]) + 49)));
                        }
                    }
                }
            }
        }
    }
}//para tablero 5x5
else {
    if (this.getTablero().length == 5) {
        //para el jugador Azul
        if (this.getTurno() % 2 == 0) {
            for (int i = 0; i < this.getTablero().length; i++) {
                for (int j = 0; j < this.getTablero().length; j++) {
                    if (this.getTablero()[j][i] != null && this.getTablero()[j][i].getEsAzul()) {
                        for (int k = 0; k < this.getTablero().length; k++) {
                            for (int l = 0; l < this.getTablero().length; l++) {
                                int[] auxCoordenadas = new int[4];
                                auxCoordenadas[0] = i;
                                auxCoordenadas[1] = j;
                                auxCoordenadas[2] = k;
                                auxCoordenadas[3] = l;
                                if (this.getTablero()[l][k] == null
                                        && movimientoValido(this.getTablero()[j][i], this.getJugadorAzul(),
                                                auxCoordenadas, this.getTablero())
                                        && !esArco(l, k, this.getTablero()) && this.listaOtrasJugadas.size() < 6) {
```

```
                        String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                        String b = String.valueOf((char) (numeroCorrespondiente5(auxCoordenadas[1]) + 49));
                        this.listaOtrasJugadas.add(a + b + " "
                            + ((char) (auxCoordenadas[2] + 65))
                            + ((char) (numeroCorrespondiente5(auxCoordenadas[3]) + 49)));
                    }
                  }
                }
              }
            }
          }
        }
        //para el jugador Rojo
        if (this.getTurno() % 2 != 0) {
          for (int i = 0; i < this.getTablero().length; i++) {
            for (int j = 0; j < this.getTablero().length; j++) {
              if (this.getTablero()[j][i] != null && !this.getTablero()[j][i].getEsAzul()) {
                for (int k = 0; k < this.getTablero().length; k++) {
                  for (int l = 0; l < this.getTablero().length; l++) {
                    int[] auxCoordenadas = new int[4];
                    auxCoordenadas[0] = i;
                    auxCoordenadas[1] = j;
                    auxCoordenadas[2] = k;
                    auxCoordenadas[3] = l;
                    if (this.getTablero()[l][k] == null
                        && movimientoValido(this.getTablero()[j][i], this.getJugadorRojo(),
                            auxCoordenadas, this.getTablero())
                        && !esArco(l, k, this.getTablero()) && this.listaOtrasJugadas.size() < 6) {

                      String a = String.valueOf((char) (auxCoordenadas[0] + 65));
                      String b = String.valueOf((char) (numeroCorrespondiente5(auxCoordenadas[1]) + 49));
                      this.listaOtrasJugadas.add(a + b + " "
                          + ((char) (auxCoordenadas[2] + 65))
                          + ((char) (numeroCorrespondiente5(auxCoordenadas[3]) + 49)));
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  public void crearTablero(int unaDimension) {
    Ficha[][] tableroAux;
    if (unaDimension == 5) {
      this.tablero = new Ficha[5][5];

      for (int j1 = 0; j1 < 5; j1++) {
        this.tablero[0][j1] = new Ficha(false, false, 0, j1, this.getJugadorRojo());
        // 0 y 4 son los arcos
      }
      for (int j2 = 0; j2 < 5; j2++) {
        this.tablero[4][j2] = new Ficha(false, true, 4, j2, this.getJugadorAzul());
      }

    }
```

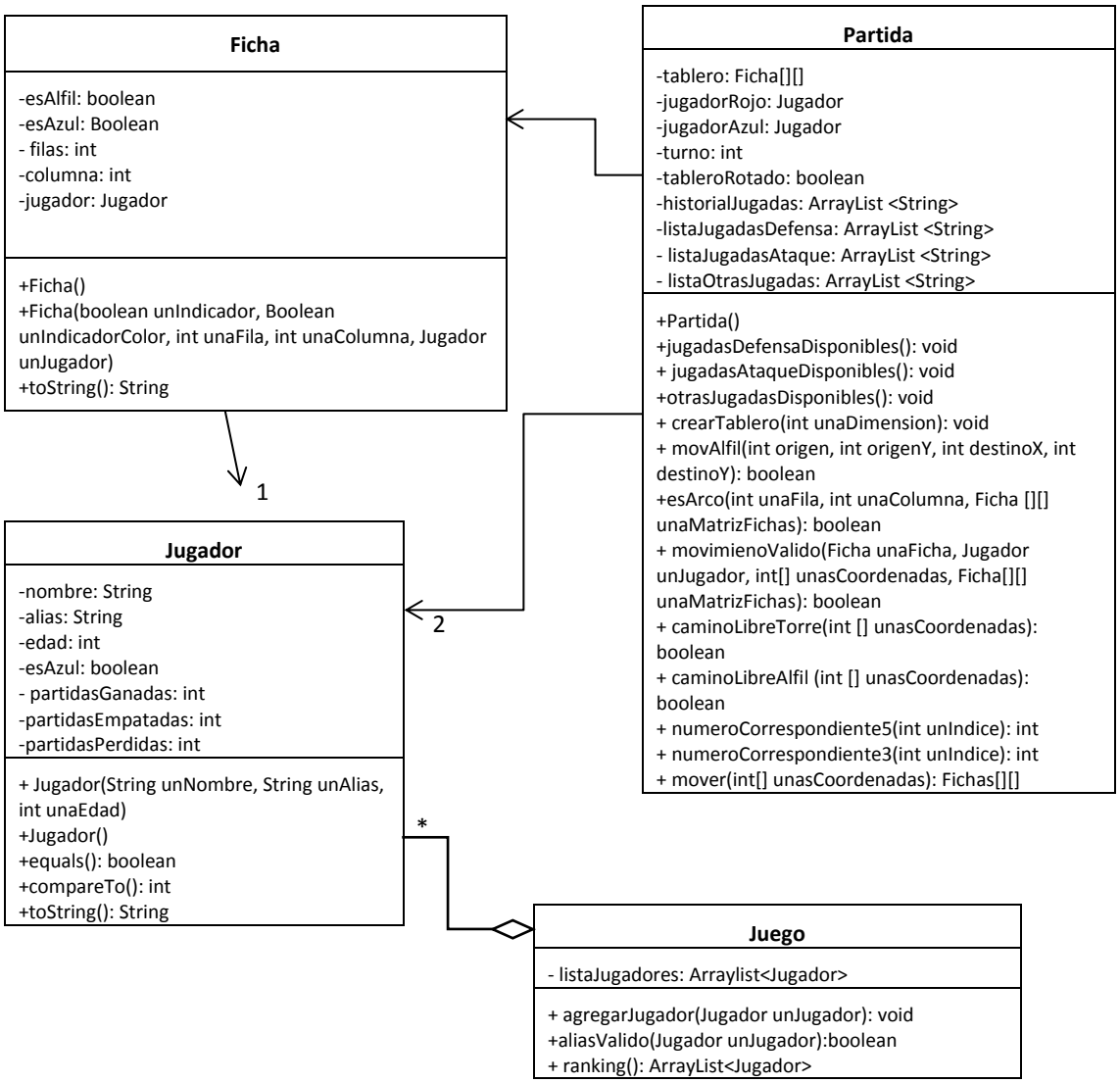
```
else {
        this.tablero = new Ficha[3][3];

        for (int j1 = 0; j1 < 3; j1++) {
          this.tablero[0][j1] = new Ficha(false, false, 0, j1, this.getJugadorRojo());

        }
        for (int j2 = 0; j2 < 3; j2++) {
          this.tablero[2][j2] = new Ficha(false, true, 2, j2, this.getJugadorAzul());
        }
      }
    }

    public boolean movAlfil(int origenX, int origenY, int destinoX, int destinoY) {
      return Math.abs(destinoX - origenX) == Math.abs(destinoY - origenY);
    }

    public boolean esArco(int unaFila, int unaColumna, Ficha[][] unaMatrizFichas) {
      boolean arco = false;
      if (unaMatrizFichas.length == 5) {
        if ((unaFila == 0 || unaFila == 4) && unaColumna == 2) {
          arco = true;
        }
      } else if ((unaFila == 0 || unaFila == 2) && unaColumna == 1) {
        arco = true;
      }
      return arco;
    }

    public boolean movimientoValido(Ficha unaFicha, Jugador unJugador, int[] unasCoordenadas, Ficha[][] unaMatrizFichas) {
      int origenX = unasCoordenadas[0];
      int origenY = unasCoordenadas[1];
      int destinoX = unasCoordenadas[2];
      int destinoY = unasCoordenadas[3];
      boolean esValido = true;
      if (this.getTablero()[origenY][origenX] == null) {
        esValido = false;
      } else {
        if (unaFicha.getEsAlfil() && unaFicha.getJugador().equals(unJugador) && caminoLibreAlfil(unasCoordenadas)) {
    //true = alfil, es del jugador del turno y el camino esta libre
          if (!movAlfil(origenX, origenY, destinoX, destinoY)) {
            esValido = false;
          }
          if (unaMatrizFichas[destinoY][destinoX] != null && !esArco(destinoY, destinoX, unaMatrizFichas)) {
            esValido = false;
          } else if (unaMatrizFichas[destinoY][destinoX] != null && esArco(destinoY, destinoX, unaMatrizFichas)
              && unaMatrizFichas[destinoY][destinoX].getJugador().equals(unJugador)) {
            esValido = false;
          }
        } else if (unaFicha.getJugador().equals(unJugador) && caminoLibreTorre(unasCoordenadas)) { //torre
          if (origenX != destinoX && origenY != destinoY) {
            esValido = false;
          }
          if (origenX == destinoX && origenY == destinoY) {
            esValido = false;
          }
          if (unaMatrizFichas[destinoY][destinoX] != null && !esArco(destinoY, destinoX, unaMatrizFichas)) {
            esValido = false;
          } else if (unaMatrizFichas[destinoY][destinoX] != null && esArco(destinoY, destinoX, unaMatrizFichas)
              && unaMatrizFichas[destinoY][destinoX].getJugador().equals(unJugador)) {
            esValido = false;
          }
```

```
        } else {
            esValido = false;
        }
    }

    return esValido;
}

public boolean caminoLibreTorre(int[] unasCoordenadas) {

    int origenX = unasCoordenadas[0];
    int origenY = unasCoordenadas[1];
    int destinoX = unasCoordenadas[2];
    int destinoY = unasCoordenadas[3];

    boolean estaLibre = true;

    if (origenX == destinoX && origenY - destinoY > 0) {
        //se mueve hacia arriba
        int fila = origenY - 1;
        while (fila > destinoY && estaLibre) {
            if (this.getTablero()[fila][origenX] != null) {
                estaLibre = false;
            }
            fila--;
        }
    }
    if (origenX == destinoX && origenY - destinoY < 0) {
        //se mueve hacia abajo
        int fila = origenY + 1;
        while (fila < destinoY && estaLibre) {
            if (this.getTablero()[fila][origenX] != null) {
                estaLibre = false;
            }
            fila++;
        }
    }
    if (origenX - destinoX < 0 && origenY == destinoY) {
        //se mueve hacia la derecha
        int columna = origenX + 1;
        while (columna < destinoX && estaLibre) {
            if (this.getTablero()[origenY][columna] != null) {
                estaLibre = false;
            }
            columna++;
        }
    } else if (origenX - destinoX > 0 && origenY == destinoY) {
        //se mueve hacia la izquierda
        int columna = origenX - 1;
        while (columna > destinoX && estaLibre) {
            if (this.getTablero()[origenY][columna] != null) {
                estaLibre = false;
            }
            columna--;
        }
    }
    return estaLibre;
}

public boolean caminoLibreAlfil(int[] unasCoordenadas) {

    int origenX = unasCoordenadas[0];
```

```
    int origenY = unasCoordenadas[1];
    int destinoX = unasCoordenadas[2];
    int destinoY = unasCoordenadas[3];

    boolean estaLibre = true;

    if (origenX - destinoX < 0 && origenY - destinoY > 0) {
      //se mueve hacia arriba a la derecha
      int fila = origenY - 1;
      int columna = origenX + 1;
      while (fila >= destinoY + 1 && columna <= destinoX - 1 && estaLibre) {
        if (this.getTablero()[fila][columna] != null) {
          estaLibre = false;
        }
        fila--;
        columna++;
      }
    }
    if (origenX - destinoX < 0 && origenY - destinoY < 0) {
      //se mueve hacia abajo a la derecha
      int fila = origenY + 1;
      int columna = origenX + 1;
      while (fila <= destinoY - 1 && columna <= destinoX - 1 && estaLibre) {
        if (this.getTablero()[fila][columna] != null) {
          estaLibre = false;
        }
        fila++;
        columna++;
      }
    }
    if (origenX - destinoX < 0 && origenY - destinoY < 0) {
      //se mueve hacia abajo a la izquierda
      int fila = origenY + 1;
      int columna = origenX - 1;
      while (fila <= destinoY - 1 && columna >= destinoX + 1 && estaLibre) {
        if (this.getTablero()[fila][columna] != null) {
          estaLibre = false;
        }
        fila++;
        columna--;
      }
    } else {
      int fila = origenY - 1;
      int columna = origenX - 1;
      while (fila >= destinoY + 1 && columna >= destinoX + 1 && estaLibre) {
        if (this.getTablero()[fila][columna] != null) {
          estaLibre = false;
        }
        fila--;
        columna--;
      }
    }
    return estaLibre;
  }

  public int numeroCorrespondiente5(int unIndice) {
    int indice = unIndice;
    int[] correspondiente = new int[]{4, 3, 2, 1, 0};
    indice = correspondiente[indice];
    return indice;
  }
  //Para transformar lo que ingreso el usuario
```

```
  public int numeroCorrespondiente3(int unIndice) {
    int indice = unIndice;
    int[] correspondiente = new int[]{2, 1, 0};
    indice = correspondiente[indice];
    return indice;
  }

  public Ficha[][] mover(int[] unasCoordenadas) {

    int origenX = unasCoordenadas[0];
    int origenY = unasCoordenadas[1];
    int destinoX = unasCoordenadas[2];
    int destinoY = unasCoordenadas[3];
    Ficha fAux = new Ficha(this.tablero[origenY][origenX].getEsAlfil(), this.tablero[origenY][origenX].getEsAzul(), destinoY,
        destinoX, this.tablero[origenY][origenX].getJugador());
    this.getTablero()[destinoY][destinoX] = fAux;
    this.getTablero()[origenY][origenX] = null;
    this.turno = this.turno + 1;

    this.getTablero()[destinoY][destinoX].setEsAlfil(!this.getTablero()[destinoY][destinoX].getEsAlfil());

    if (this.tablero.length == 5) {

      if (this.getTurno() % 2 == 0) {

        this.historialJugadas.add("Azul: " + " " + (char) (origenX + 65) + (char) (numeroCorrespondiente5(origenY) + 49) + " "
            + (char) (destinoX + 65) + (char) (numeroCorrespondiente5(destinoY) + 49));
      } else {

        this.historialJugadas.add("Rojo: " + " " + (char) (origenX + 65) + (char) (numeroCorrespondiente5(origenY) + 49) + " "
            + (char) (destinoX + 65) + (char) (numeroCorrespondiente5(destinoY) + 49));
      }
    } else {

      if (this.getTurno() % 2 == 0) {
        this.historialJugadas.add("Azul: " + " " + (char) (origenX + 65) + (char) (numeroCorrespondiente3(origenY) + 49) + " "
            + (char) (destinoX + 65) + (char) (numeroCorrespondiente3(destinoY) + 49));
      } else {

        this.historialJugadas.add("Rojo: " + " " + (char) (origenX + 65) + (char) (numeroCorrespondiente3(origenY) + 49) + " "
            + (char) (destinoX + 65) + (char) (numeroCorrespondiente3(destinoY) + 49));
      }

    }
    return this.tablero;
  }
}
```

UML


Ficha
-esAlfil: boolean
-esAzul: Boolean
- filas: int
-columna: int
-jugador: Jugador
+Ficha()
+Ficha(boolean unIndicador, Boolean unIndicadorColor, int unaFila, int unaColumna, Jugador unJugador)
+toString(): String
Partida
-tablero: Ficha[][]
-jugadorRojo: Jugador
-jugadorAzul: Jugador
-turno: int
-tableroRotado: boolean
-historialJugadas: ArrayList <String>
-listaJugadasDefensa: ArrayList <String>
- listaJugadasAtaque: ArrayList <String>
- listaOtrasJugadas: ArrayList <String>
+Partida()
+jugadasDefensaDisponibles(): void
+ jugadasAtaqueDisponibles(): void
+otrasJugadasDisponibles(): void
+ crearTablero(int unaDimension): void
+ movAlfil(int origen, int origenY, int destinoX, int destinoY): boolean
+esArco(int unaFila, int unaColumna, Ficha [][] unaMatrizFichas): boolean
+ movimienoValido(Ficha unaFicha, Jugador unJugador, int[] unasCoordenadas, Ficha[][] unaMatrizFichas): boolean
+ caminoLibreTorre(int [] unasCoordenadas): boolean
+ caminoLibreAlfil (int [] unasCoordenadas): boolean
+ numeroCorrespondiente5(int unIndice): int
+ numeroCorrespondiente3(int unIndice): int
+ mover(int[] unasCoordenadas): Fichas[][]
1
2
Jugador
-nombre: String
-alias: String
-edad: int
-esAzul: boolean
- partidasGanadas: int
-partidasEmpatadas: int
-partidasPerdidas: int
+ Jugador(String unNombre, String unAlias, int unaEdad)
+Jugador()
+equals(): boolean
+compareTo(): int
+toString(): String
*
Juego
- listaJugadores: Arraylist<Jugador>
+ agregarJugador(Jugador unJugador): void
+aliasValido(Jugador unJugador):boolean
+ ranking(): ArrayList<Jugador>