Heurística Para Jugar al dominó

Bruno Gil Ramírez, Author Name
Luis Alberto Zacarias Daniel
Williams Gustavo González Sandoval
Benemérita Universidad Autónoma de Puebla
Heroica Puebla de Zaragoza

Abstract
En este reporte se presenta el diseño e implementación de un algoritmo que juega al dominó de forma inteligente. El algoritmo utiliza una estrategia de búsqueda basada en el criterio de mejor primero, que consiste en evaluar las fichas disponibles y seleccionar la que maximiza el puntaje. Para ello, el algoritmo genera una lista con todas las fichas que son compatibles con los extremos del dominó y les asigna un valor según una función heurística. Luego, el algoritmo escoge la ficha de mayor valor y la coloca en el extremo correspondiente. El objetivo del algoritmo es ganar la partida o minimizar la diferencia de puntos con el oponente. Se realizan pruebas experimentales para medir el rendimiento del algoritmo y se comparan los resultados con otros algoritmos existentes. En este documento se hablara de su diseño e implantación.

Contents

1 Planteamiento del problema

El Dominó es un juego de mesa muy común en la vida del humano moderno, y muchas veces se da por sentado. Sin embargo, este juego tiene una larga historia que se remonta a la antigua China, y ha sido fuente de diversión y desafío para generaciones de jugadores.

1.1 El juego del dominó

El primer domino del que se tiene registro es del año 1232, cuando se menciona en un texto chino de la dinastía Song llamado Antiguos Eventos en Wulin. Sin embargo, los dominós modernos aparecieron en Italia durante el siglo XVIII, y se diferencian de los dominós chinos en varios aspectos, como el número y la disposición de los puntos. No hay una conexión confirmada entre los dos tipos de dominós. El nombre "domino" probablemente se deriva de la semejanza con un tipo de disfraz de carnaval usado durante el Carnaval de Venecia, que consistía en una túnica con capucha negra y una máscara blanca . Los dominós europeos se fabricaban tradicionalmente con hueso, marfil, ébano u otras maderas duras, con puntos contrastantes de color blanco o negro. Los dominós se han utilizado para jugar diversos juegos de mesa, como el Whist, el Matador y el Muggins, así como para la adivinación y el arte de derribar dominós[1] .

1.2 ¿Cómo una computadora puede jugar a un Juego?

Primero se tiene que tomar en cuenta la heurística que le permite evaluar el estado del juego y las posibles acciones que puede realizar.

1.2.1 ¿Qué es la heurística?

La heurística es una función que asigna un valor numérico a cada estado o acción, indicando cuán favorable o desfavorable es para la computadora. Luego se tiene que implementar un algoritmo que explore el árbol de búsqueda del juego, es decir, las distintas secuencias de movimientos que pueden ocurrir a partir de una situación inicial. El algoritmo debe seleccionar la acción que maximice el valor de la heurística, considerando también las respuestas del oponente. Así, la computadora puede jugar a un juego de forma inteligente y estratégica.

---

2 Metodología

2.1 Estados inicial y final

En el juego de domino se reparten n numero de fichas a n numero de jugadores, en algunos casos la fichas se reparten en su totalidad sin sobrar una, en nuestro caso, en el programa se determinan dos jugadores, la computadora o maquina y el usuario, por lo que las fichas se reparten entre ambos jugadores. Nuestros estados iniciales se trataran de tres arreglos, estos serian:

1. `fichas_jugador_1`: que será donde estarán las fichas del jugador 1.
2. `fichas_jugador_2`: que será donde estarán las fichas del jugador 2.
3. `tablero`: donde estarán las fichas que ya se hayan jugado.

En este caso hay dos posibles escenarios de con puede terminar el estado final, la mas sencilla es cuando, ya sea el jugador 1 o jugador 2 no tengan ninguna ficha disponible, entonces como no tiene ninguna ficha se dice que este es el ganador, el otro es cuando ya no hay posibles movimientos, es decir, supongamos que tenemos el siguiente ejemplo con el tablero [(6,2), ..., (0,6)], y digamos que ningún jugador tiene una ficha que corresponda al numero 6, por lo que el ganador sería aquel que tenga menos fichas, supongamos que el jugador 1 tiene 3 y el jugador 2 tiene 1, entonces decimos que el ganador es el jugador 2.

2.2 Función Objetivo

La función objetivo determinará cuando termina el programa por medio de ciertas condiciones, la primera es cuando el jugador 1 no tiene fichas, entones gana el jugador 1, pasa exactamente lo mismo si el jugador 2 es quien no tiene fichas disponibles y el otro caso es cuando no tiene fichas por jugar, entonces gana el jugador que tenga menos fichas disponibles.

2.3 Función Sucesor

La función sucesor se encarga de retornar una lista de fichas posibles, esta lista de fechas se ira llenando dependiendo de que condición cumplan, es decir, con ayuda de las fichas de ambos extremos veremos que posibles fichas necesitaremos ya sea que un numero esta de ambos lados o si son diferentes para así determinar que fichas serían las indicadas para retornarlas en una lista.

2.4 La función Heurística

La funci´on heur´ıstica esta basad en la búsqueda best first, esta función se encarga de retornar la primera solución que encuentre en este caso hablamos de que retornara una ficha, que por medio de la suma de sus lados se deteminará cual es la mejor.

2.5 Orden de Complejidad

El orden de complejidad del método `jugar` de la clase `jugador_Maquina` se puede calcular analizando el número de operaciones que realiza el código. El método `jugar` llama a los métodos `actualizar_fichas` y `actualizar_tablero`, que tienen una complejidad constante $O(1)$, ya que solo asignan un valor a un atributo. Luego, el método `jugar` comprueba si la lista de fichas está vacía, lo que también es una operación constante $O(1)$. Después, el método `jugar` recorre la lista de fichas con un bucle `for` y verifica si cada ficha es compatible con los extremos del tablero, lo que implica dos comparaciones por cada ficha. Esto tiene una complejidad lineal $O(n)$, donde n es el número de fichas. Finalmente, el método `jugar` llama al método `mejor_ficha`, que también recorre la lista de fichas con un bucle `for` y busca la ficha con mayor suma de sus valores. Esto tiene una complejidad lineal $O(n)$ también. Por lo tanto, el orden de complejidad del método `jugar` es $O(1) + O(n) + O(n) = O(n)$, ya que el término lineal domina sobre el término constante.

3 Diseño y descripción del algoritmo

El programa empieza creando un objeto de la clase Domino, lo que hace esta son solo tres cosas, generar las fichas, repartir las fichas en las variables `fichas_jugador_1` y `fichas_jugador_2`, finalmente lo que hará es retornar estas dos variables.

Una vez teniendo estas dos variables lo que hacemos es mandarlas como argumentos a un nuevo objeto creado de la clase `Juego_Domino`, en esta parte se reciben esos argumentos y se asignan a nuevas variables con el mismo nombre, también se declaran variables como tablero y turno, después definimos varias funciones como: `decidir_turno`, `determinar_ganador`, `string_to_tuple` y `jugar` que es la mas importante y además llama a todas las funciones anteriores, en esta función jugar, lo primero que hacemos es definir una variable maquina1 de la clase `jugador_Maquina` ya que esta sera el jugador de maquina, este objeto tiene varias funciones como: `actualizar_tablero`, `actualizar_fichas`, `mejor_ficha`, `fichas_posibles` y `jugar`, cuando creamos un objeto `maquina1` que le mandamos argumentos como `fichas_jugador_2` y `tablero`, después definimos el turno con la función `decidir_turno` que decidirá quien empezara dependiendo de quien tenga la ficha (6,6).

Una vez decidido entramos a un ciclo while dentro de la función `jugar` que esta dentro de la clase `Juego_domino`, en este ciclo se estará ejecutando hasta que la función objetivo que es `determinar_ganador` retorne a una ganador, en caso contrario por medio de un condicional con la variable turno, sabremos de quien es el turno, si el turno es igual a 1 entonces es turno del usuario, lo que hacemos es mostrarle sus fichas y después el inserta una adecuada según sea necesaria en el juego, en caso de que esta no exista pasa a ser turno de la maquina, en caso de que si exista, lo que hacemos es buscar posicionarla ya sea del lado izquierdo o del lado derecho y podemos invertirla la ficha para colocarla si es que ser necesario.

Para el jugador 2 que sera un objeto llamado `maquina1` de la clase `maquina` es algo parecido pero con varios cambios, ya que el mismo programa decidirá por si mismo que ficha debe elegir, como creamos un objeto maquina1 que tiene el tablero y tiene las fichas de este jugador solo llamaremos a su función jugar, esta función actualiza tanto las fichas del tablero como las fichas de este jugador, determinamos los valores de las fichas que están del lado izquierdo y del lado derecho, una vez que tengamos estos dos extremos, llamamos la función `fichas_posibles` que es nuestra función sucesor, esta función se encarga de retornar todas las fichas posibles que podrían encajar, para finalizar en la función `jugar`, del objeto `maquina1` esta con ayuda de una función llamada `mejor_ficha` que es nuestra función heurística y que se basa en el algoritmo best first, retornara lo que podría se la mejor ficha para jugar, ya que en esta función mandamos el resultado que nos retorna `fichas_posibles` y también las fichas que tiene este jugador, por lo que así sabrá cual seria la mejor ficha para jugar, en caso de que esta exista en las fichas que tiene este jugador.

Para terminar podemos decir que el ciclo while, no terminará hasta que tenga a un ganador, que esta es nuestra función objetivo, en caso de que uno no tenga fichas ese sera el ganador o en caso de que ambos tengan ficha pero ya no se pueda jugar debido a que no tienen las fichas para seguir el juego, la función se basara en quien tenga menos fichas y así determinar al ganador.

4 Implementación

4.1 Clase Domino

Esta clase define el comportamiento y los atributos de un objeto de tipo Domino, que representa el juego de dominó. El constructor de la clase inicializa el atributo fichas, que es una lista de tuplas que contienen los valores de cada ficha del juego. El método generar_fichas devuelve una copia de la lista de fichas, para evitar modificar el atributo original. El método repartir_fichas crea dos listas vacías para almacenar las fichas de cada jugador, genera las fichas del juego, las mezcla aleatoriamente y las reparte alternadamente entre los dos jugadores. Finalmente, devuelve las dos listas con las fichas de cada jugador.

```
class Domino:
    def __init__(self): #constructor de la clase Domino
        #genera las fichas del juego
        self.fichas = [(i, j) for i in range(7) for j in range(i, 7)]

    def generar_fichas(self): #genera las fichas del juego
        return self.fichas.copy() #retorna las fichas del juego
    #reparte las fichas a los jugadores
    def repartir_fichas(self):
        fichas_jugador_1 = []
        fichas_jugador_2 = []
        #se generan las fichas del juego
        fichas_disponibles = self.generar_fichas()
        #se mezclan las fichas del juego
        random.shuffle(fichas_disponibles)

        while len(fichas_disponibles) > 0:
            #se reparten las fichas al jugador 1
            fichas_jugador_1.append(fichas_disponibles.pop())
            #se reparten las fichas al jugador 2
            fichas_jugador_2.append(fichas_disponibles.pop())

        return fichas_jugador_1, fichas_jugador_2
```

4.2 La Clase Juego_Domino

Esta clase Administra el juego del domino, una vez que se inicializo el juego del domino generando las 28 fichas del juego, y se repartieron las fichas de forma aleatoria, la función Jugar es la que ejecuta el juego y media la respuesta del usuario con la del algoritmo de búsqueda ciega para generar la contra jugada del jugador maquina.

```
    class Juego_Domino:
    def __init__(self, fichas_jugador_1, fichas_jugador_2): #constructor de
       la clase Juego_Domino
        self.tablero = []
```

```
        self.fichas_jugador_1 = fichas_jugador_1 #fichas del jugador 1
        self.fichas_jugador_2 = fichas_jugador_2 #fichas del jugador 2
        self.ficha_central = (6, 6) #ficha central del juego
        self.turno = 0 #turno del juego
```

4.2.1 El método determinar_turno

Dependiendo de la ronda, se escoge la mula mas grande, en el caso de la primera, pues se inicia siempre con la mula 6,6. Dependiendo de quien tiene la ficha mas grande, es el numero que se devuelve.

```
    def decidir_turno(self):
        if  self.ficha_central in self.fichas_jugador_1: #si la ficha
            central esta en las fichas del jugador 1
            return 1 #retorna 1
        elif self.ficha_central in self.fichas_jugador_2: #si la ficha
            central esta en las fichas del jugador 2
            return 2 #retorna 2
```

4.2.2 El método determinar_ganador

Este método define la lógica para determinar el ganador del juego de dominó. Recibe como parámetros el estado de la mesa, las fichas de cada jugador y el número de veces que se ha saltado el turno. Devuelve el número del jugador ganador o None si el juego no ha terminado.

El método comprueba si alguno de los jugadores se ha quedado sin fichas, lo que significa que ha ganado el juego. En caso contrario, verifica si se han saltado dos turnos seguidos, lo que implica que ninguno de los jugadores puede colocar una ficha en la mesa. En ese caso, se calcula la suma de los valores de las fichas de cada jugador y se declara ganador al que tenga la menor suma. Si ninguna de estas condiciones se cumple, el método devuelve None para indicar que el juego debe continuar.

```
    def determinar_ganador(self,mesa, fichas_jugador_1, fichas_jugador_2,
        skip): #determina el ganador del juego
        if len(fichas_jugador_1) == 0:  #si el jugador 1 no tiene fichas,
            gana el jugador 1
            return 1
        elif len(fichas_jugador_2) == 0: #si el jugador 2 no tiene fichas,
            gana el jugador 2
            return 2
        elif skip>=2: #si los dos jugadores no pueden jugar, gana el jugador
            con menos fichas
            suma_fichas_jugador_1 = sum(sum(ficha) for ficha in
                fichas_jugador_1) #suma las fichas del jugador 1
            suma_fichas_jugador_2 = sum(sum(ficha) for ficha in
                fichas_jugador_2) #suma las fichas del jugador 2
            if suma_fichas_jugador_1 < suma_fichas_jugador_2: #si la suma de
                las fichas del jugador 1 es menor a la suma de las fichas
                del jugador 2, gana el jugador 1
                return 1
            else:
                return 2 #si la suma de las fichas del jugador 2 es menor a
                    la suma de las fichas del jugador 1, gana el jugador 2
        else:
            return None
```

4.2.3 El método string_to_tuple

convierte el input del jugador 1, que es una cadena a una tupla de python.

```
def string_to_tuple(self, string): #convierte un string a una tupla
    val=-1
    try:
        string.replace("(", "")
        string.replace(")", "")
        val=tuple(map(int, string.split(",")))
    except:
        pass
    return val
```

4.2.4 El método Jugar

El método jugar es el encargado de iniciar el juego y es el que maneja el flujo del juego. En primer lugar, crea una instancia de la clase maquina, que es una inteligencia artificial encargada de jugar por la computadora. Luego, decide quién es el jugador que empieza y realiza la jugada inicial colocando la ficha central en el tablero. Después, entra en un ciclo *while* que se ejecuta mientras no haya un ganador y sigue el siguiente flujo:

1. Imprime el tablero y las fichas en la mesa.
2. Si es el turno del jugador humano, le pide que ingrese la ficha que desea jugar y verifica si es una ficha válida para jugar. Si es válida, la coloca en el tablero y la elimina de la lista de fichas del jugador. Si no es válida, no juega.
3. Si es el turno de la computadora, llama al método jugar de la instancia de la clase maquina para que seleccione la ficha que jugará. Si la ficha es válida, la coloca en el tablero y la elimina de la lista de fichas de la computadora. Si no es válida, no juega.
4. Verifica si hay un ganador. Si lo hay, termina el juego y anuncia al ganador.

```
def jugar(self): #funcion que inicia el juego
    maquina1 = maquina(self.fichas_jugador_2, self.tablero) #se crea la
        maquina
    turno = self.decidir_turno() #se decide el turno
    if turno == 1: #si el turno es 1, empieza el jugador 1
        print("Empieza el jugador 1")
        self.tablero.append(self.ficha_central), self.fichas_jugador_1.
            remove(self.ficha_central) #se agrega la ficha central al
            tablero y se elimina de las fichas del jugador 1
        turno = 2 #turno de la maquina
    elif turno == 2: #si el turno es 2, empieza el jugador 2
        print("Empieza la computadora")
        self.tablero.append(self.ficha_central), self.fichas_jugador_2.
            remove(self.ficha_central) #se agrega la ficha central al
            tablero y se elimina de las fichas del jugador 2
        turno = 1 #turno del jugador 1
    self.skip=0 #contador de turnos sin jugar
    while (ganador := self.determinar_ganador(self.tablero, self.
        fichas_jugador_1, self.fichas_jugador_2,self.skip)) is None: #
        mientras no haya un ganador
```

```
print("---------------------------------------Tablero
    ----------------------------------------------")
print(f"\n\nFichas en la mesa: {self.tablero}\n\n")
print("---------------------------------------Tablero
    ----------------------------------------------\n")
time.sleep(2)
if turno == 1: #turno del jugador 1
    print("Turno del jugador 1")
    print(f"Fichas del jugador 1: {self.fichas_jugador_1}")
    ficha = input("Ingrese la ficha que desea jugar: ")
    ficha = self.string_to_tuple(ficha)
    if ficha==-1:
        self.skip+=1
    if ficha in self.fichas_jugador_1: #si la ficha esta en las
       fichas del jugador 1
        if ficha[0]==self.tablero[0][0] :
            self.tablero.insert(0, ficha[::-1]) #se invierte la
               ficha y se agrega al tablero
            self.fichas_jugador_1.remove(ficha)
            self.skip=0
        elif ficha[1]==self.tablero[0][0]: #si el primer numero
           de la ficha es igual al primer numero de la ficha del
            tablero
            self.tablero.insert(0, ficha)
            self.fichas_jugador_1.remove(ficha)
            self.skip=0
        elif ficha[1]==self.tablero[-1][1]: #si el segundo
           numero de la ficha es igual al segundo numero de la
           ficha del tablero
            self.tablero.append(ficha[::-1]) #se invierte la
               ficha y se agrega al tablero
            self.fichas_jugador_1.remove(ficha)
            self.skip=0
        elif ficha[0]==self.tablero[-1][1]: #si el primer numero
            de la ficha es igual al segundo numero de la ficha
           del tablero
            self.tablero.append(ficha)
            self.fichas_jugador_1.remove(ficha)
            self.skip=0
        else:
            print("El jugador decidio no jugar")
    else:
        print("No tiene esa ficha")
    turno = 2
elif turno == 2:
    print("\nTurno de la computadora")
    print(f"Fichas de la maquina: {self.fichas_jugador_2}")
    ficha = maquina1.jugar(self.fichas_jugador_2, self.tablero)
       #se obtiene la ficha que jugara la maquina
    print(f"La computadora jugo la ficha {ficha}\n")
    time.sleep(2)
    if ficha==-1: #si la ficha es -1, la maquina no puede jugar
        self.skip+=1 #se aumenta el contador de turnos sin jugar
    if ficha in self.fichas_jugador_2:
        if ficha[0]==self.tablero[0][0] :
            self.tablero.insert(0, ficha[::-1])
            self.fichas_jugador_2.remove(ficha)
```

```
                    self.skip=0
                elif ficha[1]==self.tablero[0][0]:
                    self.tablero.insert(0, ficha)
                    self.fichas_jugador_2.remove(ficha)
                    self.skip=0
                elif ficha[1]==self.tablero[-1][1]:
                    self.tablero.append(ficha[::-1])
                    self.fichas_jugador_2.remove(ficha)
                    self.skip=0
                elif ficha[0]==self.tablero[-1][1]:
                    self.tablero.append(ficha)
                    self.fichas_jugador_2.remove(ficha)
                    self.skip=0
                else:
                    print("La computadora decide pasar")
            else:
                print("No tiene esa ficha")
            turno = 1

    print(f"El ganador es el jugador {ganador}")
```

4.3 La Clase jugador_Maquina

Esta clase representa un jugador de máquina que puede jugar al dominó. Tiene atributos para guardar su tablero y sus fichas, y métodos para actualizarlos y elegir la mejor ficha para jugar. El método jugar devuelve la mejor ficha posible o -1 si no hay ninguna ficha válida. La mejor ficha es la que tiene la mayor suma de sus valores.

```
class jugador_Maquina:
tablero = []
fichas = []
#Algortimo utilizado: Best First Search
def __init__(self, fichas, tablero): #constructor de la clase
   jugador_Maquina
    self.fichas = fichas #inicializa las fichas del jugador
    self.tablero = tablero #inicializa el tablero del jugador
```

4.3.1 Los Métodos para Actualizar el Tablero y Fichas

```
def actualizar_tablero(self, tablero): #actualiza el tablero del jugador
    self.tablero = tablero

def actualizar_fichas(self, fichas): #actualiza las fichas del jugador
    self.fichas = fichas #debido a que es el juego quien elimina las
       fichas, se actualizan las fichas del jugador
```

4.3.2 El Método mejor_ficha

Este método define la mejor ficha para jugar en un juego de dominó, basándose en la heurística de que la ficha de mayor valor es la que se debe jugar. El método recibe como parámetro una lista de fichas y retorna la ficha que tiene la mayor suma de sus dos valores. El método empieza asignando la primera ficha de la lista como la mejor ficha, y luego recorre el resto de la lista comparando cada ficha con la mejor ficha actual. Si encuentra una ficha que tiene una suma

mayor que la mejor ficha, actualiza la mejor ficha con esa ficha. Al final del recorrido, el método devuelve la mejor ficha encontrada.

```
def mejor_ficha(self, fichas): #retorna la mejor ficha para jugar, Se
    considera que esta es la funcion que implementa la heuristica, de que
     la ficha de mayor valor es la que se debe jugar
     mejor_ficha = fichas[0] #se inicializa la mejor ficha como la
         primera ficha de la lista de fichas
     for ficha in fichas: #se recorre la lista de fichas
         if ficha[0]+ficha[1] > mejor_ficha[0]+mejor_ficha[1]: #si la
             suma de los valores de la ficha actual es mayor a la suma de
             los valores de la mejor ficha, se actualiza la mejor ficha
              mejor_ficha = ficha #se actualiza la mejor ficha
     return mejor_ficha #se retorna la mejor ficha
```

4.3.3 El Método fichas_posibles

Este método devuelve una lista de las fichas que se pueden colocar en el tablero según los valores de los extremos. Es la función sucesora que se usa para generar los posibles movimientos en el juego de dominó. Para cada ficha en la mano del jugador, se comprueba si alguno de sus valores coincide con el valor derecho o izquierdo del tablero. Si es así, se añade la ficha a la lista de fichas posibles. Finalmente, se devuelve la lista de fichas posibles.

```
def fichas_posibles(self, Valorderecho, Valorizquierdo): #retorna las
    fichas que se pueden jugar en el tablero, esta es considerada la
    funcion sucessora
     fichas_posibles = [] #se inicializa la lista de fichas posibles
     for ficha in self.fichas: #se recorre la lista de fichas
         if ficha[0] == Valorderecho or ficha[1] == Valorderecho: #si el
             valor derecho de la ficha es igual al valor derecho del
             tablero o el valor izquierdo de la ficha es igual al valor
             derecho del tablero, se agrega la ficha a la lista de fichas
             posibles
              fichas_posibles.append(ficha) #se agrega la ficha a la lista
                  de fichas posibles
         elif ficha[0] == Valorizquierdo or ficha[1] == Valorizquierdo: #
             si el valor izquierdo de la ficha es igual al valor izquierdo
              del tablero o el valor derecho de la ficha es igual al valor
              izquierdo del tablero, se agrega la ficha a la lista de
             fichas posibles
              fichas_posibles.append(ficha) #se agrega la ficha a la lista
                  de fichas posibles
     return fichas_posibles #se retorna la lista de fichas posibles
```

4.3.4 El Método jugar

Este método define la lógica del jugador para elegir la mejor ficha para jugar en cada turno. Recibe como parámetros las fichas del jugador y el tablero actual. Primero, actualiza las fichas y el tablero del jugador con los métodos correspondientes. Luego, verifica si el jugador tiene fichas disponibles. Si no tiene, retorna -1 para indicar que el juego termina. Si tiene, obtiene los valores izquierdo y derecho del tablero y busca las fichas posibles que coincidan con alguno de esos valores. Si no hay fichas posibles, retorna -1 para indicar que el jugador pasa el turno. Si hay fichas posibles, llama al método mejor_ficha para seleccionar la ficha que maximice el beneficio del jugador y la retorna.

```
def jugar(self,fichas,tablero): #retorna la mejor ficha para jugar, esta
    es considerada la funcion objetivo
    self.actualizar_fichas(fichas) #se actualizan las fichas del jugador
    self.actualizar_tablero(tablero) #se actualiza el tablero del
       jugador
    if len(self.fichas)==0: #si el jugador no tiene fichas, retorna -1,
       hace una especie de funcion objetivo, que si no tiene fichas,
       retorna -1 y el juego termina
        return -1

    Valorderecho = self.tablero[len(self.tablero)-1][1] #se obtiene el
       valor derecho del tablero
    Valorizquierdo = self.tablero[0][0] #se obtiene el valor izquierdo
       del tablero
    fichas_posibles = self.fichas_posibles(Valorderecho, Valorizquierdo)
        #se obtienen las fichas posibles

    if len(fichas_posibles) == 0: #si no hay fichas posibles, retorna
       -1, y se toma como un pasar turno, si ambos jugadores pasan turno
       , el juego termina
        return -1
    else:
        return self.mejor_ficha(fichas_posibles) #si hay fichas posibles
           , retorna la mejor ficha para jugar
```

5 Conclusión

La implementación de un programa capaz de jugar al juego de dominó de forma inteligente y estratégica requiere de la aplicación de técnicas de inteligencia artificial, como heurísticas y algoritmos de búsqueda. Además, se debe establecer una metodología clara que permita definir los estados iniciales y finales del juego, así como las funciones objetivo y sucesor que simulan el desarrollo del juego y la toma de decisiones por parte de la computadora. De esta forma, se podría diseñar un programa que simule el juego de dominó de forma efectiva y brinde una experiencia de juego desafiante y entretenida para jugadores principiantes y experimentados. El programa crea objetos de las clases Domino y Juego Domino, y utiliza funciones para decidir el turno, determinar el ganador y jugar el juego. También se describe cómo se crea un objeto de la clase jugador Maquina para el jugador de la computadora, que utiliza una función heurística llamada mejor ficha para determinar la mejor jugada posible. El programa continúa ejecutándose en un ciclo while hasta que se alcanza la función objetivo de determinar al ganador del juego.

References

[1] colaboradores de Wikipedia. Dominó. *Wikipedia, la enciclopedia libre*, 4 2023.