Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Juegos Infinitos

Autor: Daniel Berrocal González
Tutor(a): F. Águeda Mata Hernández

Madrid, Junio 2021

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Juegos Infinitos

Junio 2021

Autor: Daniel Berrocal González
Tutor: F. Águeda Mata Hernández
Matemática Aplicada a las TIC
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Los juegos han estado presentes a lo largo de toda nuestra historia. Hoy en día continúan presentes en nuestra sociedad.
Los juegos más antiguos datan del año 3000 a. C. que fueron encontrados en el Antiguo Egipto y Mesopotamia. Estos juegos tenían **información completa**, es decir, los jugadores conocían las jugadas realizadas anteriormente en el juego, y también sabían las jugadas que podían realizar con las diferentes situaciones del juego.
Con el paso de los años, los juegos se seguían desarrollando y creando nuevos juegos, sin embargo, hasta el siglo XIV no se escribieron los primeros libros analizando las estrategias para algunos juegos. Uno de los juegos que se analizó fue el *Ajedrez*, como veremos más adelante, este juego ha sido importante en el desarrollo del estudio de los juegos.
En 1713 se empezaron a desarrollar teorías matemáticas para los juegos existentes en ese momento.
A principios del siglo XX, el matemático Ernst Zermelo, elaboró la primera teoría generalizada acerca de los juegos. Zermelo se centró en el juego del ajedrez para su investigación, observó que el conjunto de posiciones posibles es un número finito y las jugadas se pueden representar como una secuencia de posiciones consecutivas. El resultado de la investigación de Zermelo fue la elaboración del **Teorema Fundamental de los Juegos Combinatorios**.
Más adelante Roland Sprague y Patrick Michael Grundy desarrollaron la **Teoría de los Juegos Imparciales** de forma independiente. El desarrollo de este trabajo gira en torno a dicha teoría, realizando el estudio de un artículo sobre la **Función de Sprague-Grundy Generalizada**. La referencia del artículo se puede consultar en la bibliografía, en el punto 2.
Unos años después, Von Neumann y Morgenstern desarrollaron la **Teoría de Juegos** en el libro *The Theory of Games Behavior* publicado en 1944. Escribieron sobre el planteamiento estratégico para juegos de dos personas como es el ajedrez. También estudiaron los juegos compuestos por varios jugadores, su objetivo era encontrar la estrategia óptima para un jugador, aunque para este caso no lograron ser tan precisos como para el caso de dos personas.
El proyecto se desarrolla en varias partes: una primera parte para explicar la historia de los juegos, una segunda parte de desarrollo donde se estudian los diferentes conceptos de los juegos y los primeros etiquetados, llegando a la tercera parte donde se estudia el artículo mencionado anteriormente, y por último, se ha desarrollado un juego de dos jugadores, una persona y la máquina, con

el objetivo de que un jugador consiga realizar el último movimiento válido a la meta para ganar la partida.

Abstract

Games have been present around throughout our history. Nowadays they are still present in our society.
The oldest games date back to 3000 BC. C. that were found in Ancient Egypt and Mesopotamia. These games had **complete information**, namely, the players knew the moves made previously in the game, and they also knew the moves they could make with the differents game situations.
Over the years, the games continued to develop and creating new games, however, it was not until the 14th century that the first books were written analyzing the strategies for some games. One of the games that was analyzed was *Chess*, as we will see later, this game has been important in the development of the study of games.
In 1713 they began to develop mathematical theories for the games that existing at the time.
At the beginning of the 20th century, the mathematician Ernst Zermelo developed the first generalized theory about games. Zermelo focused on the game of chess for his research, he observed that the set of possible positions is a finite number and the moves can be represented as a sequence of consecutive positions. The result of Zermelo's research was the elaboration of the **Fundamental Theorem of Combinatorial Games**.
Afterwards, Roland Sprague and Patrick Michael Grundy developed the **Fair Game Theory** independently. The development of this work revolves around this theory, carrying out the study of an article on the **Generalized Sprague-Grundy**. The reference of the article can be consulted in the bibliography, in the point 2.
A few years later, Von Neumann and Morgenstern developed **Game Theory** in the book *The Theory of Games Bahavior* published in 1944. They wrote about the strategic approach to a two-person game such as chess. They also studied games composed of several players. Their objective was to find the optimal strategy for one player, although for this case they could not be as precise as for the case of two-person game.
The project is developed in several parts: a first part to explain the games history, a second part of development where the differents concepts of the games and the first labeled are studied, a third part where the article mentioned above is studied. Finally, it has developed a game of two players: a person and a machine, with the objective that one player manages to make the last valid move to the goal to win the game.

Tabla de contenidos

Capítulo 1

Introducción

Los juegos han sido jugados y estudiados por muchas civilizaciones a lo largo de la historia.
Los juegos más antiguos datan de al menos el año 3000 a. C. y fueron encontrados en el Antiguo Egipto y Mesopotamia. En estos juegos se podía saber los movimientos que se habían realizado anteriormente y los que se podían hacer en los siguientes movimientos, aunque para estos juegos también intervenía un factor de azar como son los dados.
Hace al menos 2500 años, en China, se jugaba a un juego llamado *Weiqi*. Este juego cuenta hoy en día con múltiples seguidores y todavía se sigue jugando.
En el siglo XIV, en China, ya había una buena cantidad de libros que estudiaban los juegos de esa época, los libros contenían diversas estrategias para los juegos. Paralelamente en Occidente, se estudiaban estrategias para el juego del *Ajedrez*. Algunos de esos libros trataban de relacionar las clases de posiciones con el resultado del juego.
Sin embargo, hasta 1713 no sé intentó desarrollar una teoría matemática a los juegos existentes hasta la fecha. Se desarrolló una solución completa para un juego de cartas muy popular en esas fechas, aunque no se hizo ningún intento por extenderla a una teoría más general.
A principios del siglo XX, el matemático alemán Ernst Zermelo, elaboró la primera teoría generalizada acerca de los juegos. En el artículo se preguntaba si el valor de una posición arbitraria se podía definir objetivamente de forma matemática. Zermelo empezó la investigación en el juego del ajedrez, observó que el conjunto de posiciones posibles en el ajedrez es finito y cualquier jugada se puede representar como una secuencia de posiciones consecutivas. Más adelante demostró, por inducción, que el resultado de cada posición del ajedrez está determinado y los generalizó a los juegos combinatorios finitos arbitrarios. El resultado del estudio de Zermelo fue la elaboración del Teorema Fundamental de los Juegos Combinatorios que veremos más adelante desarrollado.
En la década de 1930 se desarrolló la Teoría de los Juegos Imparciales de forma independiente, los desarrolladores de la teoría fueron Roland Sprague primero, y después Patrick Michael Grundy. Aunque el trabajo de Sprague tuvo dificultades para ser reconocido. Sprague publicó sus resultados en una revista japonesa y las condiciones políticas obstaculizaron su distribución.

Richard Guy fue el encargado de juntar la Teoría de Sprague y de Grundy para conocer todas las aplicaciones que podía tener la Teoría de Sprague-Grundy. Durante el desarrollo de este trabajo nos centraremos en analizar y estudiar dicha teoría, veremos diferentes aplicaciones y ejemplos.

Capítulo 2

Desarrollo

2.1. Introducción a la Teoría de Juegos

Se llama **juego** a cualquier situación interactiva entre personas llamadas **jugadores**. La **Teoría de Juegos** hace un estudio matemático de los juegos. De entre todos los modelos matemáticos que estudian los juegos, nos centraremos en el modelo extensivo.

2.1.1. Conceptos fundamentales sobre los juegos según el modelo extensivo

- Todo juego tiene dos o más jugadores.
- Las reglas del juego deben especificar cómo empieza el juego. Los jugadores deben de saber cuál es la situación inicial al empezar el juego. Esta situación se denomina **posición inicial**.
- Las reglas del juego deben especificar los posibles movimientos de un jugador para una determinada posición dentro de la partida, es decir, el jugador debe conocer qué movimiento puede realizar desde la posición actual.
- El objetivo del juego es poder alcanzar una **posición final**. En esta posición no está permitido ningún movimiento, el juego termina al alcanzar dicha posición.
- El juego se denomina **normal** si el primer jugador que alcanza una posición final gana la partida, y se denomina **misere** si el primer jugador en alcanzar esa posición pierde la partida.
- Una **partida** de un juego es una sucesión de movimientos que empieza en una posición inicial y termina en una posición final.
- Una **estrategia** es una serie de instrucciones precisas que dicen al jugador qué movimiento debe realizar desde cada posición.
    - Una estrategia será **ganadora** si dirige al jugador a ganar independientemente de los movimientos del rival.

- Una estrategia será **no-perdedora** si el jugador no tiene una estrategia ganadora pero puede impedir que el rival alcance una victoria.

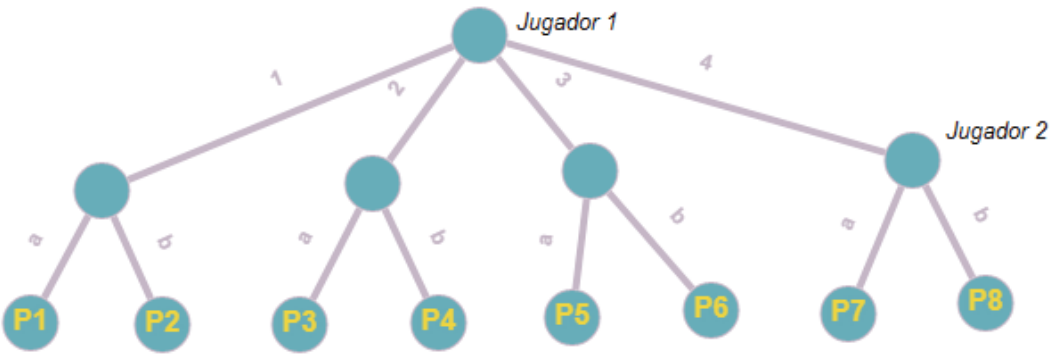
2.1.2. Tipos de Juegos

- Un juego es **finito** si las reglas del juego imposibilitan la realización de una partida con un número infinito de movimientos.
- Un juego es **infinito** si las reglas del juego posibilitan la realización de una partida con un número infinito de movimientos.
- Un juego es considerado **determinista** si en el juego no existen elementos aleatorios.
- Un juego tiene **información completa** en el caso de que todos los jugadores conozcan todas las jugadas que se han realizado en la partida y las que se pueden realizar desde cada situación del juego.
- Un juego es **imparcial** si en cada posición ambos jugadores tienen las mismas alternativas de movimiento.
- Un juego es **parcial** si desde alguna posición las alternativas de ambos jugadores no coinciden.

2.1.3. Representación de Juegos

Los juegos sencillos pueden representarse en un **diagrama** de árbol con las siguientes características:

- Cada posición es un **vértice** del diagrama. Dicho vértice se etiqueta para saber qué jugador o jugadores pueden moverse desde esa posición.
- Un posible movimiento de una posición a otra se representa mediante una **arista dirigida** del primer vértice al segundo y suele representarse en un nivel inferior al primero.
- La **posición inicial** del juego es aquella que tiene su vértice en el extremo superior del diagrama (**vértice raíz**).



- Toda posición final proporciona un **pago**.

2.2. Juegos Combinatorios Imparciales

Un juego es **combinatorio** si tiene las siguientes características:

1. Es un juego para dos personas.
2. Es determinista.
3. El juego tiene información completa.
4. Los posibles finales del juego son ganar, perder o empatar.

Los juegos combinatorios se dividen en dos grandes grupos: los juegos parciales y los juegos imparciales. Vamos a comenzar centrándonos en los juegos imparciales a los que llamaremos **Juegos** Γ. Utilizaremos los dígrafos para construir un modelo matemático para los **Juegos** Γ. Un dígrafo $Dg = (V, E)$ consiste en un conjunto finito V cuyos elementos se llaman **vértices** y una familia finita $E \subseteq V \times V$ de pares ordenados de vértices a cuyos elementos llamaremos **aristas** o **arcos** y que denotaremos por (u, v), con $u, v \in V$

2.2.1. Clasificación de los vértices de un dígrafo

Sea $Dg = (V, E)$ un dígrafo. $\forall u \in V$, se definen los conjuntos:

- **Sucesores** del vértice u:

$$F(u) = \{v \in V / (u, v) \in E\}$$

- **Predecesores** del vértice u:

$$F^{-1}(u) = \{w \in V / (w, u) \in E\}$$

- Si $F(u) = \varnothing$ entonces u es una hoja de D_g.
- Definimos:

$$P_0 = \{v \in V / F(u) = \varnothing\} \subseteq V$$

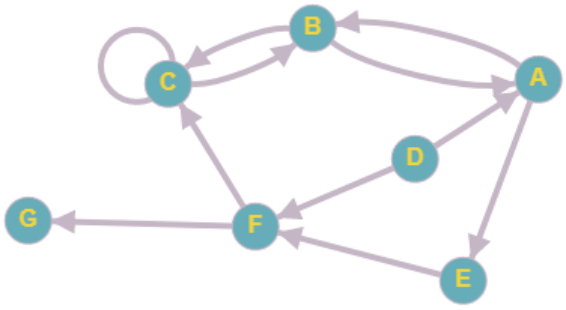
$$N_1 = \{u \in V / F(u) \cap P_0 \neq \varnothing\} \subseteq V$$

y,$\forall n \geq 1$

$$P_{2n} = \{u \in V \setminus \bigcup_{i=0}^{n-1} P_{2i} \cup N_{2i+1} / F(u) \subseteq \bigcup_{i=0}^{n-1} N_{2i+1}\}$$
$$N_{2n+1} = \{u \in V \setminus \bigcup_{i=0}^{n} N_{2i-1} \cup P_{2i} / F(u) \cap (\bigcup_{i=0}^{n} P_{2i}) \neq \varnothing\}$$

llamamos:

$$\mathbf{N} = \cup N_{2n+1} \quad \mathbf{P} = \cup P_{2n} \quad \mathbf{D} = V - (\mathbf{P} \cup \mathbf{N})$$



$P_0 = \{G\}$
$N_1 = \{F\}$
$P_2 = \{E\}$
$N_3 = \{A\}$
$P_4 = \{D\}$

Por tanto tenemos los siguientes conjuntos:
$\mathbf{N} = \{A, F\}$ $\mathbf{P} = \{D, E, G\}$ $\mathbf{D} = \{B, C\}$

Lema: Sea $Dg = (V, E)$ un dígrafo. Se tiene que:

a) $V = \mathbf{N} \cup \mathbf{P} \cup \mathbf{D}$

b) Si $u \in \mathbf{P} \Longrightarrow F(u) \subseteq \mathbf{N}$

c) Si $u \in \mathbf{N} \Longrightarrow F(u) \cap \mathbf{P} \neq \varnothing$

d) Si $u \in \mathbf{D} \Longrightarrow F(u) \cap \mathbf{P} = \varnothing$

e) $\mathbf{N} \cap \mathbf{P} = \varnothing$, $\mathbf{P} \cap \mathbf{D} = \varnothing$ y $\mathbf{N} \cap \mathbf{D} = \varnothing$

Demostración:

a) Por definición, puesto que: $\mathbf{D} = V \backslash (\mathbf{P} \cup \mathbf{N})$

b) Sea $u \in \mathbf{P} \Longrightarrow u \in P_{2n}$ para algún $n \in \mathbf{N}$, así: $F(u) \subseteq \bigcup_{i=1}^{n} N_{2i+1} \subseteq \mathbf{N}$

c) Sea $u \in \mathbf{N} \Longrightarrow \exists n / u \in N_{2n+1} \Longrightarrow F(u) \cap \bigcup_{i=0}^{n-1} P_{2i} \neq \varnothing \Longrightarrow F(u) \cap \mathbf{P} \neq \varnothing$

d) Sea $u \in \mathbf{D} \Longrightarrow u \in V \backslash (\mathbf{P} \cup \mathbf{N}) \Longrightarrow u \notin (\mathbf{P} \cup \mathbf{N}) \Longrightarrow$ Si $v \in F(u) \cap \mathbf{P} \Longrightarrow u \in \mathbf{N}$ contradicción $\Longrightarrow F(u) \cap \mathbf{P} = \varnothing$

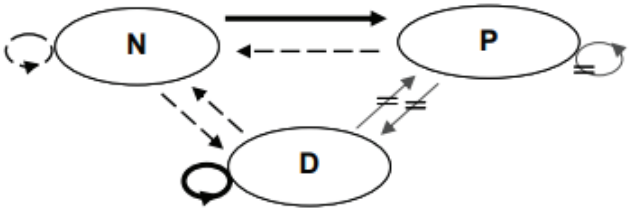
e) Por definición $\mathbf{P} \cap \mathbf{D} = \varnothing$ y $\mathbf{N} \cap \mathbf{D} = \varnothing$, veamos que $\mathbf{N} \cap \mathbf{P} = \varnothing$.
Si $u \in (\mathbf{N} \cap \mathbf{P}) \Longrightarrow \exists n_0, m_0 \in \mathbb{N}$ tales que $u \in N_{2n_0-1} \cap P_{2m_0}$.
Si $u \in N_{2n_0-1} \Longrightarrow 2m_0 > 2(n_0 - 1) \Longrightarrow m_0 > n_0 - 1$.
Si $u \in P_{2m_0} \Longrightarrow 2n_0 - 1 > 2m_0 - 1 \Longrightarrow n_0 > m_0$.
Llegando a una contradicción porque no existen $m_0, n_0 \in \mathbb{N}$ tales que $n_0 - 1 < m_0 < n_0$.

2.2.1.1. Observaciones

Hemos clasificado los vértices de un dígrafo $Dg = (V, E)$ en tres conjuntos disjuntos: **N**, **P** y **D**, que verifican lo siguiente:

- Desde un vértice de **N** tiene que haber al menos una arista a un vértice de **P**, aunque también puede haber aristas a vértices de **N** o de **D**.
- Desde un vértice **D** tiene que haber al menos una arista a otro vértice de **D**, aunque también puede haber a otros vértices de **N**.

- Desde un vértice **P** sólo puede haber aristas a vértices de **N** (y no obligatoriamente).



2.2.2. Modelo matemático de los juegos combinatorios imparciales

Dado un juego combinatorio imparcial Γ, se puede definir un dígrafo D_Γ, que llamaremos **dígrafo del juego**, del siguiente modo: $D_\Gamma = (V, E)$, donde V es el conjunto de posiciones de Γ, y $(u, v) \in E \iff$ hay un movimiento directo que va de la posición u a la posición v.

2.2.2.1. Observaciones

- Si $D_\Gamma = (V, E)$, es el dígrafo del juego finito Γ, entonces, en la partición del conjunto de vértices V se tiene $\mathbf{D} = \varnothing$, pues si $\exists u \in \mathbf{D} \implies$ existe una partida infinita en Γ, lo que contradice que Γ sea un juego finito. Por tanto se consigue $V = \mathbf{N} \cup \mathbf{P}$.
- El conjunto de posiciones finales del juego finito Γ se reduce a las posiciones en las que acaba el juego con la victoria de uno de los dos jugadores, y está contenido en el conjunto **P**.
- En los juegos **normales**, el primer jugador que alcance una **posición final**, **gana** la partida.

2.2.3. Teorema Fundamental de juegos combinatorios

Zermelo en 1912 estableció el **Teorema Fundamental de juegos combinatorios** donde dijo que en cualquier juego Γ, uno de los jugadores tiene una estrategia no perdedora. La demostración es la siguiente:

- Para ganar el juego hay que alcanzar una posición final $u \in P_0 \subseteq \mathbf{P}$.
- La única forma de llegar a una posición **P** es desde una posición **N**, por tanto, para ganar el juego hay que estar en una posición **N** y mover a una posición **P**.
- Si el juego está en un vértice **N**, el jugador que va a mover tendrá una estrategia ganadora que consiste en mover a una posición **P**.
- Si el juego está en una posición **P**, el jugador contrario tendrá una estrategia ganadora.
- Si el juego está en una posición **D**, ambos jugadores tienen una estrategia **no-perdedora**.

2.2.3.1. Solución de un juego combinatorio

Gracias al Teorema Fundamental de juegos combinatorios de Zermelo, podemos llegar a la conclusión de que, si la posición del juego es **N** o **P**, uno de los jugadores tiene una estrategia ganadora perfectamente definida. Esta estrategia consiste en mover siempre a posiciones **P**.

2.2.3.2. Algoritmo para etiquetar las pociones de un juego combinatorio

El siguiente algoritmo etiqueta las posiciones de un juego Γ en forma normal:

Paso 1. Etiquetar toda posición final como posición **P**.

Paso 2. Etiquetar como **N** toda posición que pueda alcanzar en un solo movimiento, una posición ya etiquetada como **P**.

Paso 3. Encontrar todas aquellas posiciones tales que todos sus movimientos terminan en posiciones ya etiquetadas como **N**. Etiquetar tales posiciones como **P**.

Paso 4. Si no se ha añadido ninguna nueva posición en los pasos 2 y 3, parar y etiquetar como **D** toda posición que todavía no ha sido etiquetada. En caso contrario volver al paso 2.

2.3. Función de Sprague-Grundy

Clasificar las posiciones de un juego en posiciones **N** y **P** no siempre es un problema computacionalmente tratable. Por ello R. P. Sprague en 1936 y P. M. Grundy en 1939 desarrollaron independientemente la teoría conocida como Sprague-Grundy que resulta útil para clasificar los vértices de un dígrafo con valores numéricos. Para calcular estos valores numéricos, se utiliza la función de Sprague-Grundy que se define, dado un digrafo simple, finito y sin ciclos $D_g = (V, E)$, para cada vértice $u \in V$ se define de forma recursiva el valor de $g(u)$, como el menor entero no negativo que no se encuentra entre los valores de los sucesores de u:

$$\forall u \in V \quad g(u) = min\{n \geq 0 / n \neq g(v), \forall v \in F(u)\}$$

Utilizaremos la función mex de un conjunto, que se define como el mínimo entero no negativo excluido del mismo:

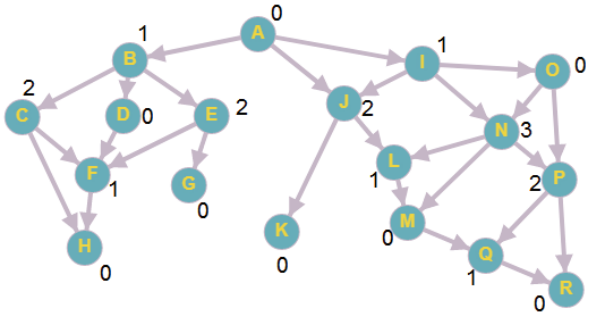
$$\forall u \in V \quad g(u) = mex\{g(v), v \in F(u)\}$$

Algunos ejemplos de uso de esta función son:

- $mex\{\varnothing\} = 0$
- $mex\{1, 0\} = 2$
- $mex\{1, 5\} = 0$

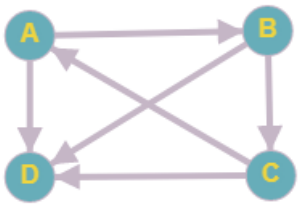
2.3.0.1. Observaciones

- La definición de la función de Sprague-Grundy es recursiva, este es un motivo por el cual el etiquetado de la función de Sprague-Grundy no parece mejorar respecto al etiquetado alfabético inicial, pero como se verá más adelante, cuando se trata de una suma de juegos este etiquetado es mucho más eficiente.
- La función de Sprague-Grundy no necesita ser inicializada porque $mex\{\varnothing\} = 0$.
- Si u_0 es una posición final del dígrafo de juego se tiene que $g(u_0) = 0$.
  Para los vértices u_1 que sólo tienen por adyacentes a posiciones finales, se verifica que $g(u_1) = 1$.



Valores para un dígrafo finito, simple y sin ciclos utilizando la función de Sprague-Grundy.

- Esta construcción inductiva funciona bien para dígrafos simples, finitos y sin ciclos, para los cuales veremos que existe una única función denominada **función Sprague-Grundy**, que asigna a cada vértice su único valor Sprague-Grundy. Para otro tipo de dígrafos puede que tal función no sea única o no exista.

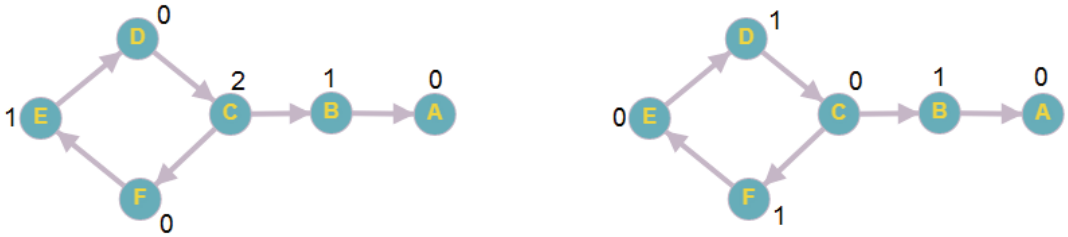


Por ejemplo, para algunos dígrafos como el de arriba no se les puede asignar correctamente los valores mex:

$$F(D) = \varnothing \Longrightarrow g(D) = 0; \quad F(A) = \{B, D\}; \quad F(C) = \{A, D\} \quad F(B) = \{C, D\}$$

$$g(A) = mex\{g(B), g(D)\}, \quad g(B) = mex\{g(C), g(D)\}, g(C) = mex\{g(A), g(D)\}$$

Si $g(B) = 1 \Longrightarrow g(A) = 2$ y $g(C) = 1 \Longrightarrow g(B) = 2$ contradicción. De forma análoga se tiene $g(A) \neq 1 \neq g(C)$



Como podemos ver en los dos grafos anteriores, a un mismo grafo se les puede asignar valores de forma diferente. Más adelante veremos la Teoría Generalizada de Sprague-Grundy donde se va a solucionar este problema.

2.3.0.2. Teorema de Existencia y Unicidad de la función de Sprague-Grundy

Un dígrafo finito, simple y sin ciclos $D_g = (V, E)$, tiene una única función de Sprague-Grundy $g : V \to \mathbb{N} \cup \{0\}$ tal que asigna a cada vértice de V su único valor de Sprague-Grundy. Además, $\forall u \in V$ el valor $g(u)$ no excede la longitud del camino más largo desde u en D_g.
Demostración:

- Consideramos los siguientes conjuntos:
  $C(0) = \{u \in V / F(u) = \varnothing\}$
  $C(1) = \{u \in V / u \notin C(0)$ y $F(u) \subseteq C(0)\}$
  $C(2) = \{u \in V / u \notin C(0) \cup C(1)$ y $F(u) \subseteq C(0) \cup C(1)\}$
  ...
  $C(r) = \{u \in V / u \notin C(0) \cup ... \cup C(r-1)$ y $F(u) \subseteq C(0) \cup ... \cup C(r-1)\}$

- $u \in C(k) \Leftrightarrow$ el camino más largo desde u tiene longitud k.
  Demostramos esto por inducción sobre k.
  Es cierto para $k = 0$ y $k = 1$.
  Supongamos que el resultado es cierto $\forall h \leq k$, y sea $u_0 \in C(k+1)$.
  Desde u_0 no puede haber un camino más largo que $k+1$, pues si $u_0, u_1, ..., u_n$ fuera un camino de longitud $n > k+1 \Rightarrow$ desde u_1 habría un camino de longitud $n-1 > k$, siendo $u_1 \in F(u) \subseteq C(0) \cup ... \cup C(k)$, contradicción.
  Veamos que desde u_0 hay un camino de longitud $k+1$:
  $u_0 \notin C(0) \cup ... \cup C(k)$, y $F(u_0) \subseteq C(0) \cup ... \cup C(k) \Rightarrow F(u_0) \nsubseteq C(0) \cup ... \cup C(k-1) \Rightarrow$
  $\Rightarrow \exists u_1 \in F(u) \cap C(k)$; por hipótesis de inducción el camino más largo desde u_1 tiene longitud $k \Rightarrow$ pasando por u_1, hay un camino dese u_0 de longitud $k+1$.

- Los conjuntos $\{C(n), n \in Z^+\}$ forman una partición de V.
  $C(k) \cap C(h) = \varnothing$, $\forall k \neq h$, pues no existe un vértice desde el cual, el camino más largo es a la vez h y k si $h \neq k$.
  $\forall u \in V$, el camino más largo desde u (por ser D_g finito y sin ciclos) tiene una longitud determinada $k \Rightarrow u \in C(k)$.

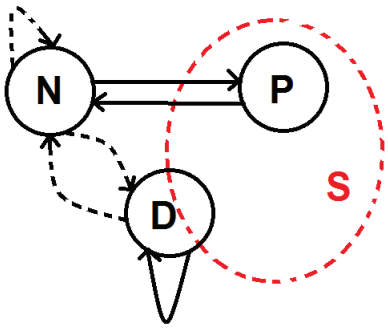
- Existe una única aplicación $g : V \to Z^+$, definida por
  $g(u) = mex\{g(v) / v \in F(u)\}$.
  Por inducción sobre k : $\forall u \in C(0), g(u) = 0$; $\forall u \in C(1), g(u) = 1$, supongamos que $\forall u \in C(0) \cup C(1) \cup ... \cup C(k)$, se ha definido de forma única $g(u) = mex\{g(v) / v \in F(u)\} \leq k$, sea $u_0 \in C(k+1) \Rightarrow F(u) \subseteq C(0) \cup ... \cup C(k) \Rightarrow$
  $\Rightarrow \forall v \in F(u_0), g(v) = mex\{g(x) / x \in F(v)\} \leq k$, y se ha definido de forma única entonces $g(u_0) = mex\{g(v) \leq k / v \in F(u_0)\} \leq k+1$, y está definido de forma única.

2.3.1. Teorema de Grundy

Si un dígrafo $D_g = (V, E)$ tiene una función Sprague-Grundy $g : V \longrightarrow \mathbb{N} \cup \{0\}$ entonces un jugador se asegura ganar o empatar si siempre elige vértices del conjunto $S = \{v \in V / g(v) = 0\}$.
La demostración es la siguiente:
Veamos que $S \supseteq$ **P**:

Por definición de la función Sprague-Grundy, las posiciones finales del grafo están en $S : P_0 \subseteq S$.
$\forall u \in S,\ g(u) = 0 \Longrightarrow \forall u \in S,\ g(u) = mex\{g(v)/v \in F(u)\} = 0 \Longrightarrow$
$\forall v \in F(u), g(v) > 0 \Longrightarrow \forall v \in F(u),\ v \notin S.$

$\forall u \notin S,\ g(u) > 0 \Longrightarrow \forall u \notin S,\ mex\{g(v)/v \in F(u)\} > 0 \Longrightarrow \exists v \in F(u)$ con $g(v) = 0 \Longrightarrow$ $F(u) \cap S \neq \varnothing$ y por el lema establecido en el apartado 2.2.1 se deduce que $\mathbf{P} \subseteq S$ y $\mathbf{N} \subseteq S^c$.



2.3.1.1. Observaciones

- Si no hay posiciones de empate, entonces, las posiciones anteriormente llamadas **P**, ahora son las posiciones en las cuales $g(u) = 0$.
- Las posiciones denominadas **N**, ahora son las posiciones en las que $g(u) \neq 0$.
- Para ganar un juego el objetivo es buscar alcanzar una posición $g(u) = 0$ desde una posición $g(w) \neq 0$.

En estos casos en los que no existen posiciones de empate, la función de Sprague-Grundy resuelve el juego por completo, y mejora considerablemente la eficacia respecto al etiquetado alfabético inicial cuando se trata de una suma de juegos. Pero el etiquetado proporcionado por la función de Sprague-Grundy, no distingue las posiciones de empate, como se puede ver en el gráfico anterior. Este problema será tratado en el siguiente capítulo donde se explicará la **Función de Sprague-Grundy Generalizada**.

Capítulo 3

Juegos Infinitos

3.1. Función Generalizada de Sprague-Grundy

Como hemos visto en las observaciones del punto 2.3.0.1, la función de Sprague-Grundy no distingue las posiciones de empate cuando existen. La función generalizada de Sprague-Grudny que veremos en esta sección será la encargada de solucionar el problema anterior. Esta nueva función implementa una nueva estrategia óptima para la suma de una amplia clase de juegos combinatorios representados por dígrafos que pueden contener ciclos o bucles. A esta nueva función la llamaremos G.
Limitaremos el resultado de los juegos de la siguiente manera:

- Si un jugador está en una posición desde la que no pueda moverse, el oponente será declarado ganador del juego.
- En los juegos en los que no haya un último movimiento, el resultado final será de **empate**.

Sea un dígrafo $D_G = (V, E)$, la función G que será útil para las estrategias de los diferentes juegos, se define de tres maneras equivalentes, como una aplicación $G : V \longrightarrow \mathbb{Z}^0 \cup \{\infty\}$. Donde $\mathbb{Z}^0$ es el conjunto de enteros no-negativos y el símbolo ∞ representa un número mayor que cualquier valor natural.

Definición 1. Una función $G : V \longrightarrow \mathbb{Z}^0 \cup \{\infty\}$ es una G-función con función contador $c : V^f \longrightarrow J$, donde $V^f = \{u \in V : G(u) < \infty\}$ y J es un conjunto bien ordenado si cumple las siguientes condiciones:

Condicción 1.1. Si $G(u) < \infty$, entonces $G(u) = mex\{G(F(u))\} = mex\{G(v) : G(v) < \infty, v \in F(u)\}$.

Condicción 1.2. Si $v \in F(u)$ y $G(v) > G(u)$, entonces existe $w \in F(v)$ que satisface $G(w) = G(u)$ y $c(w) < c(u)$.

Condicción 1.3. Si $G(u) = \infty$, entonces hay un $v \in F(u)$ con $G(v) = \infty(\mathfrak{K})$ tal que $mex\{G(F(u))\} \notin \mathfrak{K}$, donde $G(v) = \infty(\mathfrak{K})$ indica que $G(v) = \infty$ y $\mathfrak{K} = \{G(w) : G(w) < \infty, w \in F(v)\}$.

La motivación de la función contador es que el valor de G de un vértice no siempre determina al completo el siguiente movimiento óptimo. Por ejemplo, si un vértice tiene varios sucesores con el mismo valor G, elegir un sucesor que no esté en un ciclo puede ser la única opción para lograr una victoria. Ese sucesor tendrá un valor de la función contador menor que el resto. En consecuencia, para conseguir la victoria necesitaremos movernos a vértices con un valor determinado de G y un mínimo valor de la función contador.

Definición 2. Una función $G_1 : V \longrightarrow \mathbb{Z}^0 \cup \{\infty\}$ es una función G con función contador $c : V_1^f \longrightarrow J$ (donde $V_1^f = \{u \in V : G_1(u) < \infty\}$), si se cumplen las siguientes condiciones:

Condicción 2.1. Si $G_1(u) < \infty$ entonces para todo $i \in [0, G_1(u) - 1]$, existe $v \in F(u)$ satisfaciendo $G_1(v) = i$, $c(v) < c(u)$.

Condicción 2.2. Si $G_1(u) < \infty$ y $v \in F(u)$ satisface $c(v) < c(u)$, entonces $G_1(v) \neq G_1(u)$.

Condicción 2.3. Si $G_1(u) < \infty$ y $v \in F(u)$ que satisface o bien $G_1(v) = \infty$ o $c(v) \geq c(u)$, entonces existe $w \in F(v)$ tal que $G_1(w) = G_1(u)$ y $c(w) < c(u)$.

Condicción 2.4. Si $G_1(u) = \infty$, entonces hay un $v \in F(u)$ con $G_1(v) = \infty(\mathfrak{K})$ tal que $mex\{G_1(F(u))\} \notin \mathfrak{K}$, donde $G_i(v) = \infty(\mathfrak{K})$ indica que $G_1(v) = \infty$ y $\mathfrak{K} = \{G_1(w) : G_1(w) < \infty, w \in F(v)\}$.

Definición 3. Una función $G_2 : V \longrightarrow \mathbb{Z}^0 \cup \{\infty\}$ es una función G, si satisface las siguientes condiciones:

Condicción 3.1. Si $G_2(u) < \infty$, entonces $G_2(u) = mex\{G_2(F(u))\}$.

Condicción 3.2. Si $G_2(u) = \infty$, entonces hay un $v \in F(u)$ con $G_2(v) = \infty(\mathfrak{K})$ tal que $mex\{G_2(F(u))\} \notin \mathfrak{K}$ donde $G_2(v) = \infty(\mathfrak{K})$ indica que $G_2(v) = \infty$ y $\mathfrak{K} = \{G_2(w) : G_2(w) < \infty, w \in F(v)\}$.

Condicción 3.3. Todas las funciones que satisfacen las dos condiciones anteriores, son funciones con un número máximo de valores infinitos.

La construcción de una función G para cada dígrafo finito $D_G = (V, E)$ que puede contener ciclos o bucles, se consigue gracias al algoritmo que veremos a continuación. Inicialmente etiquetaremos cada vértice $u \in V$ con una etiqueta ν, esta etiqueta significa que el vértice no ha recibido aún una etiqueta definitiva.

3.2. Algoritmo Generalizado de la Función de Sprague-Grundy

Sea un dígrafo finito $D_G = \{V, E\}$:

Paso 1. *Inicializar etiquetas y contadores*. En este paso pondremos la variable $i = 0$, el contador $m = 0$ y etiquetaremos todos los vértices del dígrafo con la etiqueta ν. Sea $V_\nu = V$ el conjunto de los vértices que tienen la etiqueta igual a ν.

Paso 2. *Etiquetas y contadores*. Si existe un $u \in V_\nu$, tal que ningún sucesor de u tiene etiqueta i y cada sucesor de u que está sin etiquetar o que está etiquetado como ∞ tiene un sucesor etiquetado como i, entonces ponemos $l(u) = i$, eliminamos u del conjunto V_ν e incrementamos el contador $m = m + 1$. Repetimos el paso 2 hasta no poder etiquetar ningún vértice de V_ν de esta forma.

Paso 3. *Etiquetado* ∞. Una vez no podamos etiquetar utilizando el paso 2, entonces etiquetamos con ∞ todos los vértices sin etiqueta y que no tienen ningún sucesor etiquetado con i y eliminamos u de V_ν.

Paso 4. *Incrementar las etiquetas*. Si hay vértices sin etiquetar, entonces incrementamos $i = i + 1$ y volvemos al paso 2. En caso contrario terminamos el algoritmo.

Tras aplicar el algoritmo a un dígrafo podemos obtener dos conjuntos de etiquetas perfectamente diferenciadas. Por un lado tendremos las etiquetas con valor finito y por otro las etiquetas con valor infinito.

Definición 4. Una función contador c tal que $G(u) < G(v) < \infty \Longrightarrow c(u) < c(v)$ es llamada función contador monótona.

3.2.0.1. Relación entre el Algoritmo y la Función de Sprague-Grundy Generalizada

Las etiquetas l calculadas utilizando el algoritmo generalizado de la función de Sprague-Grundy, satisface la definición 2 vista anteriormente con una función contador monótona c para todo dígrafo finito $D_G = \{V, E\}$. La demostración es la siguiente:

i). Todo vértice $u \in V$ tiene exactamente una etiqueta.

ii). Para todo u tal que $l(u) = i < \infty$ y para todo $j \in [0, i-1]$, existe $v \in F(u)$ tal que $l(v) = j$. Supongamos que la afirmación anterior no es válida, entonces existe u tal que $l(u) = i < \infty$ y existe $j \in [0, i-1]$ tal que $l(v) \neq j \ \forall v \in F(u)$. Entonces, en esas condiciones u debería haberse etiquetado en la iteración j, con el valor j, o haberse etiquetado con ∞ en la iteración j, llegando de esta forma a una contradicción. Esto demuestra que se cumple la condición 2.1 porque se cumple también la monotonicidad de c, ya que si v fue etiquetado antes que u entonces $c(v) < c(u)$.

iii). Suponemos $l(u) = i < \infty$ y v es un sucesor de u. Queremos demostrar que entonces $l(v) \neq l(u)$. Llegamos a esta conclusión utilizando el paso 2 del algoritmo descrito anteriormente, por tanto se cumple la condición 2.2.

iv). Suponemos $l(u) = i$, $v \in F(u)$ y $(l(v) > l(u)$, o $c(v) \geq c(u))$. Si $c(v) \geq c(u)$, entonces $l(v) \geq l(u)$ por definición. Como hemos visto en el punto iii), $l(v) > l(u)$ y se cumple para cualquier caso. Por lo tanto para i-ésima iteración, v no está etiquetado o está etiquetado con $l(v) = \infty$. Dado que u se etiqueta en la i-ésima iteración, el paso 2 del algoritmo implica la existencia de algún $w \in F(v)$ que ya estuviera etiquetado como i, por lo tanto $c(w) < c(u)$ y se verifica la condición 2.3.

v). Si $l(u) = \infty$, entonces existe $v \in F(u)$ tal que $l(v) = \infty(\mathfrak{K})$, $mex\{l(F(u))\} =$ $= i \notin \mathfrak{K}$. Para todo $j \in [0, i-1]$, existe $v \in F(u)$ tal que $l(v) = j$. Por tanto, u no es etiquetado como ∞ en la j-ésima iteración. Dado que $v \in F(u) \Rightarrow$ $l(v) \neq i$, entonces u es etiquetado como ∞ por el paso 3 del algoritmo en la j-ésima iteración. Además, como u no se etiqueta utilizando el paso 2, entonces existe $v \in F(u)$ con v sin etiquetar o etiquetada con ∞, tal que $w \in F(v) \Rightarrow l(w) \neq i$. Si dicha v no estuviera etiquetada, esta se etiqueta con ∞ en el paso 3 en la i-ésima iteración. Por lo tanto se cumple la condición 2.4.

3.3. Equivalencias y singularidades de G

Para probar la equivalencia de las tres definiciones comenzamos con un resultado auxiliar.

3.3.0.1. Relación entre las definiciones 1 y 2 de la Función de Sprague-Grundy Generalizada

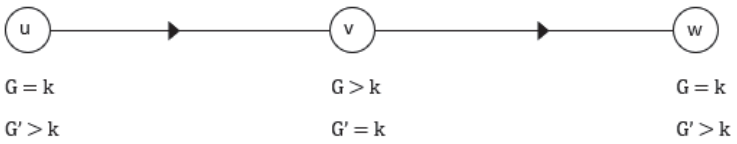
Si G satisface la definición 2, entonces G satisface la definición 1.
Demostración: Sea $u \in V^f$, donde V^f es un subconjunto de V en el cual G satisface la definición 2 con valores finitos: $G_1(u) < \infty$. Entonces veamos que $\forall v \in F(u)$ es $G(v) \neq G(u)$. Este resultado se deduce automáticamente por la condición 2.2, si se verifica que $c(v) < c(u)$, así suponemos $c(v) \geq c(u)$. Entonces por la condición 2.3 existe $w \in F(v)$ tal que $G(w) = G(u)$, $c(w) < c(u) \leq c(v)$. Si $G(v) < \infty$ entonces $c(w) < c(v) \Rightarrow G(v) \neq G(w) = G(u)$. Por lo tanto G satisface la condición 1.1.
Sea $u \in V^f$ y $v \in F(u)$. Si $G(v) = \infty$, entonces por verificarse la condición 2.3 se verifica la 1.2 si $G(u) < G(v) < \infty$, existe $z \in F(v)$ tal que $G(z) = G(u)$, $c(z) < c(v)$, por la condición 2.3 y entonces si $c(z) < c(u)$, para $w = z$, se verifica también la condición 1.2. En otro caso, si $c(u) \leq c(z) < c(v)$ por la condición 2.3 $\exists w \in F(v)$ tal que $G(w) = G(u)$ con $c(w) < c(u)$ por tanto se verifica la condición 1.2.

Aunque hay varias formas para seleccionar la función contador, las tres definiciones anteriores definen la misma función única G.

3.3.0.2. Equivalencia de las definiciones con la Función de Sprague-Grundy Generalizada

Las definiciones 1, 2 y 3 definen la misma función G, que existe y es única para todo dígrafo finito $D_G = \{V, E\}$.
Demostración: Por el teorema 1, existe una función G satisfaciendo las definición 2. Entonces la definición 1 se satisface por el lema 1, por tanto se verifican las condiciones 1.1 y 1.3 de donde se deduce que existe una función G' con las condiciones de la definición 3 para todo dígrafo finito D_G. Sea G cualquier función satisfaciendo la definición 1 o 2 con función contador c y G' cualquier función satisfaciendo la definición 3. Sea
$K = \{u \in V : G(u) < \infty, G(u) \neq G'(u)\}$, $k = \underset{u \in K}{min}(G(u), G'(u))$. Si existe $v \in K$ tal que $G'(v) = k$, entonces $k < G(v) < \infty$ y entonces también existe $u \in F(v)$ tal que $G(u) = k$ y por la condición 3.1 $G'(u) \neq k = min(G(u), G'(u)) \Rightarrow G'(u) > k \Rightarrow u \in K$.
Sea $U = \{u \in K : G(u) = k\}$. Por el razonamiento anterior se deduce que si $K \neq \varnothing \Rightarrow U \neq \varnothing$.
Sea entonces $u \in U$ con valor $c(u)$ mínimo. Entonces $G'(u) > k$. Supongamos que existe $v \in F(u)$ con $G'(v) = k$. Como $G(v) \neq k$ se deduce que $v \in K$, y como $k = min(G(v), G'(v)) \Rightarrow G(v) > k$ y por la condición 1.2, $\exists w \in F(v)$ tal que
$G(w) = k$, $c(w) < c(u)$ y por la condición 1.1 $G'(w) > k$. Por tanto $w \in U$ y contradice que $c(u)$ fuera mínimo.
Por otra parte si $G'(v) \neq k$, para todos los sucesores v de u, entonces por la condición 1.1 $G'(u) = \infty$. Para todo valor mínimo de k, si $i \in [0, k-1]$, existe $z \in F(u)$

tal que $G(z) = G'(z) = i$. Esto muestra que $mex\{G'(F(u))\} = k$. Por la condición 1.3, $\exists v \in F(u)$ tal que $G'(v) = \infty(\mathfrak{L})$ $k \notin \mathfrak{L}$. Ahora $G(v) > k$ por la condición 1.1 y por la 1.2, entonces existe $w \in F(v)$ tal que $G(w) = k$, $c(w) < c(u)$. Pero $G'(w) \notin k$ siendo $G'(w) > k$, donde $w \in U$, una contradicción del valor mínimo de c. Por lo tanto, $K = \varnothing$.
Si G satisface las condiciones 1.1 y 1.3, entonces G' es una función con un número máximo de valores infinitos entre todas las funciones, satisfaciendo las condiciones 1.1 y 1.3, teniendo entonces $G(u) = \infty \Rightarrow G'(u) = \infty$. Por lo tanto $G'(u) = G(u)$ $\forall u \in V$. De esta forma las definiciones 1, 2 y 3 se definan como la misma función.
Supongamos ahora que G_1 y G_2 satisfacen la definición 2 y G' satisface la definición 3. Por la primera parte de esta demostración, $G_1 = G'$, $G_2 = G'$, por tanto $G_1 = G_2$ demostrando la unicidad de la función.
Si $D_G = \{V, E\}$ es cualquier dígrafo finito, entonces la clásica función de Sprague-Grundy $g : V \to \mathbb{Z}^0$ es definida como $g(u) = mex\ g(F(u))$ para todo $u \in V$. Podemos observar que G es una generalización particular de g si $V^\infty = \varnothing$, implicando que $G = g$. Particularmente, si D_G es finito, sin ciclos y sin bucles, entonces $G = g$, porque $V^\infty = \varnothing$. Esto ocurre solamente si D_G es finito, sin ciclos y sin bucles, si D_G contiene ciclos, g podría existir únicamente con $g = G$, pero $g \neq G$, si el dígrafo contiene un bucle entonces implica $G(u) = \infty$.



3.4. Juegos con empates y la función G

Como hemos visto anteriormente, un empate se produce cuando ningún jugador puede forzar una victoria en un número finito de movimientos, pero ambos pueden moverse de forma infinita.

3.4.0.1. Observaciones

Como se ha visto en el tema 2, los vértices de todo dígrafo $D_G = (V, E)$ se pueden etiquetar con etiquetas **N**, **P** y **D** de forma que $u \in \mathbf{P}$ si y solo si $F(u) \subseteq \mathbf{N}$; $u \in \mathbf{N}$ si y solo si $F(u) \cap \mathbf{P} \neq \varnothing$; y $u \in \mathbf{D}$ si y solo si $F(u) \cap \mathbf{P} = \varnothing$ y $F(u) \cap \mathbf{D} \neq \varnothing$. Siendo $V = \mathbf{P} \cup \mathbf{N} \cup \mathbf{D}$ y $\mathbf{P} \cap \mathbf{N} = \mathbf{P} \cap \mathbf{D} = \mathbf{N} \cap \mathbf{D} = \varnothing$.

Además todo juego combinatorio Γ se puede modelizar mediante un dígrafo y de ellos se deduce el teorema Fundamental de juegos combinatorios que dice que en cualquier juego combinatorio, o bien uno de los dos jugadores tiene una estrategia ganadora o bien ambos pueden mantener un empate. Para comprobar esto, basta con demostrar que el conjunto de posiciones de Γ se puede dividir en tres subconjuntos **P**, **N** y **D**.
Supongamos que existe una posición u_0 que no es una posición **P** ni **N**. Entonces $F(u_0) \cap \mathbf{P} = \varnothing$, de lo contrario u_0 sería una posición **N** y $F(u_0) \nsubseteq \mathbf{N}$, o de lo contrario sería una posición **P**. Por lo tanto existe $u_1 \in F(u_0)$ tal que u_1 no es posición **P** ni **N**. Además, la única jugada no perdedora del jugador es mover de u_0 a u_1. Por lo tato, existe una secuencia infinita $u_0, u_1, ...$ con $u_{i+1} \in F(u_i)$ tal que $u_i \notin \mathbf{P} \cup \mathbf{N}$ y el mejor movimiento es ir de u_i a u_{i+1} siendo $i \geq 0$. Por lo tanto $u_0 \in \mathbf{D}$. Por lo tanto cada posición tiene una etiqueta **P**, **N** o **D**.
Suponemos $w \in \mathbf{P} \cap \mathbf{N}$. Empezando a jugar desde w, el siguiente jugador y el anterior pueden ganar, llegando a una contradicción en los posibles resultados. Por lo tanto $\mathbf{P} \cap \mathbf{N} = \varnothing$. De una forma similar se demuestra que $\mathbf{P} \cap \mathbf{D} = \mathbf{N} \cap \mathbf{D} = \varnothing$. El objetivo de la función G es determinar de una forma alternativa las etiquetas **P**, **N** y **D**.

3.4.0.2. Relación de la Función de Sprague-Grundy Generalizada con la clasificación de las posiciones

Sea $D_G = (V, E)$ un dígrafo finito. Entonces V se puede dividir de las siguiente formas:

$$\mathbf{P} = \{u \in V : G(u) = 0\}$$

$$\mathbf{D} = \{u \in V : G(u) = \infty(\mathfrak{K}), 0 \notin \mathfrak{K}\}$$

$$\mathbf{N} = \{u \in V : 0 < G(u) < \infty\} \cup \{u \in V : G(u) = \infty(\mathfrak{K}), 0 \in \mathfrak{K}\}$$

Demostración: Por lo que sabemos del tema anterior, todos los vértices de D_G tienen una única etiqueta **P**, **N** o **D**, y por el teorema 2 tiene una función G única. Llamemos **P'**, **D'** y **N'** a los tres conjuntos siguientes:

$$\mathbf{P'} = \{u \in V : G(u) = 0\}$$

$$\mathbf{D'} = \{u \in V : G(u) = \infty(\mathfrak{K}), 0 \notin \mathfrak{K}'\}$$

$$\mathbf{N} = \{u \in V : 0 < G(u) < \infty\} \cup \{u \in V : G(u) = \infty(\mathfrak{K}), 0 \in \mathfrak{K}\}$$

Entonces tenemos que:

$$\mathbf{P'} \cup \mathbf{N'} \cup \mathbf{D'} = V, \ \ \mathbf{P'} \cap \mathbf{N'} = \mathbf{P'} \cap \mathbf{D'} = \mathbf{N'} \cap \mathbf{D'} = \varnothing$$

Sea $u \in$ **P'**, $v \in F(u)$. Entonces $G(v) > 0$ por la condición 1.1 Si $G(v) = \infty(K)$, entonces $0 \in K$ por la condición 1.2. Por lo tanto $F(u) \subseteq$ **N'**.
Sea ahora $u \in$ **N'**, entonces tenemos claramente que $F(u) \cap$ **P'** $\neq \varnothing$.
Finalmente, si $u \in$ **D'**. Entonces $F(u) \cap$ **P'** $= \varnothing$. Además, $mex\{G(F(u))\} = 0$. Por lo tanto, por la condición 1.3, existe $v \in F(u)$ satisfaciendo $G(v) = \infty(\mathfrak{L})$ siendo $0 \notin \mathfrak{L}$ y $v \in$ **D'**.

3.4.0.3. Conlusiones

Al final de la primera iteración $(i = 0)$ del algoritmo G tenemos:

$$\mathbf{P} = \{u : l(u) = 0\}, \quad \mathbf{N} = V_\nu, \quad \mathbf{D} = V^\infty$$

Por lo tanto, las etiquetas **N**, **P** y **D** pueden ser calculadas en $|V| + |E|$ pasos o en $|E|$ pasos para un dígrafo conectado.
Demostración: La demostración se consigue inmediatamente del teorema 4 y la complejidad de los argumentos del algoritmo G.
El uso principal de la función de G es proporcionar una estrategia para una suma finita de los los juegos finitos. Para hacer esto definimos la suma NIM de la siguiente forma: Para cualquier entero no negativo, escribimos $H = \Sigma_{i \geq 0} h^i 2^i$ para la codificación binaria de H $(h^i) \in \{0, 1\}$. Si A y B son enteros no negativos,entonces su suma NIM $A \oplus B = C$ es definida como $c^i \equiv a^i + b^i$ (mod 2), siendo $c^i \in \{0, 1\}$ y $(i \geq 0)$. La suma NIM generalizada de un entero no negativo A y $\infty(K)$ se define como $A \oplus \infty(\mathfrak{K}) = \infty(\mathfrak{K}) \oplus A = \infty(\mathfrak{K} \oplus A)$, donde $\infty(\mathfrak{K} \oplus A) = \{k \oplus A : k \in \mathfrak{K}\}$. La suma NIM generalizada de $\infty(\mathfrak{K}_1)$ y $\infty(\mathfrak{K}_2)$ se define como $\infty(\mathfrak{K}_1) \oplus \infty(\mathfrak{K}_2) = \infty(\mathfrak{K}_2) \oplus \infty(\mathfrak{K}_1) = \infty(\varnothing)$.
Finalmente, la suma NIM generalizada para $m \geq 2$ es $\Sigma'^m_{i=1} A_i = A_1 \oplus ... \oplus A_m$ lo cual está bien definido ya que la suma NIM generalizada es asociativa.

3.4.0.4. Función de Sprague-Grundy de una Suma de Juegos

Sea $D_G = (V, E)$ el grafo del juego de la suma de los dígrafos finitos $D_{G_1}, ..., D_{G_m}$, y sea $\sigma(\mathbf{u}) = \Sigma'^m_{i=1} G(u_i)$ para todo $\mathbf{u} = (u_1, ..., u_m) \in V$, donde se nota por Σ' a la suma NIM. Entonces σ es la única función de G de D_G con función contador $c(\mathbf{u}) = \Sigma^m_{i=1} G(u_i) c_i(u_i)$, donde c_i son las funciones contador monótonas de G en D_{G_i} $(1 \leq i \leq m)$.
Para la demostración consultar la segunda referencia en la bibliografía.

3.4.0.5. Conclusiones

Sea $D_G = (V, E)$ el grafo del juego de la suma de los dígrafos finitos $D_{G_1}, ..., D_{G_m}$ $(m \geq 1)$. Entonces las etiquetas de **P**, **N** y **D** del dígrafos están determinadas por:

$$\mathbf{P} = \{\mathbf{u} \in V : \sigma(\mathbf{u}) = 0\}$$

$$\mathbf{D} = \{\mathbf{u} \in V : \sigma(\mathbf{u}) = \infty(\mathfrak{K}), 0 \notin \mathfrak{K}\}$$

$$\mathbf{N} = \{\mathbf{u} \in V : 0 < \sigma(\mathbf{u} < \infty)\} \cup \{\mathbf{u} \in V : \sigma(\mathbf{u}) = \infty(\mathfrak{K}), 0 \in \mathfrak{K}\}$$

Sea $n = max(|V_1|, ..., |V_m|)$. Si el tamaño de los juegos iniciales es al menos una potencia positiva de mn, y durante el juego ninguna posición crece mas que la fijada por la potencia de mn, entonces el cálculo de las etiquetas **P**, **N** y **D** es polinomial.

Además, dada la naturaleza de las posiciones del juego (**P**, **N** o **D**), el teorema 5 también indica un siguiente movimiento óptimo para las posiciones **N** y **D**.
La idea es que si $\mathbf{u} = (u_1, ..., u_m)$ es una posición **P**, que a su vez, es una posición con $\sigma(\mathbf{u}) = 0$, entonces nuestro oponente mueve a $\mathbf{v} \in F(\mathbf{u})$ con $\sigma(\mathbf{v}) > 0$, y podemos encontrar algún $\mathbf{w} \in F(\mathbf{v})$ con $\sigma(\mathbf{w}) = 0$ y $c(\mathbf{w}) < c(\mathbf{u})$. Esta última desigualdad se consigue moviendo un componente del juego con valor mínimo c_i de entre todos los componentes que tienen los valores específicos de G. Como la función contador está bien ordenada, se deduce que podemos obtener la victoria en un número finito de movimientos. Por la condición 1.3 tenemos que si $\mathbf{u} \in \mathbf{D}$, entonces podemos encontrar un sucesor $\mathbf{v} \in \mathbf{D}$, por lo que se puede mantener una posición **D**.

3.5. Homomorfismos entre juegos

Para el dígrafo finito $D_G = (V, E)$ que puede tener ciclos y bucles, introducimos una nueva notación $F'(u) = F(u) - \{u\}$. Si el dígrafo no tiene bucles la notación sería $F'(u) = F(u), \forall u \in V$.

Definición 5. Sea $D_G, \hat{D_G}$ dos dígrafos finitos. Una aplicación $\lambda : V(D_G) \to V(\hat{D_G})$ se llama morfismo D si para todo $u \in V(D_G)$:

1. $F_{\hat{D_G}}(\lambda(u)) \subseteq \lambda(F_{D_G}(u))$.
2. $\lambda(F'_{D_G}(u)) \subseteq F_{\hat{D_G}}(\lambda(u)) \cup F^{-1}_{\hat{D_G}}(\lambda(u))$.

Donde para cualquier conjunto S, $\lambda(S) = \{\lambda(s) : s \in S\}$, por tanto $\lambda(S_1 \cup S_2) = \lambda(S_1) \cup \lambda(S_2)$ y los subíndices D_G, $\hat{D_G}$ de F indican al dígrafo al que pertenecen los sucesores correspondientes.
Si D_G no tiene bucles, entonces la definición anterior coincide con la definición de Banerji de un morfismo D. Para dígrafos acíclicos y sin bucles, Banerji demostró que si $\hat{D_G}$ tiene una función de Sprague-Grundy g, entonces la función $g'(u) = g(\lambda(u))$ es una función de g en D_G. La demostración de Banerji se puede encontrar en la bibliografía en la tercera referencia.
Si D_G y $\hat{D_G}$ son acíclicos y sin bucles, entonces $\hat{D_G}$ tiene una función de Sprague-Grundy, que a su vez, determina una estrategia ganadora para el dígrafo D_G. Por lo tanto λ relaciona la estrategia ganadora de $\hat{D_G}$ con la de D_G. En el caso que D_G o $\hat{D_G}$ tengan ciclos o bucles, entonces podría no existir la función de Sprague-Grundy y en el caso de existir, la función de Sprague-Grundy no necesariamente determinaría la estrategia ganadora. Pero la función de Sprague-Grundy generalizada: G, existe y es única en todo dígrafo, por ello es de interés estudiar si se conserva por homomorfismos.

3.5.0.1. Relación entre los Morfismos y la Función de Sprague-Grundy Generalizada sobre valores finitos

Sean D_G y $\hat{D_G}$ dígrafos finitos, y $\lambda : V(D_G) \to V(\hat{D_G})$ un morfismo D. Entonces $G(u) = G(\lambda(u))\ \forall u \in V^f(D_G)$.
La demostración de este resultado se puede consultar en la bibliografía en la cuarta referencia. Se demuestra también que si $V^\infty(D_G) \neq \varnothing$, entonces un morfismo D en general no conserva G. Pero bajo algunas hipótesis extras, G se puede conservar, como veremos después en el teorema 7.
Si f_i y g_i son conjuntos bien ordenados con $(i \in \{1, 2\})$, entonces la ordenación lexicográfica de los pares se define como $(f_1, g_1) < (f_2, g_2)$ si i) $f_1 \leq f_2$ y ii) $f_1 = f_2 \Rightarrow g_1 < g_2$.

3.5.0.2. Relación entre los Morfismos y la Función de Sprague-Grundy Generalizada sobre todos los valores

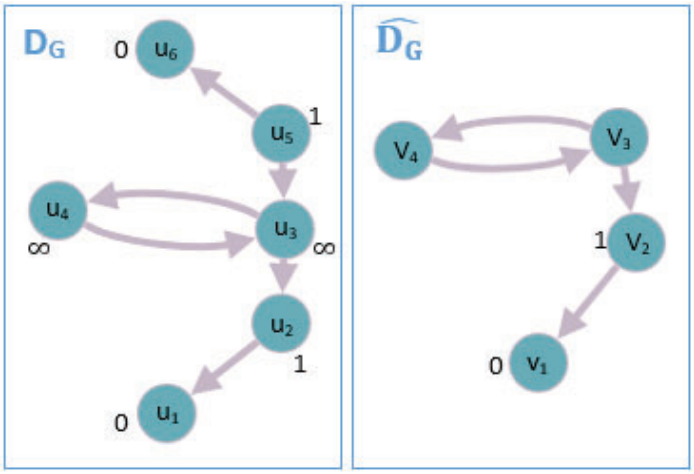
Sean D_G y $\hat{D_G}$ dígrafos finitos, y dos aplicaciones $\lambda : V(D_G) \to V(\hat{D_G})$ y $d : V(R) \to J$, donde J es cualquier conjunto bien ordenado, con las siguientes propiedades:

$\forall u \in V(D_G)$ $\mathfrak{L}$ satisface que $F_{\hat{D_G}}(\mathfrak{L}(u)) \subset \mathfrak{L}(F_{D_G}(u))$, y
$\forall u \in V(D_G)$ y $\forall v \in F_{D_G}(u)$ se cumple, o bien

1. $\lambda(v) \in F_{\hat{D_G}}(\lambda(u))$, o
2. $\exists w \in F_{D_G}(v)$ tal que $\lambda(w) = \lambda(u)$ y $d(w) < d(u)$.

Sea $G'(u) = G(\lambda(u))\forall u \in V(D_G)$ y dado $c(u) = (\hat{c}(\lambda(u)), d(u))$, ordenado lexicográficamente, siempre que esté definido $\hat{c}(\lambda(u))$, donde $\hat{c}$ es una función contador monótona en D_G. Entonces G' es una función Sprague-Grundy generalizada de D_G con función contador monótona c.

Ejemplo. Sean D_G y $\hat{D_G}$ dados en la figura de abajo, $\lambda(u_i) = v_i, (1 \leq i \leq 4)$, $\lambda(u_5) = v_2, \lambda(u_6) = v_1$. Entonces se satisface que $\forall u \in V(D_G)$, $F_{\hat{D_G}}(\mathfrak{L}(u)) \subset \mathfrak{L}(F_{D_G}(u))$ y también se satisfacen las condiciones de a) y b) mostradas anteriormente. Por ejemplo, el par (u_5, u_6) satisface a), sin embargo, el par (u_5, u_3) no satisface a), pero satisface b) con $w = u_2$, tomando $d(u_2)$ y $d(u_5)$ por ejemplo cualquier función contador monótona de D_G.



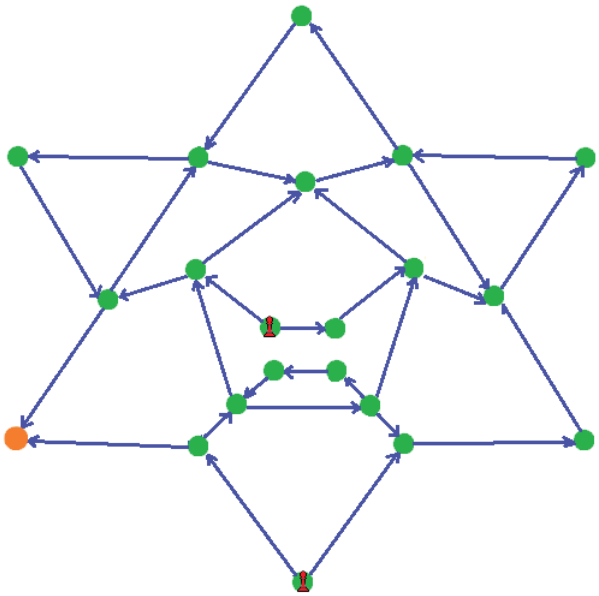
Capítulo 4

Implementación Particular de un Juego

4.1. Introducción al Juego implementado

Para complementar el estudio de los juegos imparciales, se ha desarrollado un juego imparcial entre un jugador y la CPU.
El juego consta de dos o tres fichas, a elegir por el jugador.

- Si el jugador decide jugar con dos fichas, la situación inicial del tablero será la siguiente.



  Inicialmente, las dos fichas han sido ubicadas en esas posiciones porque son vértices a los que no podemos mover, por tanto, si no empezamos con una ficha en ese vértice, dicho vértice no entraría en juego nunca.

- Si el jugador decide jugar con tres fichas, la situación inicial cambiará en

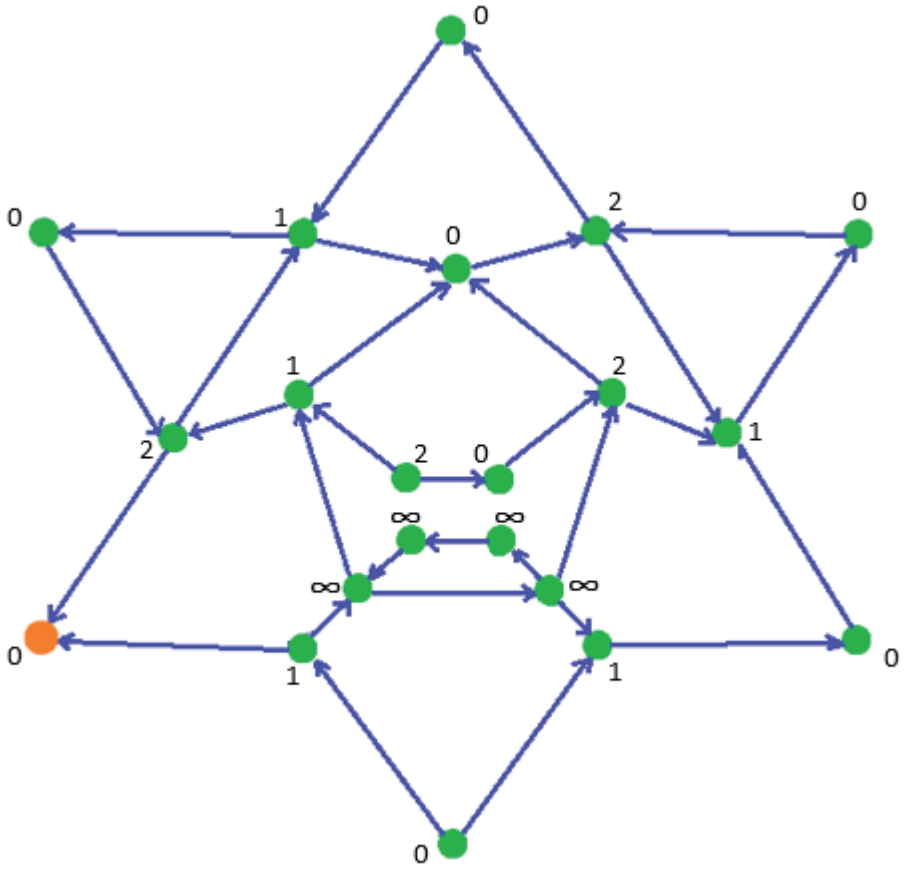
función de donde quiera ubicar la tercera ficha el jugador.

Para ganar la partida hay que llegar a la meta, vértice naranja, con todas las fichas que haya en juego. Como se ha desarrollado un juego imparcial, tanto el jugador como la CPU podrán mover la ficha que deseen en su turno. El ganador de la partida será el jugador que logre alcanzar la meta con la última ficha en juego realizando un movimiento válido.

4.2. Desarrollo de una partida

4.2.1. Etiquetado Sprague-Grundy para el Juego

Ya hemos visto anteriormente cómo es el tablero donde se van a jugar las partidas. El dígrafo representado en el tablero ha sido estudiado y etiquetado con los valores de Sprague-Grundy, obteniendo el siguiente resultado:

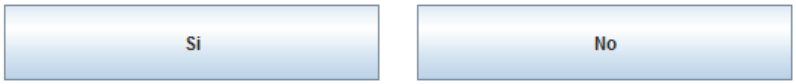


4.2.2. Pasos previos al inicio de una partida

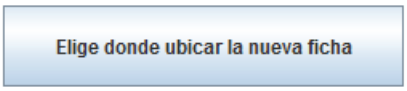
Antes de comenzar la partida, el jugador deberá elegir si utilizar dos o tres fichas durante la partida. Para ello el programa realizará la siguiente pregunta por pantalla al usuario:

¿Desea añadir otra ficha al tablero?

A continuación, el jugador deberá responder a esta pregunta utilizando uno de los dos botones habilitados para ello.



- Si el jugador pulsa el botón "Si", el botón "No" desaparecerá de la imagen y el botón "Si" tendrá un nuevo mensaje indicando al jugador que elija el lugar donde situar la tercera ficha.



  Una vez elegido el lugar de la ficha, la partida comenzará.

- Si el jugador pulsa el botón "No", tanto el botón de "Si" como el botón de "No" desaparecerán y comenzará la partida.

¡Qué empiece el juego!

4.2.3. Posiciones de Ventaja o Desventaja

Al inicio de cada turno, tanto el jugador como la CPU sabrán si están en posición de ventaja o en posición de desventaja.
El jugador comenzará moviendo ficha, además, el jugador sabrá si tiene ventaja o no.

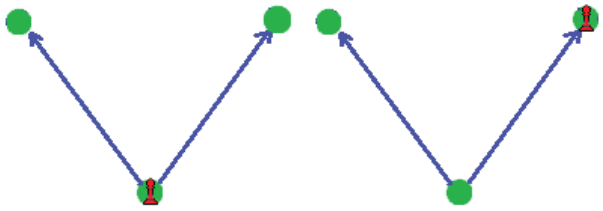
El jugador está en posición de ventaja

El cálculo para determinar si un jugador tiene ventaja o está en desventaja se realiza mediante la suma digital del valor de cada uno de los vértices donde se encuentren las fichas.

- Si el cálculo es igual a cero, el jugador que vaya a mover en ese turno se encontrará en posición de desventaja.
- Si el cálculo es mayor que cero, el jugador que mueva en ese turno estará en posición de ventaja.
- Si el cálculo es $\infty(\Re)$ y $0 \in \Re$, el jugador que mueva en ese turno tiene la posibilidad de mantener un empate.

4.2.4. Cómo mover Ficha durante la partida

Para realizar el movimiento de una ficha, hay que hacer "click" en el lugar donde se encuentre la ficha y después seleccionar el vértice al que se quiere ir.



Si al vértice al que queremos mover no es accesible desde el lugar de la ficha, se imprimirá por pantalla un mensaje informando al usuario de que elija una casilla correcta para mover.

```
Seleccione una casilla correcta para mover.
```

Si el vértice que elegimos para mover no tiene ficha, se mostrará por pantalla un mensaje para informar al usuario de que elija una posición que contenga una ficha.

```
Seleccione una casilla ocupada por una ficha.
```

4.2.5. Final del Juego

Cuando todas las fichas alcancen la meta, el juego habrá finalizado. El ganador se mostrará por pantalla.

- Si el ganador es el jugador, se mostrará el siguiente mensaje:

```
**************************************
* ¡Enhorabuena, has ganado la partida! *
**************************************
```

- Si gana la CPU, el mensaje será el siguiente:

```
La CPU ha ganado la partida.
```

4.3. Programación del Juego

Para programar este juego se han utilizado principalmente tres librerías:

1. Librería *event*. Esta librería se ha utilizado para implementar las acciones del usuario con el tablero. Dentro de la librería *event* se han utilizado las clases *ActionEvent*, *ActionListener*, *MouseEvent* y *ActionListener*.

2. Librería *swing*. Esta librería se ha utilizado para la interfaz gráfica del juego. Con esta librería podemos visualizar el tablero, las fichas y los botones iniciales de "Si" y "No". Las clases utilizadas dentro de la librería son *ImageIcon*, *JButton*, *JFrame* y *JLabel*.

3. Librería *util*. Esta librería se ha usado como complemento a la implementación de los movimientos, bien sea de la CPU o del usuario, más adelante veremos su uso con mayor profundidad. Las estructuras de datos usadas de esta librería son los *ArrayList* y *List*.

Además de las librerías mencionadas anteriormente, se han creado tres clases adicionales para la implementación.

1. La clase *Tablero*. Con esta clase se implementa todo lo relacionado con el juego, las dos clases restantes sirven de apoyo para el manejo de las distintas listas utilizadas. Los métodos utilizados son los siguientes:
   - Tablero(). Es el constructor de la clase. Con este método se inicializan las diferentes estructuras de datos que veremos más adelante.
   - iniciarDetectores(). Con este método se implementan los detectores utilizados cuando realizamos un movimiento. Este método está conectado con el método mouseClicked que veremos después.
   - botonesIniciales(). Inicializa los botones utilizados durante el inicio de la partida.
   - fichasIniciales(). Inicializa las fichas del juego, situando las dos fichas que tienen posición fija en su vértice correspondiente.
   - mouseClicked(MouseEvent evento).Este método está dividido en dos partes. La primera parte implementa la acción de elegir la casilla inicial para una ficha adicional. En la segunda parte se implementa el movimiento de las fichas, además de actualizar las estructuras de datos relacionadas con los movimientos que veremos más adelante.
   - actionPerformed(ActionEvent e). Con este método detectamos el botón utilizado al inicio para saber si añadir o no una nueva ficha.
   - boolean esPrimerClick(List<SituacionVertices>sv). El tipo SituacionVertice lo veremos como la segunda clase creada para ayudarnos en la implementación. Este método devuelve *true* en el caso de que sea la primera vez que pulsamos en un vértice para realizar el movimiento de la ficha después. Devuelve *false* para el caso contrario.
   - List<Integer>posParaMover(int[][] matriz, int posJ). Este método devuelve una lista de enteros. Dada la columna posJ, se busca en la matriz las posiciones a las que podemos mover, una vez encontrado el movimiento válido se añade a las lista.
   - int casillaDondeMover(List<Integer>posiciones, List<SituacionVertices>sv). Este método devuelve las posición donde mueve una ficha.
   - int piezaMover(boolean[] situacionVerticeAnterior). Busca la ficha que se va a mover para poder realizar el movimiento gráficamente en el tablero.
   - boolean[] actualizarSV(int pos, boolean[] actualizarV). Devuelve un array booleano con el objetivo de actualizar el ArrayList <SituacionVertices>que veremos más adelante su contenido.
   - boolean esVacia(boolean[] situacionNuevoVertice). Comprueba si el vértice está vacío o no. Este método se utiliza para poder visualizar dos fichas en el caso de que coincidan en el mismo vértice.

- int calcularSP(List<SituacionVertices>sv). Calcula el valor Sprague-Grundy para la situación que haya en ese momento en el tablero y poder informar al jugador si tiene ventaja o no.
- int calcularSP(int[] vertice). Este método realiza el mismo cálculo que el método anterior, sin embargo, el método anterior realiza el cálculo para una situación actual del tablero. Este nuevo método realiza el cálculo para una hipotética situación del tablero. Este método se utiliza para que la CPU realice el mejor movimiento posible y pueda obtener ventaja.
- moverCPU(int valorSP). Este método se encarga de preparar el movimiento de la CPU. Para hacer el movimiento, primero comprueba las diferentes situaciones del tablero que se pueden dar para cada ficha que haya en juego y, después, llama al método de moverFichaCPU con los parámetros óptimos para realizar el movimiento.
- moverFichaCPU(int casillaInicial, int casillaFinal, String piezaMover). Este método realiza el movimiento de la CPU, actualiza gráficamente el tablero con el movimiento utilizado y además, las estructuras de datos de apoyo para los movimientos son también actualizadas.
- int[][] inicializarMatrizAdyacencia(int [][] matriz). Inicializa la matriz de adyacencia para el juego.

2. La clase *SituacionVertices*. Esta clase crea un nuevo tipo llamado "SituacionVertices" que se utiliza de tipo en la lista más importante utilizada en la implementación. Los métodos utilizados en la clase *SituacionVertices* son los siguientes:
   - SituacionVertices(boolean[] casilla, int numV, int etiquetaV, boolean casillaPulsada). Es el constructor de la clase. El array de booleanos "casilla" contiene la situación cada uno de los vértices, la longitud de este array es cuatro: si el primer valor del array es true, entonces la casilla está vacía, un true en las tres posiciones siguientes indican que una determinada ficha está en esa casilla. El segundo parámetro del constructor indica el número del vértice en el que nos encontramos. El entero de "etiquetaV" indica el valor de la etiqueta de un determinado vértice, con este valor se calcula durante la partida el valor de Sprague-Grundy. Y por último, el boolean "casillaPulsada" indica si una casilla ha sido ya pulsada para después realizar el movimiento a una casilla válida.
   - El resto de métodos de la clase son getters y setters que ayudan en el manejo de la List<SituacionVertices>mencionada anteriormente.
3. La clase *ValorSP*. Esta clase implementa un nuevo tipo de datos utilizado para calcular el valos de Sprague-Grundy en los posibles movimientos a realizar por la CPU. Los métodos que contiene esta clase son los siguientes:
   - ValorSP(int valorSpragueGrundy, int vDondeMover, int vInicialFicha). vDondeMover y vInicialFicha indican los vértices donde se encuentra

una ficha antes de mover y el vértice donde estaría una ficha tras realizar el movimiento. El valorSpragueGrundy guarda el valor de Sprague-Grundy para después analizar el movimiento más óptimo de la CPU.

- Los métodos restantes son getters y setters para poder manejar mejor la estructura de datos utilizada para el movimiento de la ficha de la CPU.

El código utilizado para la implementación se encuentra al final de la memoria en el capítulo del Anexo.

Capítulo 5

Resultados y conclusiones

5.1. Resultados

Los resultados logrados con la elaboración de este trabajo son:

- Entender los conceptos presentes en los juegos.
- El conocimiento de los primeros etiquetados utilizados en los juegos. La comprensión de los etiquetados utilizados actualmente y sus ventajas respecto a los etiquetados anteriores.
- El juego implementado es el resultado del estudio y la elaboración de este proyecto.

5.2. Conclusiones

Tras la elaboración de esta memoria podemos sacar las siguientes conclusiones:

- Los conceptos utilizados durante la elaboración de este proyecto son, a priori, sencillos de entender porque son conceptos que estamos acostumbrados a manejar cuando jugamos a un juego.
- Sin embargo, algunas demostraciones o teoremas utilizadas pueden llegar a ser más difíciles de entender.
- En cuanto a la programación del juego desarrollado, podemos decir que ha sido una implementación laboriosa, llegando a utilizar aproximadamente 5700 líneas de código para implementar el juego.
- Personalmente, me ha gustado poder elaborar este trabajo orientado a los juegos imparciales, me parecía interesante estudiar sobre un tema desconocido pero a la vez tan presente en la sociedad como son los juegos. Aunque ha habido conceptos que me han costado más entender, algunas fórmulas he tenido que estudiarlas con mucha profundidad para finalmente llegar a entenderlas.

5.2.1. Líneas Futuras

El estudio de este trabajo se ha centrado en los juegos imparciales, un posible trabajo para un futuro sería centrarse en los juegos parciales. Durante la realización de este proyecto hemos mencionado la suma NIM de forma breve. Un proyecto de futuro podría centrarse en la suma NIM con mayor profundidad y ampliar con otras operaciones que pueden realizarse con los valores de un juego, como por ejemplo el producto NIM.

Capítulo 6

Análisis de impacto

En este capítulo vamos a estudiar el impacto que tiene la Teoría de Juegos sobre las empresas, economía y a nivel personal.

La Teoría de Juegos se utiliza con frecuencia en la toma de decisiones en las empresas. En general, las empresas compiten entre sí con el objetivo de tener mayores beneficios que otras del mismo sector. Para lograr la meta de ser la mejor empresa, las empresas utilizan la Teoría de Juegos, ya que, una empresa necesita de una estrategia que le indique el modo de actuar ante las distintas circunstancias que se puedan presentar, y a ser posible, dicha estrategia debe ser óptima. Una empresa con una buena estrategia estará preparada ante los posibles obstáculos que puedan aparecer en el camino a la meta. Por ejemplo, imaginemos un departamento de Marketing de una empresa. El objetivo del departamento de Marketing es lograr las máximas ventas posibles. Para lograrlo necesitan tener una estrategia que pase por tener en cuenta la opinión del cliente frente a los diferentes productos de la empresa y de los productos análogos de la competencia, de este modo, el departamento de Marketing puede orientar las ventas para conseguir la meta. En definitiva, con una estrategia óptima que tiene en cuenta la opinión del cliente y que consigue anticiparse a sus necesidades o a las nuevas tendencias de consumo, la empresa conseguirá ser mejor que las empresas rivales.

Después de conocer el impacto que tiene en la empresa la Teoría de Juegos, es fácil relacionarlo con el impacto a nivel económico. En el ejemplo anterior, el objetivo era lograr las máximas ventas posibles, es decir, conseguir que la empresa mejore a nivel económico.

En la sociedad, la Teoría de Juegos está presente de forma indirecta. Todos los juegos, sean de niños o adultos, de mesa o deportivos, siguen pautas modeladas por la Teoría de Juegos. En los juegos mencionados anteriormente, hay una meta a conseguir para poder ganar la partida, también se plantean diferentes situaciones, algunas más favorables y otras no tanto, por lo tanto, para poder maximizar las ganancias y minimizar las pérdidas del juego es necesario saber elegir una estrategia que se adapte mejor a la situación inicial que se presenta, al igual que sucede en la Teoría de Juegos.

A nivel personal la Teoría de Juegos también ha tenido impacto. Me ha servido

para aprender y profundizar acerca de una rama matemática que desconocía al principio.

Además, el proyecto cumple algunos de los Objetivos de Desarrollo Sostenible de la Agenda 2030:

- **Educación de calidad**. Este proyecto cumple con el punto 4.7 de las metas del objetivo 4. En dicho punto menciona que los alumnos deben adquirir los conocimientos teóricos y prácticos necesarios para promover el desarrollo sostenible. En este trabajo no se utiliza ningún material perjudicial para el medioambiente. Además, el punto 4.7 también menciona la importancia de la educación en la igualdad de género, la promoción de una cultura de paz y no violencia, la valoración de la diversidad cultural y la contribución de la cultura al desarrollo sostenible. Para este proyecto también se cumplen los objetivos mencionados anteriormente, los juegos pueden ser jugados tanto por mujeres como por hombres, pudiendo darse partidas mixtas entre una mujer y un hombre, al igual que no importa la cultura de los jugadores para poder jugar a los juegos. Las diferentes culturas han hecho posible la creación de este proyecto como ya pudimos ver en el capítulo 1 con la introducción.

- **Lograr la igualdad entre los géneros y empoderar a todas las mujeres y las niñas**. Dentro de las metas del objetivo 5 relacionado con la igualdad de género se cumplen los siguientes puntos:

  5.1 "*Poner fin a todas las formas de discriminación contra todas las mujeres y las niñas en todo el mundo*".

  5.2 "*Eliminar todas las formas de violencia contra todas las mujeres y las niñas en los ámbitos público y privado, incluidas la trata y la explotación sexual y otros tipos de explotación*".

  5.5 "*Asegurar la participación plena y efectiva de las mujeres y la igualdad de oportunidades de liderazgo a todos los niveles decisorios en la vida política, económica y pública.*"

  Como ya mencionamos en el punto anterior, en el juego desarrollado en este trabajo no importa el género de los jugadores. En el transcurso de la partida cada jugador tendrá opciones de poder ganar o empatar independientemente de cuál sea el género del jugador.

- **Adoptar medidas urgentes para combatir el cambio climático y sus efectos**. En la meta del objetivo 13.3 se menciona la importancia de mejorar la educación, la sensibilización y la capacidad humana respecto a la mitigación del cambio climático. El trabajo ayuda a esta reducción de efectos del cambio climático porque no necesita el uso de plásticos u otros materiales perjudiciales para el medioambiente.

- **Promover sociedades justas, pacíficas e inclusivas**. El proyecto cumple las metas de reducir todas las formas de violencia y por ende reducir las tasas de mortalidad en todo el mundo, porque toda forma de conocimiento, y

en particular la Teoría de Juegos, fomenta la cooperación entre los pueblos y es una herramienta para el desarrollo.

Resumiendo, el proyecto cumple con los siguientes Objetivos de Desarrollo Sostenible de la Agenda 2030:

Análisis de Impacto	Aplica	Forma que Aplica
Fin de la pobreza	No	-
Hambre cero	No	-
Salud y bienestar	No	-
Educación de calidad	Sí	Cumple el punto 4.7
Igualdad de género	Sí	Cumple los puntos 5.1, 5.2 y 5.5
Agua limpia y saneamiento	No	-
Energía asequible y no contaminante	No	-
Trabajo decente y crecimiento económico	No	-
Industria, innovación e infraestructura	No	-
Reducción de las desigualdades	No	-
Ciudades y comunidades sostenibles	No	-
Producción y consumo responsable	No	-
Acción por el clima	Sí	Cumple el punto 13.3
Vida submarina	No	-
Vida de ecosistemas terrestres	No	-
Paz, justicia e instituciones sólidas	Sí	Cumple los puntos 16.1 y 16.2
Alianzas para lograr los objetivos	No	-

Bibliografía

[1] Aaron N. Siegel, *Combinatorial Game Theory*, Ed. Committee, Providence RI:MSC, 2013.

[2] Aviezri S. Fraenkel and Yacoov Yesha, "The Generalized Sprague-Grundy Function and Its Invariance under Certain Mappings" in *Journal of Combinatorial Theory*, Amsterdam:Netherlands, 1986, pp 165-177.

[3] R. B. Banerji, "Artificial Intelligence - A theoretical Approach", Elsevier/North-Holland, New York, 1980.

[4] A. S. Frankel and Y. Yesha, Theory of annihilation games - I, *J. Combin. Theory Ser. B* **33** (1982), 60-86.

[5] https://www.master-valencia.com/administracion/como-influye-teoria-de-juegos-administracion-empresa/#:~:text=La%20Teoría%20de%20Juegos%20es,diario%20en%20las%20relaciones%20empresariales.

[6] http://www.juntadeandalucia.es/averroes/centros-tic/14002996/helvia/aula/archivos/repositorio/250/271/html/economia/juegos/index.htm

[7] https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Appendices

Anexos

En este capítulo se muestra parte del código utilizado para la implementación del juego.

Clase Tablero

Esta clase implementa el juego: crea el tablero, realiza los movimientos, tanto del jugador como de la CPU, calcula para una determinada situación del juego las posiciones de ventaja o desventaja y muestra el mensaje de quién ha ganado el juego.

```
1  public class Tablero extends JFrame implements MouseListener, ActionListener{
2
3          public Tablero() {
4                  sv = new ArrayList<SituacionVertices>();
5
6                  sv.add(0, new SituacionVertices(casillaVacia, 1, 0, false));
7                  sv.add(1, new SituacionVertices(casillaVacia, 2, 0, false));
8                  sv.add(2, new SituacionVertices(casillaVacia, 3, 1, false));
9                  sv.add(3, new SituacionVertices(casillaVacia, 4, 2, false));
10                 sv.add(4, new SituacionVertices(casillaVacia, 5, 0, false));
11                 sv.add(5, new SituacionVertices(casillaVacia, 6, 0, false));
12                 sv.add(6, new SituacionVertices(casillaVacia, 7, 1, false));
13                 sv.add(7, new SituacionVertices(casillaVacia, 8, 2, false));
14                 sv.add(8, new SituacionVertices(casillaVacia, 9, 2, false));
15                 sv.add(9, new SituacionVertices(casillaVacia, 10, 1, false));
16                 sv.add(10, new SituacionVertices(casillaBlanca, 11, 2, false));
17                 sv.add(11, new SituacionVertices(casillaVacia, 12, 0, false));
18                 sv.add(12, new SituacionVertices(casillaVacia, 13, -1, false));
19                 sv.add(13, new SituacionVertices(casillaVacia, 14, -1, false));
20                 sv.add(14, new SituacionVertices(casillaVacia, 15, -1, false));
21                 sv.add(15, new SituacionVertices(casillaVacia, 16, -1, false));
22                 sv.add(16, new SituacionVertices(casillaVacia, 17, 0, false));
23                 sv.add(17, new SituacionVertices(casillaVacia, 18, 1, false));
24                 sv.add(18, new SituacionVertices(casillaVacia, 19, 1, false));
25                 sv.add(19, new SituacionVertices(casillaVacia, 20, 0, false));
26                 sv.add(20, new SituacionVertices(casillaNegra, 21, 0, false));
27
28                 matrizAdyacencia = inicializarMatrizAdyacencia(matrizAdyacencia);
29
30                 iniciarDetectores();
31                 botonesIniciales();
32                 fichasIniciales();
33         }
34
35         @Override
36         public void mouseClicked(MouseEvent evento) {
37                 if(eligeFichaExtra) {
```

```
38                      if(evento.getSource() == det1) {
39                              rojas.setBounds(210,10,100,100);
40                              tablero.add(rojas);
41                              sv.get(0).setCasilla(false, false, false, true);
42                      }
43                      if(evento.getSource() == det2) {
44                              rojas.setBounds(-25,125,100,100);
45                              tablero.add(rojas);
46                              sv.get(1).setCasilla(false, false, false, true);
47                      }
48                      if(evento.getSource() == det21) {
49                              rojas.setBounds(251,515,30,30);
50                              tablero.add(rojas);
51                              sv.get(20).setCasilla(false, true, false, true);
52                              negras.setBounds(241,515,30,30);
53                      }
54                      eligeFichaExtra=false;
55                      siMasFichas.setVisible(false);
56                      noMasFichas.setVisible(false);
57                      System.out.println("¡Qué empiece el juego!");
58                      int valorSP = calcularSP(sv);
59                      if(valorSP > 0)
60                      System.out.println("El jugador está en posición de ventaja");
61                      else
62                      System.out.println("El jugador está en posición de desventaja");
63              }
64              else {
65
66                      /***Vertice 1***/
67                      if(evento.getSource() == det1) {
68                              if(esPrimerClick(sv)) {
                                                        //Se marca la casilla
69                                      if(sv.get(0).getCasilla()[1]==true)
                                            //Casilla con ficha Negra
70                                      sv.get(0).setCasillaPulsada(true);
71                                      else if(sv.get(0).getCasilla()[2]==true)
                                            //Casilla con ficha Blanca
72                                      sv.get(0).setCasillaPulsada(true);
73                                      else if(sv.get(0).getCasilla()[3]==true)
                                            //Casilla con ficha Roja
74                                      sv.get(0).setCasillaPulsada(true);
75                                      else
                                                                        //No hay
                                            ficha en casilla
76                                      System.out.println("Seleccione una casilla
                                            ocupada por una ficha.");
77                              }
78                              else {
                                                                    //Hay una casilla
                                    previa marcada
79                                      List<Integer> posicionPosibles = posParaMover(
                                            matrizAdyacencia, 0);
80                                      //Buscar la posicion con TRUE
81                                      int posDondeMover = casillaDondeMover(
                                            posicionPosibles, sv);
82                                      if(posDondeMover == 3) {
83                                              //Ficha para mover desde el vertice
                                                    anterior
84                                              int piezaMover = piezaMover(sv.get(
                                                    posDondeMover).getCasilla());
85                                              boolean[] actualizar = actualizarSV(
                                                    piezaMover, sv.get(posDondeMover).
                                                    getCasilla());
86                                              sv.get(posDondeMover).setCasilla(
                                                    actualizar);
87                                              //Actualizar el vertice con la ficha
                                                    movida
```

```
88                                  //Comprobar si era una casilla vacia
89                                  boolean vacio = esVacia(sv.get(0).
                                        getCasilla());
90                                  if(vacio) {
                                                                    //Lo
                                        ponemos sin vacio + la nueva ficha
91                                      switch (piezaMover) {
92                                              case 1:


                                                                    //Movemos
                                                Pieza Negra
93                                              negras.setBounds(210,
                                                    10, 100, 100);
94                                              sv.get(0).setCasilla(
                                                    false, true, false,
                                                    false);
95                                              break;
96                                              case 2:


                                                                    //Movemos
                                                Pieza Blanca
97                                              blancas.setBounds(210,
                                                    10, 100, 100);
98                                              sv.get(0).setCasilla(
                                                    false, false, true,
                                                    false);
99                                              break;
100                                             case 3:


                                                                    //Movemos
                                                Pieza Roja
101                                             rojas.setBounds(210, 10,
                                                     100, 100);
102                                             sv.get(0).setCasilla(
                                                    false, false, false,
                                                     true);
103                                             break;
104                                     }
105                                 }
106                                 else {
107                                     switch (piezaMover) {
108                                             case 1:


                                                                    //Movemos
                                                pieza Negra
109                                             if(sv.get(0).getCasilla
                                                    ()[2]) { //Coincide
                                                Negra y Blanca
110                                                 blancas.
                                                        setBounds
                                                        (205, 10,
                                                        100, 100);
111                                                 negras.setBounds
                                                        (215, 10,
                                                        100, 100);
112                                                 sv.get(0).
                                                        setCasilla(
                                                        false, true,
                                                         true, false
                                                        );
113                                             }
114                                             else if(sv.get(0).
                                                    getCasilla()[3]){ //
```

```
                                        Coincide Negra y
                                        Roja
115                                         negras.setBounds
                                                (205, 10,
                                                100, 100);
116                                         rojas.setBounds
                                                (215, 10,
                                                100, 100);
117                                         sv.get(0).
                                                setCasilla(
                                                false, true,
                                                 false, true
                                                );
118                                     }
119                                     break;
120                                     case 2:


                                                //Movemos
                                        pieza Blanca
121                                     if(sv.get(0).getCasilla
                                        ()[1]) { //Coincide
                                        Blanca y Negra
122                                         blancas.
                                                setBounds
                                                (205, 10,
                                                100, 100);
123                                         negras.setBounds
                                                (215, 10,
                                                100, 100);
124                                         sv.get(0).
                                                setCasilla(
                                                false, true,
                                                 true, false
                                                );
125                                     }
126                                     else if(sv.get(0).
                                        getCasilla()[3]){ //
                                        Coincide Blanca y
                                        Roja
127                                         blancas.
                                                setBounds
                                                (205, 10,
                                                100, 100);
128                                         rojas.setBounds
                                                (215, 10,
                                                100, 100);
129                                         sv.get(0).
                                                setCasilla(
                                                false, false
                                                , true, true
                                                );
130                                     }
131                                     break;
132                                     case 3:


                                                //Movemos
                                        pieza Roja
133                                     if(sv.get(0).getCasilla
                                        ()[1]) { //Coincide
                                        Roja y Negra
134                                         rojas.setBounds
                                                (205, 10,
                                                100, 100);
135                                         negras.setBounds
                                                (215, 10,
```

```
                                                                100, 100);
136                                                         sv.get(0).
                                                                setCasilla(
                                                                false, true,
                                                                 false, true
                                                                );
137                                                 }
138                                                 else if(sv.get(0).
                                                        getCasilla()[2]){ //
                                                        Coincide Roja y
                                                        Blanca
139                                                         blancas.
                                                                setBounds
                                                                (205, 10,
                                                                100, 100);
140                                                         rojas.setBounds
                                                                (215, 10,
                                                                100, 100);
141                                                         sv.get(0).
                                                                setCasilla(
                                                                false, false
                                                                , true, true
                                                                );
142                                                 }
143                                                 break;
144                                             }
145                                         }
146                                         sv.get(posDondeMover).setCasillaPulsada(
                                                false); //Preparamos la casilla para
                                                 nuevos movimientos
147                                         int valorSP = calcularSP(sv);
148                                         if(valorSP > 0)
149                                         System.out.println("La CPU está en
                                                posición de ventaja");
150                                         else
151                                         System.out.println("La CPU está en
                                                posición de desventaja");
152                                         //Llamar al turno de la CPU
153                                         moverCPU(valorSP);
154                                 }
155                                 else {
156                                         System.out.println("Seleccione una
                                                casilla correcta para mover.");
157                                 }
158                         }
159                 }
160
161                 /***Vertice 17***/
162                 if(evento.getSource() == det17) {
163                         boolean ganador = false;
164                         if(esPrimerClick(sv)) {
                                                //Se marca la casilla
165                                 if(sv.get(16).getCasilla()[1]==true)
                                        //Casilla con ficha Negra
166                                 sv.get(16).setCasillaPulsada(true);
167                                 else if(sv.get(16).getCasilla()[2]==true)
                                        //Casilla con ficha Blanca
168                                 sv.get(16).setCasillaPulsada(true);
169                                 else if(sv.get(16).getCasilla()[3]==true)
                                        //Casilla con ficha Roja
170                                 sv.get(16).setCasillaPulsada(true);
171                                 else
                                                                //No hay
                                        ficha en casilla
172                                 System.out.println("Seleccione una casilla
                                        ocupada por una ficha.");
173                         }
```

```
174                         else {
                                                    //Hay una casilla
                                previa marcada
175                             List<Integer> posicionPosibles = posParaMover(
                                    matrizAdyacencia, 16);
176                             //Buscar la posicion con TRUE
177                             int posDondeMover = casillaDondeMover(
                                    posicionPosibles, sv);
178                             if(posDondeMover == 8 || posDondeMover == 17) {
179                                     //Ficha para mover desde el vertice
                                            anterior
180                                     int piezaMover = piezaMover(sv.get(
                                            posDondeMover).getCasilla());
181                                     boolean[] actualizar = actualizarSV(
                                            piezaMover, sv.get(posDondeMover).
                                            getCasilla());
182                                     sv.get(posDondeMover).setCasilla(
                                            actualizar);
183                                     //Actualizar el vertice con la ficha
                                            movida
184                                     //Comprobar si era una casilla vacia
185                                     boolean vacio = esVacia(sv.get(16).
                                            getCasilla());
186                                     if(vacio) {
                                                                    //Lo
                                             ponemos sin vacio + la nueva ficha
187                                             switch (piezaMover) {
188                                                     case 1:


                                                                    //Movemos
                                                         Pieza Negra
189                                                     negras.setBounds(8, 396,
                                                             30, 30);
190                                                     sv.get(16).setCasilla(
                                                             false, true, false,
                                                             false);
191                                                     break;
192                                                     case 2:


                                                                    //Movemos
                                                         Pieza Blanca
193                                                     blancas.setBounds(8,
                                                             396, 30, 30);
194                                                     sv.get(16).setCasilla(
                                                             false, false, true,
                                                             false);
195                                                     break;
196                                                     case 3:


                                                                    //Movemos
                                                         Pieza Roja
197                                                     rojas.setBounds(8, 396,
                                                             30, 30);
198                                                     sv.get(16).setCasilla(
                                                             false, false, false,
                                                              true);
199                                                     break;
200                                             }
201                                     }
202                                     else {
203                                             switch (piezaMover) {
204                                                     case 1:
```

```
                                                    //Movemos
                                                pieza Negra
205                                             if(sv.get(16).getCasilla
                                                    ()[2] && !sv.get(16)
                                                    .getCasilla()[3]) {
                                                    //Coincide Negra y
                                                    Blanca
206                                                     blancas.
                                                            setBounds(3,
                                                             396, 30,
                                                            30);
207                                                     negras.setBounds
                                                            (13, 396,
                                                            30, 30);
208                                                     if(!
                                                            eligeFichaExtra
                                                            ) //solo dos
                                                             fichas
209                                                     ganador = true;
210                                                     sv.get(16).
                                                            setCasilla(
                                                            false, true,
                                                             true, false
                                                            );
211                                             }
212                                             else if(sv.get(16).
                                                    getCasilla()[3] && !
                                                    sv.get(16).
                                                    getCasilla()[2]){ //
                                                    Coincide Negra y
                                                    Roja
213                                                     negras.setBounds
                                                            (3, 396, 30,
                                                             30);
214                                                     rojas.setBounds
                                                            (13, 396,
                                                            30, 30);
215                                                     sv.get(16).
                                                            setCasilla(
                                                            false, true,
                                                             false, true
                                                            );
216                                             }
217                                             else { //Todas las
                                                    fichas coinciden
218                                                     blancas.
                                                            setBounds(3,
                                                             396, 30,
                                                            30);
219                                                     negras.setBounds
                                                            (13, 396,
                                                            30, 30);
220                                                     rojas.setBounds
                                                            (8, 396, 30,
                                                             30);
221                                                     ganador = true;
222                                                     sv.get(16).
                                                            setCasilla(
                                                            false, true,
                                                             true, true)
                                                            ;
223                                             }
224                                             break;
225                                             case 2:


                                                    //Movemos
```

```
                                                    pieza Blanca
226                                             if(sv.get(16).getCasilla
                                                    ()[1] && !sv.get(16)
                                                    .getCasilla()[3]) {
                                                    //Coincide Blanca y
                                                    Negra
227                                                     blancas.
                                                            setBounds(3,
                                                             396, 30,
                                                            30);
228                                                     negras.setBounds
                                                            (13, 396,
                                                            30, 30);
229                                                     if(!
                                                            eligeFichaExtra
                                                            ) //solo dos
                                                             fichas
230                                                     ganador = true;
231                                                     sv.get(16).
                                                            setCasilla(
                                                            false, true,
                                                             true, false
                                                            );
232                                             }
233                                             else if(sv.get(16).
                                                    getCasilla()[3] && !
                                                    sv.get(16).
                                                    getCasilla()[1]){ //
                                                    Coincide Blanca y
                                                    Roja
234                                                     blancas.
                                                            setBounds(3,
                                                             396, 30,
                                                            30);
235                                                     rojas.setBounds
                                                            (13, 396,
                                                            30, 30);
236                                                     sv.get(16).
                                                            setCasilla(
                                                            false, false
                                                            , true, true
                                                            );
237                                             }
238                                             else {
239                                                     blancas.
                                                            setBounds(3,
                                                             396, 30,
                                                            30);
240                                                     negras.setBounds
                                                            (13, 396,
                                                            30, 30);
241                                                     rojas.setBounds
                                                            (8, 396, 30,
                                                             30);
242                                                     ganador = true;
243                                                     sv.get(16).
                                                            setCasilla(
                                                            false, true,
                                                             true, true)
                                                            ;
244                                             }
245                                             break;
246                                             case 3:


                                                            //Movemos
                                                    pieza Roja
```

```
247                                                 if(sv.get(16).getCasilla
                                                        ()[1] && !sv.get(16)
                                                        .getCasilla()[2]) {
                                                        //Coincide Roja y
                                                        Negra
248                                                         rojas.setBounds
                                                                (3, 396, 30,
                                                                 30);
249                                                         negras.setBounds
                                                                (13, 396,
                                                                30, 30);
250                                                         sv.get(16).
                                                                setCasilla(
                                                                false, true,
                                                                 false, true
                                                                );
251                                                 }
252                                                 else if(sv.get(16).
                                                        getCasilla()[2] && !
                                                        sv.get(16).
                                                        getCasilla()[1]){ //
                                                        Coincide Roja y
                                                        Blanca
253                                                         blancas.
                                                                setBounds(3,
                                                                 396, 30,
                                                                30);
254                                                         rojas.setBounds
                                                                (13, 396,
                                                                30, 30);
255                                                         sv.get(16).
                                                                setCasilla(
                                                                false, false
                                                                , true, true
                                                                );
256                                                 }
257                                                 else {
258                                                         blancas.
                                                                setBounds(3,
                                                                 396, 30,
                                                                30);
259                                                         negras.setBounds
                                                                (13, 396,
                                                                30, 30);
260                                                         rojas.setBounds
                                                                (8, 396, 30,
                                                                 30);
261                                                         ganador = true;
262                                                         sv.get(16).
                                                                setCasilla(
                                                                false, true,
                                                                 true, true)
                                                                ;
263                                                 }
264                                                 break;
265                                         }
266                                 }
267                                 if(!ganador) {
268                                         sv.get(posDondeMover).
                                                setCasillaPulsada(false); //
                                                Preparamos la casilla para
                                                nuevos movimientos
269                                         int valorSP = calcularSP(sv);
270                                         if(valorSP > 0)
271                                         System.out.println("La CPU está
                                                en posición de ventaja");
272                                         else
```

```
273                                                 System.out.println("La CPU está
                                                        en posición de desventaja");
274                                                 //Llamar al turno de la CPU
275                                                 moverCPU(valorSP);
276                                         }
277                                         else
278                                         System.out.println("
                                                ************************************\
                                                n"
279                                         + "* ¡Enhorabuena, has ganado la partida
                                                ! *\n"
280                                         + "
                                                ************************************
                                                ");
281                                 }
282                                 else {
283                                         System.out.println("Seleccione una
                                                casilla correcta para mover.");
284                                 }
285                         }
286                 }
287
288                 /***Vertice 21***/
289                 if(evento.getSource() == det21) {
290                         if(esPrimerClick(sv)) {
                                                //Se marca la casilla
291                                 if(sv.get(20).getCasilla()[1]==true)
                                        //Casilla con ficha Negra
292                                 sv.get(20).setCasillaPulsada(true);
293                                 else if(sv.get(20).getCasilla()[2]==true)
                                        //Casilla con ficha Blanca
294                                 sv.get(20).setCasillaPulsada(true);
295                                 else if(sv.get(20).getCasilla()[3]==true)
                                        //Casilla con ficha Roja
296                                 sv.get(20).setCasillaPulsada(true);
297                                 else
                                                                //No hay
                                        ficha en casilla
298                                 System.out.println("Seleccione una casilla
                                        ocupada por una ficha.");
299                         }
300                         else {
301                                 System.out.println("No se puede mover a esta
                                        casilla.");
302                         }
303                 }
304         }
305 }
306
307 public void iniciarDetectores() {
308         ImageIcon imDet1 = new ImageIcon(urlDet);
309         det1 = new JLabel(imDet1);
310         det1.setBounds(210,10,100,100);
311         det1.addMouseListener(this);
312         add(det1);
313
314         ImageIcon imDet2 = new ImageIcon(urlDet);
315         det2 = new JLabel(imDet2);
316         det2.setBounds(-25,130,100,100);
317         det2.addMouseListener(this);
318         add(det2);
319
320         ImageIcon imDet3 = new ImageIcon(urlDet);
321         det3 = new JLabel(imDet3);
322         det3.setBounds(125,130,100,100);
323         det3.addMouseListener(this);
324         add(det3);
```

```
325         }
326
327         public void fichasIniciales() {
328                 ImageIcon imNegras = new ImageIcon(urlNegras);
329                 negras = new JLabel(imNegras);
330                 negras.setBounds(246,515,30,30);
331                 tablero.add(negras);
332
333                 ImageIcon imBlancas = new ImageIcon(urlBlancas);
334                 blancas = new JLabel(imBlancas);
335                 blancas.setBounds(218,303,30,30);
336                 tablero.add(blancas);
337
338                 ImageIcon imRandom = new ImageIcon(urlRandom);
339                 rojas = new JLabel(imRandom);
340                 rojas.setBounds(1500,1500,30,30);
341                 tablero.add(rojas);
342         }
343
344         public void botonesIniciales(){
345                 System.out.println("¿Desea añadir otra ficha al tablero?");
346                 siMasFichas = new JButton("Si");
347                 siMasFichas.setBounds(25, 90, 250, 50);
348                 tablero.add(siMasFichas);
349                 siMasFichas.addActionListener(this);
350
351                 noMasFichas = new JButton("No");
352                 noMasFichas.setBounds(300, 90, 250, 50);
353                 tablero.add(noMasFichas);
354                 noMasFichas.addActionListener(this);
355         }
356
357         @Override
358         public void actionPerformed(ActionEvent e) {
359                 if(e.getSource() == siMasFichas) {
360                         siMasFichas.setText("Elige donde ubicar la nueva ficha");
361                         noMasFichas.setVisible(false);
362                         eligeFichaExtra = true;
363                 }
364                 if(e.getSource() == noMasFichas) {
365                         System.out.println("¡Qué empiece el juego!");
366                         noMasFichas.setVisible(false);
367                         siMasFichas.setVisible(false);
368                         int valorSP = calcularSP(sv);
369                         if(valorSP > 0)
370                         System.out.println("El jugador está en posición de ventaja");
371                         else
372                         System.out.println("El jugador está en posición de desventaja");
373                 }
374         }
375
376         public boolean esPrimerClick(List<SituacionVertices> sv) {
377                 boolean clickado = true;
378                 for(int i=0; i<sv.size() && clickado; i++) {
379                         if(sv.get(i).isCasillaPulsada()==true)
380                         clickado = false;
381                 }
382                 return clickado;
383         }
384
385         public List<Integer> posParaMover(int[][] matriz, int posJ) {
386                 List<Integer> posiciones = new ArrayList<Integer>();
387                 for(int i=0; i<matriz.length; i++) {
388                         if(matriz[i][posJ]==1) {        //posicion para mover
389                                 posiciones.add(i);
390                         }
391                 }
```

```
392                 return posiciones;
393         }
394
395         public int casillaDondeMover(List<Integer> posiciones, List<SituacionVertices>
                sv) {
396                 int pos = -1;
397                 boolean encontrado = false;
398                 for(int i=0; i<posiciones.size() && !encontrado; i++) {
399                         if(sv.get(posiciones.get(i)).isCasillaPulsada()) {
400                                 pos = posiciones.get(i);
401                                 encontrado = true;
402                         }
403                 }
404                 return pos;
405         }
406
407         public int piezaMover(boolean [] situacionVerticeAnterior){
408                 int pos = -1;
409                 boolean salida = false;
410                 //buscamos la ficha para mover
411                 for(int i=0; i<situacionVerticeAnterior.length && !salida; i++) {
412                         if(situacionVerticeAnterior[i]==true) {
413                                 pos = i;
414                                 salida = true;
415                         }
416                 }
417                 return pos;
418         }
419
420         public boolean[] actualizarSV(int pos, boolean[] actualizarV) {
421                 actualizarV[pos]=false; //La ficha a mover no existe en el vertice
                        anterior
422                 //Actualizamos el vertice de donde movemos
423                 boolean esVacio = esVacia(actualizarV);
424                 if(esVacio)
425                 actualizarV[0] = true;
426                 return actualizarV;
427         }
428
429         public boolean esVacia(boolean[] situacionNuevoVertice) {
430                 boolean vacia = false;
431                 if(situacionNuevoVertice[1]!=true
432                 && situacionNuevoVertice[2]!=true
433                 && situacionNuevoVertice[3]!=true) {
434                         vacia = true;
435                 }
436                 return vacia;
437         }
438
439         public int calcularSP(List<SituacionVertices> sv) {
440                 int res = 0;
441                 boolean salida = false;
442                 for(int i=0; i<sv.size() && !salida; i++) {
443                         for(int j=1; j<sv.get(i).getCasilla().length && !salida; j++) {
444                                 if(sv.get(i).getCasilla()[j])
445                                 if(sv.get(i).getEtiquetaV()!=-1) {
446                                         res ^= sv.get(i).getEtiquetaV();
447                                 }
448                                 else {
449                                         res=0;
450                                         salida=true;
451                                 }
452                         }
453                 }
454                 return res;
455         }
456
```

```
457         public int calcularSP(int[] vertice) {
458                 int res = 0;
459                 int i=0;
460                 boolean salida = false;
461                 while(i < 3 && !salida) {
462                         if(vertice[i] != -1)
463                         if(sv.get(vertice[i]).getEtiquetaV()!=-1) {
464                                 res ^= sv.get(vertice[i]).getEtiquetaV();
465                         }
466                         else {
467                                 res=0;
468                                 salida=true;
469                         }
470                         i++;
471                 }
472                 return res;
473         }
474
475         public void moverCPU(int valorSP) {
476                 int[] movimientos1 = new int[21];
477                 int[] movimientos2 = new int[21];
478                 int[] movimientos3 = new int[21];
479                 int vez = 1;
480                 int[] vertice = new int[3];
481                 vertice[2]=-1;          //Por si hay solo 2 fichas
482
483                 String ficha1 = "";
484                 String ficha2 = "";
485                 String ficha3 = "";
486                 //MATRIZ ADYACENCIA LOS MOVIMIENTOS
487                 for(int i=0; i<sv.size(); i++) {
488                         if(!sv.get(i).getCasilla()[0]) { //La casilla no está vacía
489                                 for(int j=0; j<matrizAdyacencia.length; j++) {
490                                         switch (vez) {
491                                                 case 1:
492                                                 vertice[0]=i;
493                                                 movimientos1[j]=matrizAdyacencia[i][j];
494                                                 break;
495                                                 case 2:
496                                                 vertice[1]=i;
497                                                 movimientos2[j]=matrizAdyacencia[i][j];
498                                                 break;
499                                                 case 3:
500                                                 vertice[2]=i;
501                                                 movimientos3[j]=matrizAdyacencia[i][j];
502                                                 break;
503                                         }
504                                 }
505                                 switch (vez) {
506                                         case 1:
507                                         if(sv.get(i).getCasilla()[1]) {
508                                                 ficha1 = "N";
509                                         }
510                                         else if(sv.get(i).getCasilla()[2]) {
511                                                 ficha1 = "B";
512                                         }
513                                         else if(sv.get(i).getCasilla()[3]) {
514                                                 ficha1 = "R";
515                                         }
516                                         break;
517                                         case 2:
518                                         if(sv.get(i).getCasilla()[1]) {
519                                                 ficha2 = "N";
520                                         }
521                                         else if(sv.get(i).getCasilla()[2]) {
522                                                 ficha2 = "B";
523                                         }
```

```
524                                     else if(sv.get(i).getCasilla()[3]) {
525                                             ficha2 = "R";
526                                     }
527                                     break;
528                                     case 3:
529                                     if(sv.get(i).getCasilla()[1]) {
530                                             ficha3 = "N";
531                                     }
532                                     else if(sv.get(i).getCasilla()[2]) {
533                                             ficha3 = "B";
534                                     }
535                                     else if(sv.get(i).getCasilla()[3]) {
536                                             ficha3 = "R";
537                                     }
538                                     break;
539                             }
540                             vez++;
541                     }
542             }
543
544             int vInicialFicha1 = vertice[0];
545             int vInicialFicha2 = vertice[1];
546             int vInicialFicha3 = vertice[2];
547             ArrayList<ValorSP> l_SP1 = new ArrayList<ValorSP>();
548             ArrayList<ValorSP> l_SP2 = new ArrayList<ValorSP>();
549             ArrayList<ValorSP> l_SP3 = new ArrayList<ValorSP>();
550             for(int i=0; i<movimientos1.length; i++) {
551                     if(movimientos1[i]==1) {
552                             vertice[0]=i;
553                             ValorSP valor = new ValorSP(calcularSP(vertice), i,
                                    vInicialFicha1);
554                             l_SP1.add(valor);
555                     }
556             }
557             vertice[0] = vInicialFicha1;
558             for(int i=0; i<movimientos2.length; i++) {
559                     if(movimientos2[i]==1) {
560                             vertice[1]=i;
561                             ValorSP valor = new ValorSP(calcularSP(vertice), i,
                                    vInicialFicha2);
562                             l_SP2.add(valor);
563                     }
564             }
565             vertice[1] = vInicialFicha2;
566             if(vertice[2]!=-1) {
567                     for(int i=0; i<movimientos3.length; i++) {
568                             if(movimientos3[i]==1) {
569                                     vertice[2]=i;
570                                     ValorSP valor = new ValorSP(calcularSP(vertice),
                                            i, vInicialFicha3);
571                                     l_SP3.add(valor);
572                             }
573                     }
574                     vertice[2] = vInicialFicha3;
575             }
576             //Vertices 17, 8 y 16 a tratar
577             boolean encontradoMov = false;
578             int posList = -1;
579             int vElegido = -1;
580
581             //Si SP = 0. Buscamos si hay opción de ir a la meta
582             for(int i = 0; i < l_SP1.size() && !encontradoMov; i++) {
583                     if(l_SP1.get(i).getValorSprague Grundy()==0                //Valor
                            SP=0 y opcion de mover a meta
584                     && l_SP1.get(i).getvDondeMover()==16) {
585                             vElegido = 0;
586                             posList = i;
```

```
587                                 encontradoMov=true;
588                         }
589                 }
590
591                 //Si SP = 0. No hay opción de meta, intentamos evitar 8 y 17
592                 for(int i = 0; i < l_SP1.size() && !encontradoMov; i++) {
593                         if(l_SP1.get(i).getValorSpragueGrundy()==0
594                         && !(l_SP1.get(i).getvDondeMover()==8 ||
595                         l_SP1.get(i).getvDondeMover()==17)){
596                                 vElegido = 0;
597                                 posList = i;
598                                 encontradoMov=true;
599                         }
600                 }
601
602                 //Seguimos con SP = 0 y sin mover, entonces hay que mover a 8 o 17.
603                 for(int i = 0; i < l_SP1.size() && !encontradoMov; i++) {
604                         if(l_SP1.get(i).getValorSpragueGrundy()==0
605                         && (l_SP1.get(i).getvDondeMover()==8 ||
606                         l_SP1.get(i).getvDondeMover()==17)){
607                                 vElegido = 0;
608                                 posList = i;
609                                 encontradoMov=true;
610                         }
611                 }
612
613                 //Caso de SP > 0. Buscamos primero meta
614                 for(int i = 0; i < l_SP1.size() && !encontradoMov; i++) {
615                         if(l_SP1.get(i).getValorSpragueGrundy()>0                //Valor
                                SP>0 y opcion de mover a meta
616                         && l_SP1.get(i).getvDondeMover()==16) {
617                                 vElegido = 0;
618                                 posList = i;
619                                 encontradoMov=true;
620                         }
621                 }
622
623                 //SP > 0 y no hay meta. Intenamos evitar ir a 8 y 17.
624                 for(int i = 0; i < l_SP1.size() && !encontradoMov; i++) {
625                         if(l_SP1.get(i).getValorSpragueGrundy()>0
626                         && !(l_SP1.get(i).getvDondeMover()==8 ||
627                         l_SP1.get(i).getvDondeMover()==17)){
628                                 vElegido = 0;
629                                 posList = i;
630                                 encontradoMov=true;
631                         }
632                 }
633
634                 //SP > 0  y como ultima opcion movemos a 8 o 17
635                 for(int i = 0; i < l_SP1.size() && !encontradoMov; i++) {
636                         if(l_SP1.get(i).getValorSpragueGrundy()>0
637                         && (l_SP1.get(i).getvDondeMover()==8 ||
638                         l_SP1.get(i).getvDondeMover()==17)){
639                                 vElegido = 0;
640                                 posList = i;
641                                 encontradoMov=true;
642                         }
643                 }
644
645                 switch (vElegido) {
646                         case 0:
647                         //Ya sabemos donde hay que mover y la pieza
648                         moverFichaCPU(l_SP1.get(posList).getvInicialFicha(),
649                         l_SP1.get(posList).getvDondeMover(),
650                         ficha1);
651                         break;
652                         case 1:
```

```
653                         //Ya sabemos donde hay que mover y la pieza
654                         moverFichaCPU(l_SP2.get(posList).getvInicialFicha(),
655                         l_SP2.get(posList).getvDondeMover(),
656                         ficha2);
657                         break;
658                         case 2:
659                         //Ya sabemos donde hay que mover y la pieza
660                         moverFichaCPU(l_SP3.get(posList).getvInicialFicha(),
661                         l_SP3.get(posList).getvDondeMover(),
662                         ficha3);
663                         break;
664                 }
665         }
666
667
668         private void moverFichaCPU(int casillaInicial, int casillaFinal, String
                colorPieza) {
669                 boolean casillaFinalVacia = false;
670                 boolean casillaInicialVacia = false;
671                 boolean ganador=false;
672                 switch (casillaInicial) {
673                         case 0:                                         //Vertice 1:
                            Solo movemos a V3
674                         casillaFinalVacia = esVacia(sv.get(2).getCasilla());
675                         if(colorPieza.equals("N")) {
676                                 if(casillaFinalVacia) { //Casilla Vacia
677                                         negras.setBounds(125, 127, 100, 100);
678                                         //Actualizamos los flags de la Casilla
679                                         sv.get(casillaFinal).setCasilla(false, true,
                                            false, false);
680                                         sv.get(casillaInicial).setCasilla(sv.get(
                                            casillaInicial).getCasilla()[0], false, sv.
                                            get(casillaInicial).getCasilla()[2], sv.get(
                                            casillaInicial).getCasilla()[3]);
681                                         casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
682                                         if(casillaInicialVacia)
683                                         sv.get(casillaInicial).setCasilla(true, false,
                                            false, false);
684                                 }
685                                 else {                                  //Casilla
                                    Ocupada
686                                     if(sv.get(2).getCasilla()[2]) {         //
                                        Casilla Ocupada por Blanca
687                                             blancas.setBounds(119, 127, 100, 100);
688                                             negras.setBounds(129, 127, 100, 100);
689                                             sv.get(casillaFinal).setCasilla(false,
                                                true, true, false);
690                                             sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0],
                                                false, false, sv.get(casillaInicial)
                                                .getCasilla()[3]);
691                                             casillaInicialVacia = esVacia(sv.get(
                                                casillaInicial).getCasilla());
692                                             if(casillaInicialVacia)
693                                             sv.get(casillaInicial).setCasilla(true,
                                                false, false, false);
694                                     }
695                                     else if(sv.get(2).getCasilla()[3]) { //Casilla
                                        Ocupada por Roja
696                                             rojas.setBounds(119, 127, 100, 100);
697                                             negras.setBounds(129, 127, 100, 100);
698                                             sv.get(casillaFinal).setCasilla(false,
                                                true, false, true);
699                                             sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0],
                                                false, sv.get(casillaInicial).
```

```
                                        getCasilla()[2], false);
700                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
701                                     if(casillaInicialVacia)
702                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
703                                 }
704                             }
705                         }
706                         else if(colorPieza.equals("B")) {
707                             if(casillaFinalVacia) { //Casilla Vacia
708                                 blancas.setBounds(125, 127, 100, 100);
709                                 //Actualizamos los flags de la Casilla
710                                 sv.get(casillaFinal).setCasilla(false, false,
                                        true, false);
711                                 sv.get(casillaInicial).setCasilla(sv.get(
                                        casillaInicial).getCasilla()[0], sv.get(
                                        casillaInicial).getCasilla()[1], false, sv.
                                        get(casillaInicial).getCasilla()[3]);
712                                 casillaInicialVacia = esVacia(sv.get(
                                        casillaInicial).getCasilla());
713                                 if(casillaInicialVacia)
714                                 sv.get(casillaInicial).setCasilla(true, false,
                                        false, false);
715                             }
716                             else {                                  //Casilla
                                    Ocupada
717                                 if(sv.get(2).getCasilla()[1]) {         //
                                        Casilla Ocupada por Negra
718                                     negras.setBounds(119, 127, 100, 100);
719                                     blancas.setBounds(129, 127, 100, 100);
720                                     sv.get(casillaFinal).setCasilla(false,
                                            true, true, false);
721                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0],
                                            false, false, sv.get(casillaInicial)
                                            .getCasilla()[3]);
722                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
723                                     if(casillaInicialVacia)
724                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
725                                 }
726                                 else if(sv.get(2).getCasilla()[3]) { //Casilla
                                        Ocupada por Roja
727                                     rojas.setBounds(119, 127, 100, 100);
728                                     blancas.setBounds(129, 127, 100, 100);
729                                     sv.get(casillaFinal).setCasilla(false,
                                            false, true, true);
730                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0], sv
                                            .get(casillaInicial).getCasilla()
                                            [1], false, false);
731                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
732                                     if(casillaInicialVacia)
733                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
734                                 }
735                             }
736                         }
737                         else if(colorPieza.equals("R")) {
738                             if(casillaFinalVacia) { //Casilla Vacia
739                                 rojas.setBounds(125, 127, 100, 100);
740                                 //Actualizamos los flags de la Casilla
741                                 sv.get(casillaFinal).setCasilla(false, false,
                                        false, true);
```

```
742                                     sv.get(casillaInicial).setCasilla(sv.get(
                                            casillaInicial).getCasilla()[0], sv.get(
                                            casillaInicial).getCasilla()[1], sv.get(
                                            casillaInicial).getCasilla()[2], false);
743                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
744                                     if(casillaInicialVacia)
745                                     sv.get(casillaInicial).setCasilla(true, false,
                                            false, false);
746                                 }
747                                 else {                                  //Casilla
                                        Ocupada
748                                     if(sv.get(2).getCasilla()[1]) {         //
                                            Casilla Ocupada por Negra
749                                         negras.setBounds(119, 127, 100, 100);
750                                         rojas.setBounds(129, 127, 100, 100);
751                                         sv.get(casillaFinal).setCasilla(false,
                                                true, false, true);
752                                         sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0],
                                                false, sv.get(casillaInicial).
                                                getCasilla()[2], false);
753                                         casillaInicialVacia = esVacia(sv.get(
                                                casillaInicial).getCasilla());
754                                         if(casillaInicialVacia)
755                                         sv.get(casillaInicial).setCasilla(true,
                                                false, false, false);
756                                     }
757                                     else if(sv.get(2).getCasilla()[2]) { //Casilla
                                            Ocupada por Blanca
758                                         blancas.setBounds(119, 127, 100, 100);
759                                         rojas.setBounds(129, 127, 100, 100);
760                                         sv.get(casillaFinal).setCasilla(false,
                                                false, true, true);
761                                         sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0], sv
                                                .get(casillaInicial).getCasilla()
                                                [1], false, false);
762                                         casillaInicialVacia = esVacia(sv.get(
                                                casillaInicial).getCasilla());
763                                         if(casillaInicialVacia)
764                                         sv.get(casillaInicial).setCasilla(true,
                                                false, false, false);
765                                     }
766                                 }
767
768                             }
769                             break;
770
771                             case 8:                                     //Vertice 9.
                                    Movemos a V3 y V17
772                             casillaFinalVacia = esVacia(sv.get(casillaFinal).getCasilla());
773                             if(colorPieza.equals("N")) {
774                                     if(casillaFinalVacia) { //Casilla Vacia
775                                         switch (casillaFinal) {
776                                             case 2: //V3
777                                             negras.setBounds(125, 127, 100, 100);
778                                             break;
779                                             case 16: //V17
780                                             negras.setBounds(8, 396, 30, 30);
781                                             break;
782                                         }
783                                         sv.get(casillaFinal).setCasilla(false, true,
                                                false, false);
784                                         sv.get(casillaInicial).setCasilla(sv.get(
                                                casillaInicial).getCasilla()[0], false, sv.
                                                get(casillaInicial).getCasilla()[2], sv.get(
```

```
                                    casillaInicial).getCasilla()[3]);
785                                 casillaInicialVacia = esVacia(sv.get(
                                        casillaInicial).getCasilla());
786                                 if(casillaInicialVacia)
787                                 sv.get(casillaInicial).setCasilla(true, false,
                                        false, false);
788                             }
789                             else {                                  //Casilla
                                    Ocupada
790                                 if(sv.get(casillaFinal).getCasilla()[2] && !sv.
                                        get(casillaFinal).getCasilla()[3]) {
                                            //Casilla Ocupada por Blanca
791                                     switch (casillaFinal) {
792                                             case 2: //V3
793                                             blancas.setBounds(119, 127, 100,
                                                  100);
794                                             negras.setBounds(129, 127, 100,
                                                 100);
795                                             break;
796                                             case 16:        //V17
797                                             blancas.setBounds(3, 396, 30,
                                                 30);
798                                             negras.setBounds(13, 396, 30,
                                                 30);
799                                             if(!eligeFichaExtra) { //hay dos
                                                  fichas?
800                                                 ganador = true;
801                                             }
802                                             break;
803                                     }
804                                     sv.get(casillaFinal).setCasilla(false,
                                            true, true, false);
805                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0],
                                            false, false, sv.get(casillaInicial)
                                            .getCasilla()[3]);
806                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
807                                     if(casillaInicialVacia)
808                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
809                                 }
810                                 else if(sv.get(casillaFinal).getCasilla()[3] &&
                                         !sv.get(casillaFinal).getCasilla()[2]) { //
                                        Casilla Ocupada por Roja
811                                     switch (casillaFinal) {
812                                             case 2: //V3
813                                             rojas.setBounds(119, 127, 100,
                                                 100);
814                                             negras.setBounds(129, 127, 100,
                                                 100);
815                                             break;
816                                             case 16:        //V17
817                                             rojas.setBounds(3, 396, 30, 30);
818                                             negras.setBounds(13, 396, 30,
                                                 30);
819                                             break;
820                                     }
821                                     sv.get(casillaFinal).setCasilla(false,
                                            true, false, true);
822                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0],
                                            false, sv.get(casillaInicial).
                                            getCasilla()[2], false);
823                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
824                                     if(casillaInicialVacia)
```

```
825                                         sv.get(casillaInicial).setCasilla(true,
                                                false, false, false);
826                                 }
827                                 else { //Roja y Blanca en casilla
828                                         rojas.setBounds(3, 396, 30, 30);
829                                         negras.setBounds(13, 396, 30, 30);
830                                         blancas.setBounds(8, 396, 30, 30);
831                                         ganador = true;
832                                         sv.get(casillaFinal).setCasilla(false,
                                                true, true, true);
833                                         sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0],
                                                false, false, false);
834                                         casillaInicialVacia = esVacia(sv.get(
                                                casillaInicial).getCasilla());
835                                         if(casillaInicialVacia)
836                                         sv.get(casillaInicial).setCasilla(true,
                                                false, false, false);
837                                 }
838                         }
839                 }
840                 else if(colorPieza.equals("B")) {
841                         if(casillaFinalVacia) { //Casilla Vacia
842                                 switch (casillaFinal) {
843                                         case 2: //V3
844                                         blancas.setBounds(125, 127, 100, 100);
845                                         break;
846                                         case 16: //V17
847                                         blancas.setBounds(8, 396, 30, 30);
848                                         break;
849                                 }
850                                 sv.get(casillaFinal).setCasilla(false, false,
                                        true, false);
851                                 sv.get(casillaInicial).setCasilla(sv.get(
                                        casillaInicial).getCasilla()[0], sv.get(
                                        casillaInicial).getCasilla()[1], false, sv.
                                        get(casillaInicial).getCasilla()[3]);
852                                 casillaInicialVacia = esVacia(sv.get(
                                        casillaInicial).getCasilla());
853                                 if(casillaInicialVacia)
854                                 sv.get(casillaInicial).setCasilla(true, false,
                                        false, false);
855                         }
856                         else {                                  //Casilla
                                Ocupada
857                                 if(sv.get(casillaFinal).getCasilla()[1] && !sv.
                                        get(casillaFinal).getCasilla()[3]) {     //
                                        Casilla Ocupada por Negra
858                                         switch (casillaFinal) {
859                                                 case 2: //V3
860                                                 negras.setBounds(119, 127, 100,
                                                        100);
861                                                 blancas.setBounds(129, 127, 100,
                                                         100);
862                                                 break;
863                                                 case 16:        //V17
864                                                 negras.setBounds(3, 396, 30, 30)
                                                        ;
865                                                 blancas.setBounds(13, 396, 30,
                                                        30);
866                                                 if(!eligeFichaExtra) { //hay dos
                                                         fichas?
867                                                         ganador = true;
868                                                 }
869                                                 break;
870                                         }
871                                         sv.get(casillaFinal).setCasilla(false,
```

```
                                        true, true, false);
872                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0],
                                            false, false, sv.get(casillaInicial)
                                            .getCasilla()[3]);
873                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
874                                     if(casillaInicialVacia)
875                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
876                                 }
877                                 else if(sv.get(casillaFinal).getCasilla()[3] &&
                                        !sv.get(casillaFinal).getCasilla()[1]) {
                                             //Casilla Ocupada por Roja
878                                     switch (casillaFinal) {
879                                             case 2: //V3
880                                             rojas.setBounds(119, 127, 100,
                                                 100);
881                                             blancas.setBounds(129, 127, 100,
                                                  100);
882                                             break;
883                                             case 16:        //V17
884                                             blancas.setBounds(3, 396, 30,
                                                 30);
885                                             rojas.setBounds(13, 396, 30, 30)
                                                 ;
886                                             break;
887                                     }
888                                     sv.get(casillaFinal).setCasilla(false,
                                            false, true, true);
889                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0], sv
                                            .get(casillaInicial).getCasilla()
                                            [1], false, false);
890                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
891                                     if(casillaInicialVacia)
892                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
893                                 }
894                                 else { //Negra y Roja en casilla
895                                     rojas.setBounds(3, 396, 30, 30);
896                                     negras.setBounds(13, 396, 30, 30);
897                                     blancas.setBounds(8, 396, 30, 30);
898                                     ganador = true;
899                                     sv.get(casillaFinal).setCasilla(false,
                                            true, true, true);
900                                     sv.get(casillaInicial).setCasilla(sv.get
                                            (casillaInicial).getCasilla()[0],
                                            false, false, false);
901                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
902                                     if(casillaInicialVacia)
903                                     sv.get(casillaInicial).setCasilla(true,
                                            false, false, false);
904                                 }
905                             }
906                         }
907                         else if(colorPieza.equals("R")) {
908                             if(casillaFinalVacia) { //Casilla Vacia
909                                     switch (casillaFinal) {
910                                             case 2: //V3
911                                             rojas.setBounds(125, 127, 100, 100);
912                                             break;
913                                             case 16: //V17
914                                             rojas.setBounds(8, 396, 30, 30);
915                                             break;
```

```
916                                     }
917                                     sv.get(casillaFinal).setCasilla(false, false,
                                            false, true);
918                                     sv.get(casillaInicial).setCasilla(sv.get(
                                            casillaInicial).getCasilla()[0], sv.get(
                                            casillaInicial).getCasilla()[1], sv.get(
                                            casillaInicial).getCasilla()[2], false);
919                                     casillaInicialVacia = esVacia(sv.get(
                                            casillaInicial).getCasilla());
920                                     if(casillaInicialVacia)
921                                     sv.get(casillaInicial).setCasilla(true, false,
                                            false, false);
922                                 }
923                                 else {                                  //Casilla
                                        Ocupada
924                                     if(sv.get(casillaFinal).getCasilla()[1] && !sv.
                                            get(casillaFinal).getCasilla()[2]) {     //
                                            Casilla Ocupada por Negra
925                                         switch (casillaFinal) {
926                                                 case 2: //V3
927                                                 negras.setBounds(119, 127, 100,
                                                        100);
928                                                 rojas.setBounds(129, 127, 100,
                                                        100);
929                                                 break;
930                                                 case 16:        //V17
931                                                 negras.setBounds(3, 396, 30, 30)
                                                        ;
932                                                 rojas.setBounds(13, 396, 30, 30)
                                                        ;
933                                                 break;
934                                         }
935                                         sv.get(casillaFinal).setCasilla(false,
                                                true, false, true);
936                                         sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0],
                                                false, sv.get(casillaInicial).
                                                getCasilla()[2], false);
937                                         casillaInicialVacia = esVacia(sv.get(
                                                casillaInicial).getCasilla());
938                                         if(casillaInicialVacia)
939                                         sv.get(casillaInicial).setCasilla(true,
                                                false, false, false);
940                                     }
941                                     else if(sv.get(casillaFinal).getCasilla()[2] &&
                                            !sv.get(casillaFinal).getCasilla()[1]) {
                                                //Casilla Ocupada por Blanca
942                                         switch (casillaFinal) {
943                                                 case 2: //V3
944                                                 blancas.setBounds(119, 127, 100,
                                                        100);
945                                                 rojas.setBounds(129, 127, 100,
                                                        100);
946                                                 break;
947                                                 case 16:        //V17
948                                                 blancas.setBounds(3, 396, 30,
                                                        30);
949                                                 rojas.setBounds(13, 396, 30, 30)
                                                        ;
950                                                 break;
951                                         }
952                                         sv.get(casillaFinal).setCasilla(false,
                                                false, true, true);
953                                         sv.get(casillaInicial).setCasilla(sv.get
                                                (casillaInicial).getCasilla()[0], sv
                                                .get(casillaInicial).getCasilla()
                                                [1], false, false);
```

```
954                                             casillaInicialVacia = esVacia(sv.get(
                                                    casillaInicial).getCasilla());
955                                             if(casillaInicialVacia)
956                                             sv.get(casillaInicial).setCasilla(true,
                                                    false, false, false);
957                                     }
958                                     else {
959                                             rojas.setBounds(3, 396, 30, 30);
960                                             negras.setBounds(13, 396, 30, 30);
961                                             blancas.setBounds(8, 396, 30, 30);
962                                             ganador = true;
963                                             sv.get(casillaFinal).setCasilla(false,
                                                    true, true, true);
964                                             sv.get(casillaInicial).setCasilla(sv.get
                                                    (casillaInicial).getCasilla()[0],
                                                    false, false, false);
965                                             casillaInicialVacia = esVacia(sv.get(
                                                    casillaInicial).getCasilla());
966                                             if(casillaInicialVacia)
967                                             sv.get(casillaInicial).setCasilla(true,
                                                    false, false, false);
968                                     }
969                             }
970                     }
971                     break;
972             }
973             if(!ganador) {
974                     int valorSP = calcularSP(sv);
975                     if(valorSP > 0)
976                     System.out.println("El jugador está en posición de ventaja");
977                     else
978                     System.out.println("El jugador está en posición de desventaja");
979             }
980             else
981             System.out.println("La CPU ha ganado la partida.");
982     }
983
984
985     public int[][] inicializarMatrizAdyacencia(int [][] matriz) {
986             matriz = new int[21][21];
987             for(int i=0; i<matriz.length; i++) {
988                     for(int j=0; j<matriz.length; j++) {
989                             switch (i) {
990                                     case 0:
991                                     if(j==2) {
992                                             matriz[i][j]=1;
993                                     }
994                                     break;
995                                     case 1:
996                                     if(j==8) {
997                                             matriz[i][j]=1;
998                                     }
999                                     break;
1000                                    case 2:
1001                                    if(j==1) {
1002                                            matriz[i][j]=1;
1003                                    }
1004                                    if(j==5) {
1005                                            matriz[i][j]=1;
1006                                    }
1007                                    break;
1008                                    case 3:
1009                                    if(j==0) {
1010                                            matriz[i][j]=1;
1011                                    }
1012                                    if(j==9) {
1013                                            matriz[i][j]=1;
```

```
1014                                    }
1015                                    break;
1016                                    case 4:
1017                                    if(j==3) {
1018                                            matriz[i][j]=1;
1019                                    }
1020                                    break;
1021                                    case 5:
1022                                    if(j==3) {
1023                                            matriz[i][j]=1;
1024                                    }
1025                                    break;
1026                                    case 6:
1027                                    if(j==5) {
1028                                            matriz[i][j]=1;
1029                                    }
1030                                    if(j==8) {
1031                                            matriz[i][j]=1;
1032                                    }
1033                                    break;
1034                                    case 7:
1035                                    if(j==5) {
1036                                            matriz[i][j]=1;
1037                                    }
1038                                    if(j==9) {
1039                                            matriz[i][j]=1;
1040                                    }
1041                                    break;
1042                                    case 8:
1043                                    if(j==2) {
1044                                            matriz[i][j]=1;
1045                                    }
1046                                    if(j==16) {
1047                                            matriz[i][j]=1;
1048                                    }
1049                                    break;
1050                                    case 9:
1051                                    if(j==4) {
1052                                            matriz[i][j]=1;
1053                                    }
1054                                    break;
1055                                    case 10:
1056                                    if(j==6) {
1057                                            matriz[i][j]=1;
1058                                    }
1059                                    if(j==11) {
1060                                            matriz[i][j]=1;
1061                                    }
1062                                    break;
1063                                    case 11:
1064                                    if(j==7) {
1065                                            matriz[i][j]=1;
1066                                    }
1067                                    break;
1068                                    case 12:
1069                                    if(j==14) {
1070                                            matriz[i][j]=1;
1071                                    }
1072                                    break;
1073                                    case 13:
1074                                    if(j==12) {
1075                                            matriz[i][j]=1;
1076                                    }
1077                                    break;
1078                                    case 14:
1079                                    if(j==6) {
1080                                            matriz[i][j]=1;
```

```
1081                                                    }
1082                                                    if(j==15) {
1083                                                            matriz[i][j]=1;
1084                                                    }
1085                                                    break;
1086                                                    case 15:
1087                                                    if(j==7) {
1088                                                            matriz[i][j]=1;
1089                                                    }
1090                                                    if(j==13) {
1091                                                            matriz[i][j]=1;
1092                                                    }
1093                                                    if(j==18) {
1094                                                            matriz[i][j]=1;
1095                                                    }
1096                                                    break;
1097                                                    case 16:
1098                                                    break;
1099                                                    case 17:
1100                                                    if(j==14) {
1101                                                            matriz[i][j]=1;
1102                                                    }
1103                                                    if(j==16) {
1104                                                            matriz[i][j]=1;
1105                                                    }
1106                                                    break;
1107                                                    case 18:
1108                                                    if(j==19) {
1109                                                            matriz[i][j]=1;
1110                                                    }
1111                                                    break;
1112                                                    case 19:
1113                                                    if(j==9) {
1114                                                            matriz[i][j]=1;
1115                                                    }
1116                                                    break;
1117                                                    case 20:
1118                                                    if(j==17) {
1119                                                            matriz[i][j]=1;
1120                                                    }
1121                                                    if(j==18) {
1122                                                            matriz[i][j]=1;
1123                                                    }
1124                                                    break;
1125                                            }
1126                                    }
1127                            }
1128                            return matriz;
1129                    }
1130 }
```

Clase SituacionVertices

Esta clase implementa el tipo de "SituacionVertices" que se utiliza en algunas estructuras de datos de la clase "Tablero", para que el jugador y la máquina puedan realizar los movimientos solicitados.

```
1  public class SituacionVertices {
2          private boolean[] casilla;      //{Vacia, Negra, Blanca, Roja}
3          private int numV;
4          private int etiquetaV;
5          private boolean casillaPulsada;
6
7          public SituacionVertices(boolean[] casilla, int numV,
8          int etiquetaV, boolean casillaPulsada) {
9                  this.casilla = casilla;
10                 this.numV = numV;
11                 this.etiquetaV = etiquetaV;
12                 this.casillaPulsada = casillaPulsada;
13         }
14
15         /* Getters y Setters */
16         public boolean isCasillaPulsada() {
17                 return casillaPulsada;
18         }
19
20         public void setCasillaPulsada(boolean casillaPulsada) {
21                 this.casillaPulsada = casillaPulsada;
22         }
23
24         public boolean[] getCasilla() {
25                 return casilla;
26         }
27
28         public int getNumV() {
29                 return numV;
30         }
31
32         public int getEtiquetaV() {
33                 return etiquetaV;
34         }
35
36         public void setCasilla(boolean[] casilla) {
37                 this.casilla = casilla;
38         }
39
40         public void setCasilla(boolean vacio, boolean negra, boolean blanca, boolean
               roja) {
41                 boolean[] set = new boolean[4];
42                 set[0]=vacio;
43                 set[1]=negra;
44                 set[2]=blanca;
45                 set[3]=roja;
46                 this.casilla = set;
47         }
48
49         public void setNumV(int numV) {
50                 this.numV = numV;
51         }
52
53         public void setEtiquetaV(int etiquetaV) {
54                 this.etiquetaV = etiquetaV;
55         }
56
57 }
```

Clase ValorSP

Esta clase implementa el tipo de "ValorSP" que se utiliza en algunas estructuras de datos de la clase "Tablero". Con esta clase la CPU es capaz de analizar a qué vértice es mejor mover.

```
1   public class ValorSP {
2
3       private int valorSpragueGrundy;
4       private int vDondeMover;
5       private int vInicialFicha;
6       public ValorSP(int valorSpragueGrundy, int vDondeMover, int
            vInicialFicha) {
7           this.valorSpragueGrundy = valorSpragueGrundy;
8           this.vDondeMover = vDondeMover;
9           this.vInicialFicha = vInicialFicha;
10      }
11
12      /* Getters y Setters */
13      public int getValorSpragueGrundy() {
14          return valorSpragueGrundy;
15      }
16
17      public int getvDondeMover() {
18          return vDondeMover;
19      }
20
21      public int getvInicialFicha() {
22          return vInicialFicha;
23      }
24
25      public void setValorSpragueGrundy(int valorSpragueGrundy) {
26          this.valorSpragueGrundy = valorSpragueGrundy;
27      }
28
29      public void setvDondeMover(int vDondeMover) {
30          this.vDondeMover = vDondeMover;
31      }
32
33      public void setvInicialFicha(int vInicialFicha) {
34          this.vInicialFicha = vInicialFicha;
35      }
36
37  }
```

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jul 01 16:51:14 CEST 2021
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)