Proyecto Final. Primera entrega. Dominó

Integrantes
Acosta Porcayo Alan Omar 320206102
Gutiérrez Grimaldo Alejandro 320282098
Medina Villa Samuel 320249538

Índice

1. Antecedentes históricos

El origen de este juego no es tan claro, ya que algunos historiadores y antropólogos fueron los griegos, también otros afirman que fueron los hebreos. Sin embargo, el juego actual del dominó parece que se inició en China, donde se jugaba hace 1.500 años de manera semejante a como se hace hoy. Se podría considerar como el primer dominó de la historia.

La forma actual conocida en Europa y el mundo no aparece hasta el siglo XVIII, cuando lo introdujeron los italianos en este continente. Gracias a la enorme expansión de la cultura europea a través del mundo, llegó a diversos países y culturas.

Inicialmente, las fichas se fabricaban mediante la unión de dos láminas de ébano en ambos lados de la ficha de hueso. Este método tenía la ventaja de prevenir posibles trampas al ocultar los puntos en la parte posterior de la ficha bajo ciertas condiciones de iluminación, además de crear un atractivo contraste entre los puntos blancos y el fondo negro, lo que permitía que el hueso fuera visible a través de los agujeros en el ébano.

2. Descripción

El objetivo principal del juego de dominó es ser el primer jugador en quedarse sin fichas en su mano. Para lograrlo, los jugadores deben emparejar las fichas que poseen con las que ya están en la mesa, asegurándose de que los números de los puntos coincidan. Esto implica tomar decisiones estratégicas sobre cuándo y cómo jugar sus fichas para maximizar sus posibilidades de quedarse sin fichas antes que sus oponentes.

3. Definición de clases

3.1. Clase *Ficha*

Esta clase representa una ficha de dominó simple.

3.1.1. Atributos

1. *caraIzq.* Variable privada de tipo *int* con el valor de la parte izquierda de la ficha.
2. *caraDer.* Variable privada de tipo *int* con el valor de la parte derecha de la ficha.
3. *suma.* Variable privada de tipo *int* con la suma de ambas partes de la ficha, se calcula al momento de crear la ficha.
4. *mula.* Variable privada de tipo *boolean* que indica si la ficha es una mula o no.

3.1.2. Métodos

1. *Ficha(int caraIzq, int caraDer).* Método constructor que recibe dos enteros que representan los valores de las caras de la ficha. Calcula la suma y determina si la ficha es una mula.
2. *getCaraIzq().* Método público que retorna el valor de la cara izquierda de la ficha.

3. *getCaraDer().* Método público que retorna el valor de la cara derecha de la ficha.
4. *getSuma().* Método público que retorna el valor de la suma de la ficha.
5. *esMula().* Método público que retorna un booleano que indica si la ficha es una mula o no.
6. *girar().* Método público que intercambia los valores de las caras de la ficha.
7. *toString().* Método público (sobreescritura) que retorna una cadena con formato del valor de las caras de la ficha.

3.2. Clase Abstracta *Jugador*

Esta clase representa a un jugador de dominó, ya sea humano o bot.

3.2.1. Atributos

1. *nombre.* Variable protegida de tipo *String* con el nombre del jugador.
2. *fichas.* Variable protegida de tipo *ArrayList<Ficha>* con las fichas del jugador.
3. *puedeJugar.* Variable protegida de tipo *boolean* que indica si el jugador tiene una ficha que pueda jugar.

3.2.2. Métodos

1. *getNombre().* Método público que retorna el nombre del jugador.
2. *getFichas().* Método público que retorna las fichas del jugador.
3. *getFicha(int i).* Método público que retorna la ficha en la posición i del jugador.
4. *getCantidadFichas().* Método público que retorna la cantidad de fichas del jugador.
5. *agregarFicha(Ficha ficha).* Método público que agrega una ficha a las fichas del jugador.
6. *puedeJugar().* Método público que retorna un booleano que indica si el jugador puede jugar.
7. *puedeJugar(ArrayList<Ficha> mesaActual).* Método público (sobrecarga) que comprueba si dentro de las fichas del jugador hay alguna que pueda jugarse en la mesa actual y actualiza el valor de *puedeJugar*.
8. *robar(ArrayList<Ficha> pozo).* Método público que sustrae una ficha del pozo y la agrega a las fichas del jugador.
9. *primerTurno(ArrayList<Ficha> mesaActual).* Método público abstracto que realiza el primer turno del jugador.
10. *turno(ArrayList<Ficha> mesaActual).* Método público abstracto que realiza los siguientes turnos del jugador.

3.3. Clase *Humano*

Esta clase representa a un jugador controlado por el usuario.

3.3.1. Métodos

1. *Humano(String nombre).* Método constructor que recibe el nombre del jugador.
2. *primerTurno(ArrayList<Ficha> mesaActual).* Método público (sobreescritura) que realiza el primer turno de manera automática jugando la mula mas alta o si no tiene mula, la ficha con la suma mas alta.
3. *turno(ArrayList<Ficha> mesaActual).* Método público (sobreescritura) que realiza los siguientes turnos por medio de la entrada del usuario. Primero le pide al usuario que elija una ficha dentro del rango de sus fichas, luego le pide que elija un lado de la mesa en el que la quiere colocar y si se introdujeron valores correctos realiza el movimiento, si no, vuelve a pedir los valores.

3.4. Clase *Bot*

Esta clase representa a un jugador controlado por la computadora.

3.4.1. Métodos

1. *Bot(String nombre).* Método constructor que recibe el nombre del jugador.
2. *primerTurno(ArrayList<Ficha> mesaActual).* Método público (sobreescritura) que realiza el primer turno de manera automática jugando la mula mas alta o si no tiene mula, la ficha con la suma mas alta.
3. *turno(ArrayList<Ficha> mesaActual).* Método público (sobreescritura) que realiza los siguientes turnos de manera automática buscando una ficha que pueda ser colocada empezando en la parte izquierda de la mesa y si no encuentra, busca en la parte derecha.

3.5. Clase *Mesa*

Esta clase representa la mesa de juego.

3.5.1. Atributos

1. *mesa.* Variable privada y estática de tipo *ArrayList<Ficha>* con las fichas de la mesa.
2. *pozo.* Variable privada y estática de tipo *ArrayList<Ficha>* con las fichas sobrantes del juego.
3. *jugadores.* Variable privada y estática de tipo *List<Jugador>* con los jugadores del juego. Inicialmente contiene un objeto de tipo *Humano* y dos de tipo *Bot*.

3.5.2. Métodos

1. *Mesa(int tipoJuego).* Método constructor que crea las fichas de la mesa y las agrega a *mesa*. Determina el tipo de juego entre jugador vs bot o bot vs bot. Mezcla las fichas de la mesa, las reparte entre los jugadores y decide quien empieza.
2. *getMesa().* Método público que retorna las fichas de la mesa.
3. *getPozo().* Método público que retorna las fichas del pozo.

4. *getJugadores().* Método público que retorna los jugadores del juego.
5. *mezclar().* Método privado que por medio de un número aleatorio intercambia las fichas de la mesa un total de 28 veces.
6. *repartir().* Método privado que reparte 7 fichas de la mesa a cada uno de los dos jugadores y las restantes las agrega al pozo.
7. *decidirPrimerTurno().* Método privado que determina quien empieza el juego buscando quien tiene la mula mas alta y si no hay mula, quien tiene la ficha con la suma mas alta. En caso de que el primer turno sea del jugador en la posición 1 de la lista de jugadores se cambia el orden de los jugadores.
8. *cambioDeTurno().* Método público que cambia el orden de los jugadores en la lista.
9. *imprimir().* Método público que imprime el estado actual de la mesa.
10. *imprimir(ArrayList<Ficha> fichas).* Método público (sobrecarga) que imprime las fichas que recibe como parámetro con el número de ficha arriba de cada una.

3.6. Clase *Domino*

Esta clase contiene el método *main* y es la encargada de ejecutar la lógica principal del juego, la cual será explicada en la sección de algoritmo.

4. Definición de objetos

- *ArrayList<Ficha>.* Colección de objetos de tipo *Ficha*. Se crea en las clases *Mesa*, *Humano* y *Bot* para almacenar las fichas de la mesa, las fichas del jugador y las fichas que puede jugar el bot respectivamente.
- *List<Jugador>.* Colección de objetos de tipo *Jugador*. En ella se almacenan objetos de tipo *Humano* y *Bot* que representan a los jugadores del juego. Se crea en la clase *Mesa*.
- *Mesa.* Objeto de tipo *Mesa* que contiene la configuración inicial del juego. Se crea en la clase *Domino*.

5. Algoritmo

Se comienza creando un objeto *Scanner* llamado *sc* para leer las entradas del usuario y un objeto *Clip* llamado *musica* para reproducir los clips de audio. Para reproducir un clip se hace dentro de un bloque *try-catch* para evitar que el programa se detenga en caso de que no se encuentre el archivo de audio.

```
Scanner sc = new Scanner(System.in);
Clip musica = null;

try {
  musica = AudioSystem.getClip();
  musica.open(AudioSystem.getAudioInputStream(Domino.class.getResource("/recursos/MoonlightSonata.wav")));
  musica.loop(Clip.LOOP_CONTINUOUSLY);
} catch (Exception e) {
  e.printStackTrace();
}
```

Luego, se imprime el menú con las opciones de juego para que el usuario pueda seleccionar jugador vs bot o bot vs bot. Se lee la entrada del usuario y se crea un objeto de tipo *Mesa* llamado *mesa* con el tipo de juego seleccionado por el usuario.

```
System.out.println("%%%%%%%%%%%%%%% Bienvenido a Domino %%%%%%%%%%%%%%%");
System.out.println("Escoga el modo de juego:");
System.out.println("1. Jugador vs Bot");
System.out.println("2. Bot vs Bot");
System.out.print("$ ");
int tipoJuego = sc.nextInt();
Mesa mesa = new Mesa(tipoJuego);
```

Dentro de un ciclo *while* con la condición de repetición de que los jugadores tengan fichas y el pozo no este vacío, se limpia la terminal usando una un bloque *try-catch* y un objeto *ProcessBuilder*.

```
try {
  new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
} catch (Exception e) {
  e.printStackTrace();
}
```

Posteriormente, se imprime las fichas de la mesa, el número de fichas en el pozo y se anuncia el turno del jugador en la posición 0 de la lista de jugadores e imprime sus fichas.

```
System.out.println("\n\nMesa actual:");
mesa.imprimir();
System.out.println("\nNumero de fichas en el Pozo: " + mesa.getPozo().size());

System.out.println("\n%%%%% Turno de " + mesa.getJugadores().get(0).getNombre() + " %%%%%");
mesa.imprimir(mesa.getJugadores().get(0).getFichas());
```

El siguiente bloque de código corresponde al turno de los jugadores. Si la mesa esta vacia quiere decir que es el primer turno, en ese caso se llama al método *primerTurno* y se continua con el flujo del programa. Para los turnos siguientes primero se comprueba si el jugador puede jugar una ficha realiza su turno con el método *turno*. En caso de que no pueda jugar y no haya fichas en el pozo se imprime un mensaje diciendo que el jugador con el turno actual pasa de turno, y si hay fichas en el pozo roba una, se vuelve a imprimir sus fichas y se pasa de turno.

```
if (mesa.getMesa().isEmpty())
  mesa.getJugadores().get(0).primerTurno(mesa.getMesa());
else {
  if (mesa.getJugadores().get(0).puedeJugar(mesa.getMesa()))
    mesa.getJugadores().get(0).turno(mesa.getMesa());
  else {
    if (mesa.getPozo().isEmpty()) {
      System.out.println("No puedes jugar y no hay fichas en el pozo, pasas tu turno");

      try {
        Thread.sleep(3000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    } else {
      if (mesa.getJugadores().get(0) instanceof Humano)
        System.out.println("Necesitas robar una ficha");

      try {
        Thread.sleep(3000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
```

```
      System.out.println(mesa.getJugadores().get(0).getNombre() + " roba 1 ficha y pasa su turno");
      mesa.getJugadores().get(0).robar(mesa.getPozo());
      mesa.imprimir(mesa.getJugadores().get(0).getFichas());

      try {
        Thread.sleep(3000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Al salir del ciclo *while*, dentro de un bloque *try-catch* se para la reproducción del clip de audio y se inicia otro clip de audio con la música de victoria.

```
try {
  if (musica != null)
    musica.stop();

  musica = AudioSystem.getClip();
  musica.open(AudioSystem.getAudioInputStream(Domino.class.getResource("/recursos/Victoria.wav")));
  musica.start();
} catch (Exception e) {
  e.printStackTrace();
}
```

Para terminar se decide el ganador. Si un jugador se queda sin fichas se imprime un mensaje diciendo que gano pero si el pozo se vacía se busca quien tiene la suma más alta y se imprime un mensaje diciendo que gano o si hay empate.

```
if (!mesa.getJugadores().get(0).puedeJugar() &&
      !mesa.getJugadores().get(1).puedeJugar() &&
      mesa.getPozo().isEmpty() ) {
  int suma1 = 0, suma2 = 0;
  System.out.println("Se acabaron las fichas");

  for (int i = 0; i < mesa.getJugadores().get(0).getCantidadFichas(); i++)
    suma1 += mesa.getJugadores().get(0).getFicha(i).getSuma();
  for (int i = 0; i < mesa.getJugadores().get(1).getCantidadFichas(); i++)
    suma2 += mesa.getJugadores().get(1).getFicha(i).getSuma();

  if (suma1 < suma2)
    System.out.println("El ganador es " + mesa.getJugadores().get(0).getNombre());
  else if (suma1 > suma2)
    System.out.println("El ganador es " + mesa.getJugadores().get(1).getNombre());
  else
    System.out.println("Empate");
} else
  System.out.println("\n%%%%%%%%%% El ganador es " + mesa.getJugadores().get(0).getNombre() + " %%%%%%%%%%");
```

6. Herencia de clases

La única herencia que realiza es de la clase abstracta *Jugador* a las clases *Humano* y *Bot*. La definición de *Jugador* contiene dos métodos abstractos que son *primerTurno* y *turno* que son implementados en las clases *Humano* y *Bot* respectivamente.

7. Conclusiones

Este proyecto de implementación de un juego de dominó en Java ha abordado varios aspectos esenciales de la programación orientada a objetos y la interacción con el usuario. El diseño se basa en la encapsulación de fichas, jugadores y la mesa en clases independientes, lo que promueve la organización eficiente y el mantenimiento del código.

La simulación del juego sigue las reglas auténticas del dominó, con atención al detalle para determinar la jugabilidad de las fichas en función de las reglas del juego. Se gestionan situaciones como el robo de fichas y el cambio de turno de manera efectiva, demostrando una comprensión profunda de las dinámicas del dominó. Esto se logra mediante el uso de la herencia y la implementación de métodos abstractos en las clases de jugadores. La implementación también aborda la gestión de excepciones, garantizando que el juego sea robusto y confiable en diversas circunstancias.

Al realizar este tipo de trabajo promueve a utilizar nuestras habilidades que hemos estado adquiriendo durante el curso y también promueve a la investigación y al trabajo colectivo.

8. Referencias

De Tenis De Huelva, R. C. R. (s. f.). *Dominó* — https://rcrtenishuelva1889.com/page/9/domino#:~:text=El%20domin%C3%B3%20surgi%C3%B3%20hace%20mil,los%20italianos%20por%20todas%20partes

Familia Vintage. (2018, June 5). *Como Jugar Al Domino, Reglas del Dominó* [Video]. YouTube. https://www.youtube.com/watch?v=-hbARCT1qow

Historia, C., & Historia, C. (2023, 22 abril). *La historia del dominó, es una de las más interesantes que existe Read more.* CurioSfera Historia. https://curiosfera-historia.com/historia-del-domino-origen-inventor/

Java Platform SE 8. (2023). Oracle.com. https://docs.oracle.com/javase/8/docs/api/

JUEGOS CON HISTORIA: *El Dominó* — Lekotek. (s. f.). https://www.lekotek.org.ar/juegos-con-historia-el-domino/

Solano, J. (2017, 20 enero). *Manual de prácticas de Programación Orientada a Objetos.* Laboratorio de Computación Salas A y B. http://lcp02.fi-b.unam.mx/

9. Apéndice

9.1. Diagramas UML de clases

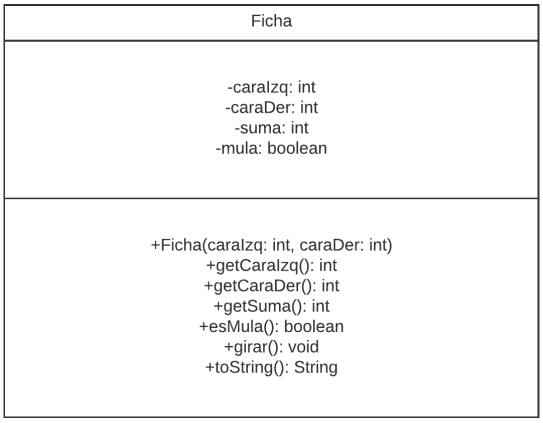

Figura 1: UML de clase *Ficha*

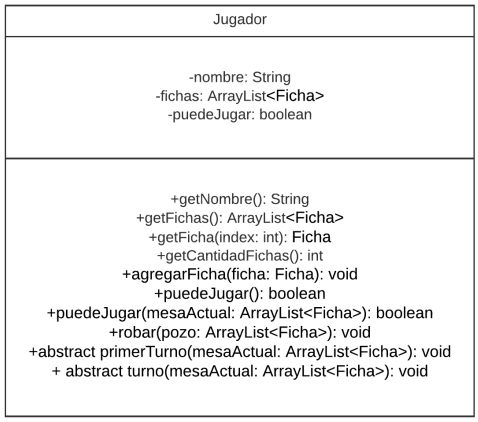

Figura 2: UML de clase *Jugador*

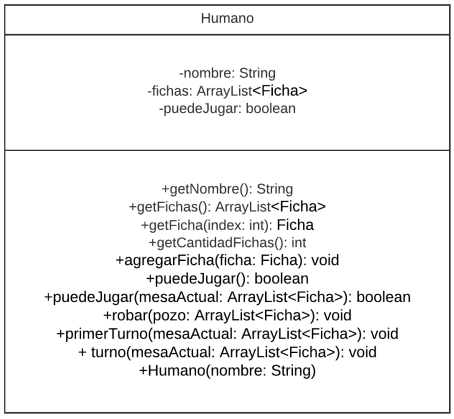

Figura 3: UML de clase *Humano*

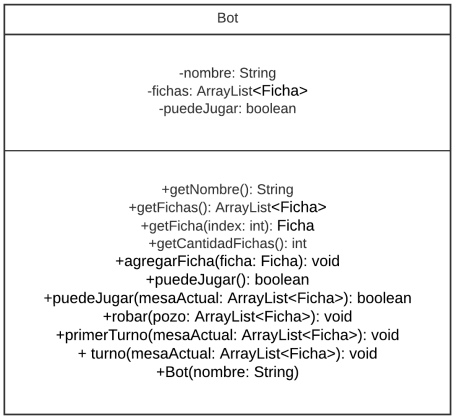

Figura 4: UML de clase *Bot*

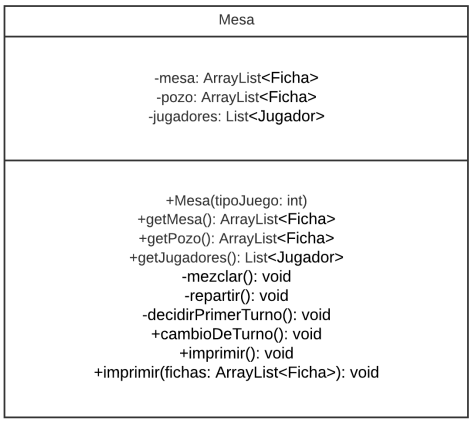

Figura 5: UML de clase *Mesa*

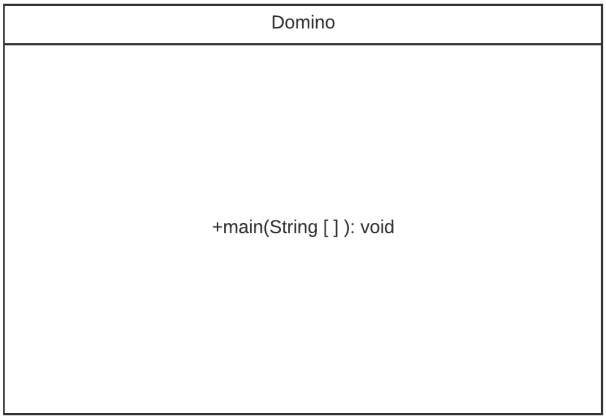

Figura 6: UML de clase *Domino*

9.2. Diagramas UML de objetos

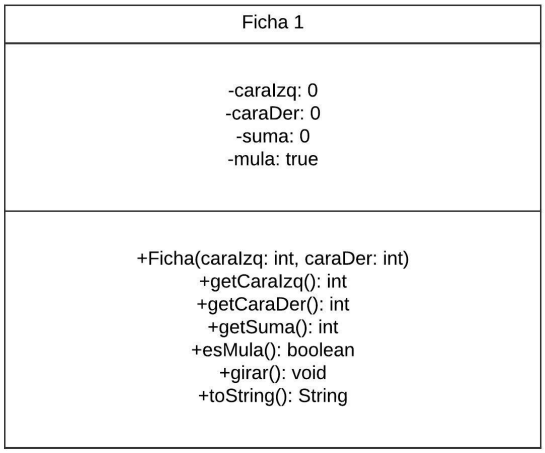

Figura 7: UML de objeto *Ficha1*

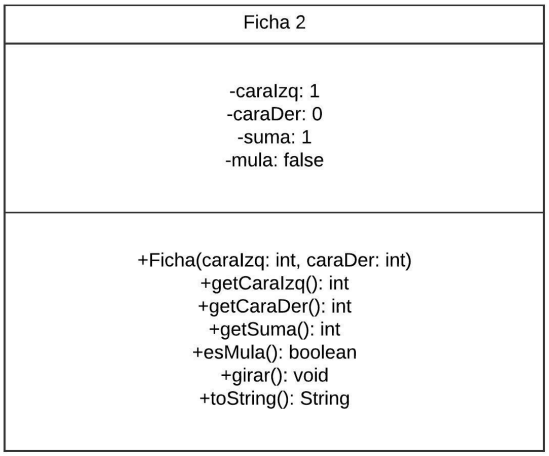

Figura 8: UML de objeto *Ficha2*

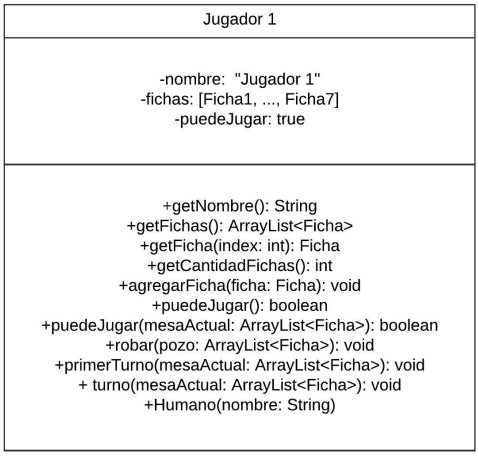

Figura 9: UML de objeto *Jugador1*

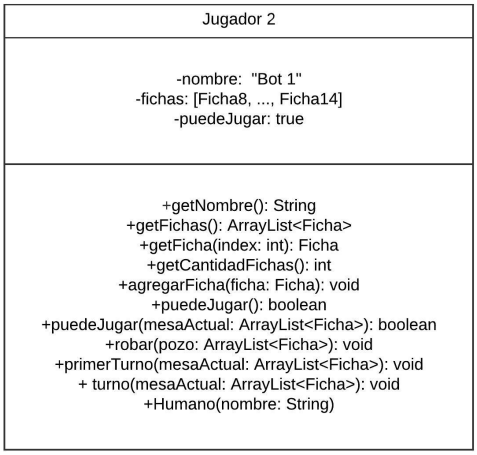

Figura 10: UML de objeto *Jugador1*

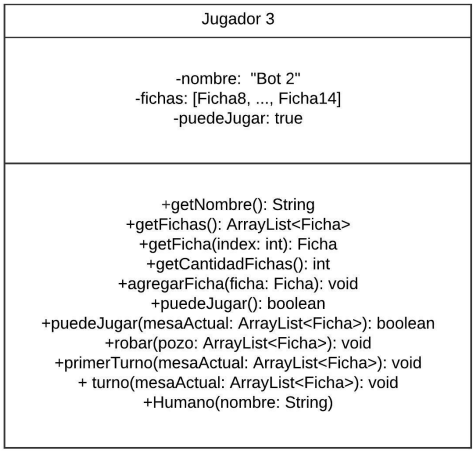

Figura 11: UML de objeto *Jugador1*

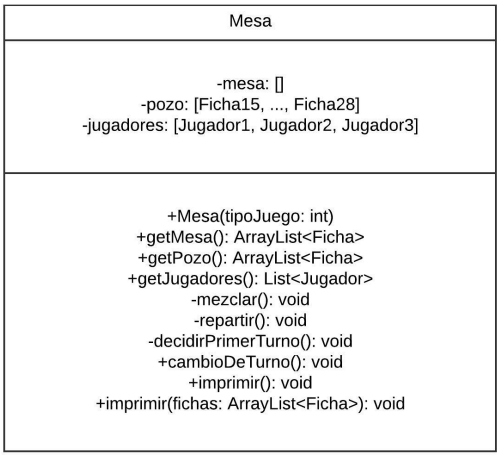

Figura 12: UML de objeto *Mesa*

9.3. Diagrama UML de casos de uso

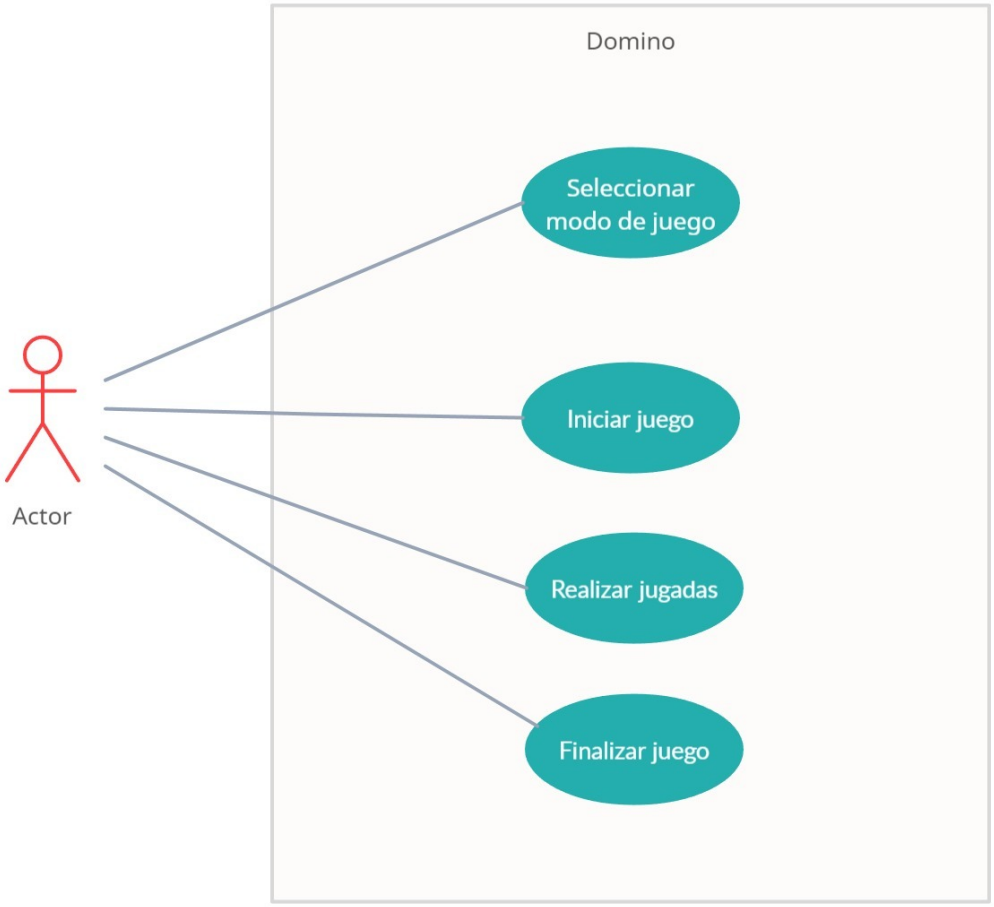

Figura 13: UML de casos de uso

9.4. Código fuente

9.4.1. Ficha.java

```
package domino;

/**
  * Esta clase representa una ficha de domino con dos caras
  */

public class Ficha {
  private int caraIzq;
  private int caraDer;
  private int suma;
  private boolean mula = false;

  /**
    * Metodo constructor que recibe las dos caras de la ficha
    * @param caraIzq El numero de la cara izquierda
    * @param caraDer El numero de la cara derecha
    */
  public Ficha(int caraIzq, int caraDer) {
    this.caraIzq = caraIzq;
```

```
    this.caraDer = caraDer;
    this.suma = caraIzq + caraDer;
    if (caraIzq == caraDer)
      mula = true;
  }

  /**
    * Metodo getter para la cara izquierda
    * @return Numero de la cara izquierda
    */
  public int getCaraIzq() {
    return caraIzq;
  }

  /**
    * Metodo getter para la cara derecha
    * @return Numero de la cara derecha
    */
  public int getCaraDer() {
    return caraDer;
  }

  /**
    * Metodo getter para la suma de las caras
    * @return Suma de las caras
    */
  public int getSuma() {
    return suma;
  }

  /**
    * Metodo getter para saber si la ficha es mula
    * @return true si la ficha es mula, false si no
    */
  public boolean esMula() {
    return mula;
  }

  /**
    * Cambia la cara izquierda por la derecha y viceversa
    */
  public void girar() {
    int aux = caraIzq;
    caraIzq = caraDer;
    caraDer = aux;
  }

  /**
    * Imprime la ficha con el formato [caraIzq|caraDer]
    */
  public String toString() {
    return "[" + caraIzq + "|" + caraDer + "]";
  }
}
```

9.4.2. Jugador.java

```
package domino;

import java.util.ArrayList;

/**
  * Esta clase abstracta representa un jugador de domino
  */

public abstract class Jugador {
  protected String nombre;
```

```
protected ArrayList<Ficha> fichas = new ArrayList<Ficha>();
protected boolean puedeJugar = true;

/**
 * Metodo getter para el nombre del jugador
 * @return Nombre del jugador
 */
public String getNombre() {
  return nombre;
}

/**
 * Metodo getter para las fichas del jugador
 * @return Fichas del jugador
 */
public ArrayList<Ficha> getFichas() {
  return fichas;
}

/**
 * Metodo getter para una ficha especifica del jugador
 * @param i Indice de la ficha
 * @return Ficha del jugador
 */
public Ficha getFicha(int i) {
  return fichas.get(i);
}

/**
 * Metodo getter para la cantidad de fichas del jugador
 * @return Cantidad de fichas del jugador
 */
public int getCantidadFichas() {
  return fichas.size();
}

/**
 * Agrega una ficha a las fichas del jugador
 * @param ficha Ficha a agregar
 */
public void agregarFicha(Ficha ficha) {
  fichas.add(ficha);
}

/**
 * Metodo getter para saber si el jugador puede jugar
 * @return true si el jugador puede jugar, false si no
 */
public boolean puedeJugar() {
  return puedeJugar;
}

/**
 * Metodo setter para saber si el jugador puede jugar
 * @param mesaActual La fichas de la mesa actual
 * @return puedeJugar true si el jugador puede jugar, false si no
 */
public boolean puedeJugar(ArrayList<Ficha> mesaActual) {
  for (Ficha ficha : fichas) {
    if (ficha.getCaraIzq() == mesaActual.get(0).getCaraIzq() ||
           ficha.getCaraDer() == mesaActual.get(0).getCaraIzq() ||
           ficha.getCaraIzq() == mesaActual.get(mesaActual.size() - 1).getCaraDer() ||
           ficha.getCaraDer() == mesaActual.get(mesaActual.size() - 1).getCaraDer()) {
      puedeJugar = true;
      return puedeJugar;
    }
  }
  puedeJugar = false;
  return puedeJugar;
```

```
  }

  /**
    * Metodo para robar una ficha del pozo
    * @param pozo El pozo de fichas
    */
  public void robar(ArrayList<Ficha> pozo) {
    fichas.add(pozo.get(0));
    pozo.remove(0);
  }

  /**
    * Metodo abstracto para el primer turno del jugador
    * @param mesaActual La fichas de la mesa actual
    */
  public abstract void primerTurno(ArrayList<Ficha> mesaActual);

  /**
    * Metodo abstracto para el turno del jugador
    * @param mesaActual La fichas de la mesa actual
    */
  public abstract void turno(ArrayList<Ficha> mesaActual);
}
```

9.4.3. Humano.java

```
package domino;

import java.util.ArrayList;
import java.util.Scanner;

/**
 * Esta clase representa un jugador humano de domino
 */

public class Humano extends Jugador {
  /**
    * Metodo constructor
    * @param nombre Nombre del jugador
    */
  public Humano(String nombre) {
    this.nombre = nombre;
  }

  /**
    * Metodo para el primer turno del jugador
    * @param mesaActual Mesa actual del juego
    */
  @Override
  public void primerTurno(ArrayList<Ficha> mesaActual) {
    Scanner sc = new Scanner(System.in);
    int ficha = -1;
    int max = -1;

    for (Ficha f : fichas) {
      if (f.esMula() && f.getSuma() > max) {
        max = f.getSuma();
        ficha = fichas.indexOf(f);
      }
    }

    if (ficha != -1)
      System.out.println(nombre + " juegas la mula mas alta: " + fichas.get(ficha));
    else {
      for (int i = 0; i < fichas.size(); i++) {
        if (fichas.get(i).getSuma() > max) {
          max = fichas.get(i).getSuma();
```

```
          ficha = i;
        }
      }
      System.out.println(nombre + " juegas la ficha mas alta: " + fichas.get(ficha));
    }

    mesaActual.add(fichas.get(ficha));
    fichas.remove(ficha);

    try {
      Thread.sleep(3000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  /**
   * Metodo para el turno del jugador
   * @param mesaActual Mesa actual del juego
   */
  @Override
  public void turno(ArrayList<Ficha> mesaActual) {
    Scanner sc = new Scanner(System.in);
    int ficha = 0;
    char lado = ' ';

    do {
      do {
        System.out.println("Que ficha quieres jugar? (1 - " + fichas.size() + ")");
        ficha = sc.nextInt() - 1;
        if (ficha < 0 || ficha >= fichas.size())
          System.out.println("Ficha invalida");
      } while (ficha < 0 || ficha >= fichas.size());

      System.out.println("Donde la quieres jugar? (izquierda o derecha)");
      lado = sc.next().charAt(0);
      lado = Character.toLowerCase(lado);

      if (lado == 'i') {
        if (fichas.get(ficha).getCaraIzq() == mesaActual.get(0).getCaraIzq()) {
          fichas.get(ficha).girar();
          mesaActual.add(0, fichas.get(ficha));
          fichas.remove(ficha);
          return;
        } else if (fichas.get(ficha).getCaraDer() == mesaActual.get(0).getCaraIzq()) {
          mesaActual.add(0, fichas.get(ficha));
          fichas.remove(ficha);
          return;
        }
      } else if (lado == 'd') {
        if (fichas.get(ficha).getCaraIzq() == mesaActual.get(mesaActual.size() - 1).getCaraDer()) {
          mesaActual.add(fichas.get(ficha));
          fichas.remove(ficha);
          return;
        } else if (fichas.get(ficha).getCaraDer() == mesaActual.get(mesaActual.size() - 1).getCaraDer()) {
          fichas.get(ficha).girar();
          mesaActual.add(fichas.get(ficha));
          fichas.remove(ficha);
          return;
        }
      }

      System.out.println("No puedes jugar esa ficha. Intenta de nuevo.");
    } while (true);
  }
}
```

9.4.4. Bot.java

```
package domino;

import java.util.ArrayList;

/**
 * Esta clase representa un jugador bot de domino
 */

public class Bot extends Jugador {
  /**
    * Metodo constructor
    * @param nombre Nombre del bot
    */
  public Bot(String nombre) {
    this.nombre = nombre;
  }

  /**
    * Metodo para el primer turno del bot
    * @param mesaActual Mesa actual del juego
    */
  @Override
  public void primerTurno(ArrayList<Ficha> mesaActual) {
    int ficha = -1;
    int max = -1;

    for (Ficha f : fichas) {
      if (f.esMula() && f.getSuma() > max) {
        max = f.getSuma();
        ficha = fichas.indexOf(f);
      }
    }

    if (ficha != -1)
      System.out.println(nombre + " juega la mula mas alta: " + fichas.get(ficha));
    else {
      for (int i = 0; i < fichas.size(); i++) {
        if (fichas.get(i).getSuma() > max) {
          max = fichas.get(i).getSuma();
          ficha = i;
        }
      }
      System.out.println(nombre + " juega la ficha mas alta: " + fichas.get(ficha));
    }

    mesaActual.add(fichas.get(ficha));
    fichas.remove(ficha);

    try {
      Thread.sleep(3000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  /**
    * Metodo para el turno del bot
    * @param mesaActual Mesa actual del juego
    */
  @Override
  public void turno(ArrayList<Ficha> mesaActual) {
    try {
      Thread.sleep(3000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
```

```
    for (int i = 0; i < fichas.size(); i++) {
      if (fichas.get(i).getCaraIzq() == mesaActual.get(0).getCaraIzq()) {
        fichas.get(i).girar();
        mesaActual.add(0, fichas.get(i));
        System.out.println(nombre + " juega " + fichas.get(i) + " a la izquierda");
        fichas.remove(i);
        break;
      } else if (fichas.get(i).getCaraDer() == mesaActual.get(0).getCaraIzq()) {
        mesaActual.add(0, fichas.get(i));
        System.out.println(nombre + " juega " + fichas.get(i) + " a la izquierda");
        fichas.remove(i);
        break;
      } else if (fichas.get(i).getCaraIzq() == mesaActual.get(mesaActual.size() - 1).getCaraDer()) {
        mesaActual.add(fichas.get(i));
        System.out.println(nombre + " juega " + fichas.get(i) + " a la derecha");
        fichas.remove(i);
        break;
      } else if (fichas.get(i).getCaraDer() == mesaActual.get(mesaActual.size() - 1).getCaraDer()) {
        fichas.get(i).girar();
        mesaActual.add(fichas.get(i));
        System.out.println(nombre + " juega " + fichas.get(i) + " a la derecha");
        fichas.remove(i);
        break;
      }
    }

    try {
      Thread.sleep(3000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

9.4.5. Mesa.java

```
package domino;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

/**
 * Esta clase representa la mesa de juego
 */

public class Mesa {
  private static ArrayList<Ficha> mesa = new ArrayList<Ficha>();
  private static ArrayList<Ficha> pozo = new ArrayList<Ficha>();
  private static List<Jugador> jugadores = new ArrayList<Jugador>() {{
    add(new Humano("Jugador 1"));
    add(new Bot("Bot 1"));
    add(new Bot("Bot 2"));
  }};

  /**
    * Metodo constructor
    * @param tipoJuego Tipo de juego (1: Jugador vs Bot, 2: Bot vs Bot)
    */
  public Mesa(int tipoJuego) {
    for (int i = 0; i <= 6; i++)
      for (int j = i; j <= 6; j++)
        mesa.add(new Ficha(i, j));
    this.mezclar();

    if (tipoJuego == 1)
      jugadores.remove(2);
```

```
    else if (tipoJuego == 2)
      jugadores.remove(0);

    this.repartir();
    this.decidirPrimerTurno();
    mesa.clear();
  }

  /**
    * Metodo getter para las fichas de la mesa
    * @return Fichas de la mesa
    */
  public ArrayList<Ficha> getMesa() {
    return mesa;
  }

  /**
    * Metodo getter para las fichas del pozo
    * @return Fichas del pozo
    */
  public ArrayList<Ficha> getPozo() {
    return pozo;
  }

  /**
    * Metodo getter para el ArrayList de jugadores
    * @return Jugadores
    */
  public List<Jugador> getJugadores() {
    return jugadores;
  }

  /**
    * Metodo para mezclar las fichas de la mesa
    */
  private void mezclar() {
    Random random = new Random();
    for (int i = 0; i < mesa.size(); i++) {
      int j = random.nextInt(mesa.size());
      Ficha aux = mesa.get(i);
      mesa.set(i, mesa.get(j));
      mesa.set(j, aux);
    }
  }

  /**
    * Metodo para repartir 7 fichas a cada jugador
    */
  private void repartir() {
    for (int i = 0; i < 7; i++) {
      jugadores.get(0).agregarFicha(mesa.get(i));
      jugadores.get(1).agregarFicha(mesa.get(i + 7));
    }
    for (int i = 14; i < mesa.size(); i++)
      pozo.add(mesa.get(i));
  }

  /**
    * Metodo para decidir quien empieza el juego
    */
  private void decidirPrimerTurno() {
    int max = -1, mano = -1;

    // Busca la mula con el numero mas alto
    for (int i = 0; i < 7; i ++) {
      if (jugadores.get(0).getFicha(i).esMula() && jugadores.get(0).getFicha(i).getSuma() > max) {
        max = jugadores.get(0).getFicha(i).getSuma();
        mano = 0;
      }
```

```
      if (jugadores.get(1).getFicha(i).esMula() && jugadores.get(1).getFicha(i).getSuma() > max) {
        max = jugadores.get(1).getFicha(i).getSuma();
        mano = 1;
      }
    }

    if (mano != -1) {
      if (mano == 1)
        cambioDeTurno();
      return;
    }

    max = -1;
    for (int i = 0; i < 7; i ++) {
      if (jugadores.get(0).getFicha(i).getSuma() > max) {
        max = jugadores.get(0).getFicha(i).getSuma();
        mano = 0;
      }
      if (jugadores.get(1).getFicha(i).getSuma() > max) {
        max = jugadores.get(1).getFicha(i).getSuma();
        mano = 1;
      }
    }

    if (mano == 1)
      cambioDeTurno();
  }

  /**
    * Metodo para cambiar el turno de los jugadores
    */
  public void cambioDeTurno() {
    Jugador aux = jugadores.get(1);
    jugadores.set(1, jugadores.get(0));
    jugadores.set(0, aux);
  }

  /**
    * Metodo para imprimir las fichas de la mesa
    */
  public void imprimir() {
    for (Ficha ficha : mesa)
      System.out.print(ficha + "\t");
    System.out.println();
  }

  /**
    * Metodo para imprimir las fichas de un ArrayList
    * @param fichas ArrayList de fichas
    */
  public void imprimir(ArrayList<Ficha> fichas) {
    for (int i = 1; i <= fichas.size(); i++)
      System.out.print(" " + i + "\t");
    System.out.println();

    for (Ficha ficha : fichas)
      System.out.print(ficha + "\t");
    System.out.println();
  }
}
```

9.4.6. Domino.java

```
package domino;

import java.util.Scanner;
```

```
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;

/**
 * Esta clase representa el juego de domino
 */

public class Domino {
  /**
    * Metodo main.
    * Contiene la logica del juego.
    * @param args Argumentos de la linea de comandos
    */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Clip musica = null;

    try {
      musica = AudioSystem.getClip();
      musica.open(AudioSystem.getAudioInputStream(Domino.class.getResource("/recursos/MoonlightSonata.wav")));
      musica.loop(Clip.LOOP_CONTINUOUSLY);
    } catch (Exception e) {
      e.printStackTrace();
    }

    System.out.println("%%%%%%%%%%%%%%% Bienvenido a Domino %%%%%%%%%%%%%%%");
    System.out.println("Escoga el modo de juego:");
    System.out.println("1. Jugador vs Bot");
    System.out.println("2. Bot vs Bot");
    System.out.print("$ ");
    int tipoJuego = sc.nextInt();
    Mesa mesa = new Mesa(tipoJuego);

    while ((mesa.getJugadores().get(0).puedeJugar() ||
           mesa.getJugadores().get(1).puedeJugar()) ||
           !mesa.getPozo().isEmpty()) {
      try {
        new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
      } catch (Exception e) {
        e.printStackTrace();
      }

      System.out.println("\n\nMesa actual:");
      mesa.imprimir();
      System.out.println("\nNumero de fichas en el Pozo: " + mesa.getPozo().size());

      System.out.println("\n%%%%% Turno de " + mesa.getJugadores().get(0).getNombre() + " %%%%%");
      mesa.imprimir(mesa.getJugadores().get(0).getFichas());

      if (mesa.getMesa().isEmpty())
        mesa.getJugadores().get(0).primerTurno(mesa.getMesa());
      else {
        if (mesa.getJugadores().get(0).puedeJugar(mesa.getMesa()))
            mesa.getJugadores().get(0).turno(mesa.getMesa());
        else {
          if (mesa.getPozo().isEmpty()) {
            System.out.println("No puedes jugar y no hay fichas en el pozo, pasas tu turno");

            try {
              Thread.sleep(3000);
            } catch (InterruptedException e) {
              e.printStackTrace();
            }
          } else {
            if (mesa.getJugadores().get(0) instanceof Humano)
                System.out.println("Necesitas robar una ficha");

            try {
              Thread.sleep(3000);
```

```
            } catch (InterruptedException e) {
              e.printStackTrace();
            }

            System.out.println(mesa.getJugadores().get(0).getNombre() + " roba 1 ficha y pasa su turno");
            mesa.getJugadores().get(0).robar(mesa.getPozo());
            mesa.imprimir(mesa.getJugadores().get(0).getFichas());

            try {
              Thread.sleep(3000);
            } catch (InterruptedException e) {
              e.printStackTrace();
            }
          }
        }
      }

      if (mesa.getJugadores().get(0).getCantidadFichas() == 0) {
        System.out.println("\n\nMesa final:");
        mesa.imprimir(mesa.getMesa());
        break;
      }

      mesa.cambioDeTurno();
    }

    try {
      if (musica != null)
        musica.stop();

      musica = AudioSystem.getClip();
      musica.open(AudioSystem.getAudioInputStream(Domino.class.getResource("/recursos/Victoria.wav")));
      musica.start();
    } catch (Exception e) {
      e.printStackTrace();
    }

    if (!mesa.getJugadores().get(0).puedeJugar() &&
            !mesa.getJugadores().get(1).puedeJugar() &&
            mesa.getPozo().isEmpty() ) {
      int suma1 = 0, suma2 = 0;
      System.out.println("Se acabaron las fichas");

      for (int i = 0; i < mesa.getJugadores().get(0).getCantidadFichas(); i++)
        suma1 += mesa.getJugadores().get(0).getFicha(i).getSuma();
      for (int i = 0; i < mesa.getJugadores().get(1).getCantidadFichas(); i++)
        suma2 += mesa.getJugadores().get(1).getFicha(i).getSuma();

      if (suma1 < suma2)
        System.out.println("El ganador es " + mesa.getJugadores().get(0).getNombre());
      else if (suma1 > suma2)
        System.out.println("El ganador es " + mesa.getJugadores().get(1).getNombre());
      else
        System.out.println("Empate");
    } else
      System.out.println("\n%%%%%%%%%% El ganador es " + mesa.getJugadores().get(0).getNombre() + " %%%%%%%%%%");

    try {
      Thread.sleep(5000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    sc.close();
  }
}
```

9.5. Pruebas de funcionamiento

Ejecutando el archivo *Domino.jar* con el comando *java -jar Domino.jar* se muestra el menú de opciones de juego.

```
%%%%%%%%%%%%%%%% Bienvenido a Domino %%%%%%%%%%%%%%%%
Escoga el modo de juego:
1. Jugador vs Bot
2. Bot vs Bot
$
```

Figura 14: Menú de opciones de juego

Al seleccionar una opción se limpia la terminal y se inicia el juego tirando la mula más alta o la ficha más alta.

```
Mesa actual:


Numero de fichas en el Pozo: 14

%%%%% Turno de Bot 2 %%%%%
  1       2       3       4       5       6       7
[4|4]   [4|5]   [1|5]   [2|6]   [5|5]   [1|3]   [0|6]
Bot 2 juega la mula mas alta: [5|5]
```

Figura 15: Inicio del juego

Los siguientes turnos son diferentes dependiendo del tipo de jugador. En caso de que sea un bot se espera 3 segundos antes de que juegue su ficha. En caso de que sea un jugador humano se le pide que seleccione la ficha que quiere jugar y el lado donde la quiere jugar.

```
Mesa actual:
[1|6]   [6|6]

Numero de fichas en el Pozo: 14

%%%%% Turno de Jugador 1 %%%%%
  1       2       3       4       5       6
[1|4]   [1|2]   [0|3]   [0|4]   [4|4]   [2|6]
¿Qué ficha quieres jugar? (1 - 6)
6
¿Dónde la quieres jugar? (izquierda o derecha)
d
```

Figura 16: Turno de jugador Humano

```
Mesa actual:
[1|6]   [6|6]   [6|2]

Numero de fichas en el Pozo: 14

%%%%% Turno de Bot 1 %%%%%
  1       2       3       4       5       6
[2|4]   [2|3]   [0|1]   [2|2]   [1|3]   [3|4]
Bot 1 juega [2|4] a la derecha
```

Figura 17: Turno de jugador Bot

En caso de que un jugador necesite robar una ficha se muestra un mensaje indicandolo y se vuelven a imprimir sus fichas.

```
Mesa actual:
[2|4]   [4|6]   [6|6]   [6|1]   [1|0]   [0|5]   [5|4]

Numero de fichas en el Pozo: 14

%%%%% Turno de Bot 2 %%%%%
  1       2       3       4
[1|3]   [3|5]   [0|0]   [3|6]
Bot 2 roba 1 ficha y pasa su turno
  1       2       3       4       5
[1|3]   [3|5]   [0|0]   [3|6]   [4|4]
```

Figura 18: Mensaje de robar ficha

Al ganar el jugador se muestra un mensaje de victoria y se termina el programa.

```
Mesa final:
  1       2       3       4       5       6       7       8       9       10      11      12      13
[0|3]   [3|1]   [1|2]   [2|3]   [3|4]   [4|0]   [0|1]   [1|6]   [6|6]   [6|2]   [2|4]   [4|4]   [4|1]

%%%%%%%%%% El ganador es Jugador 1 %%%%%%%%%%
```

Figura 19: Mensaje de victoria

9.6. Documentación de JavaDoc

Al ejecutar el comando *javadoc -d documentacion domino/*.java* se genera la documentación de JavaDoc en la carpeta *documentacion*. Algunos ejemplos de la documentación generada se muestran a continuación y lo restante se encuentra en la dirección https://documentacion-domino.netlify.app.

Figura 20: Página principal

Figura 21: Árbol de jerarquia de clases

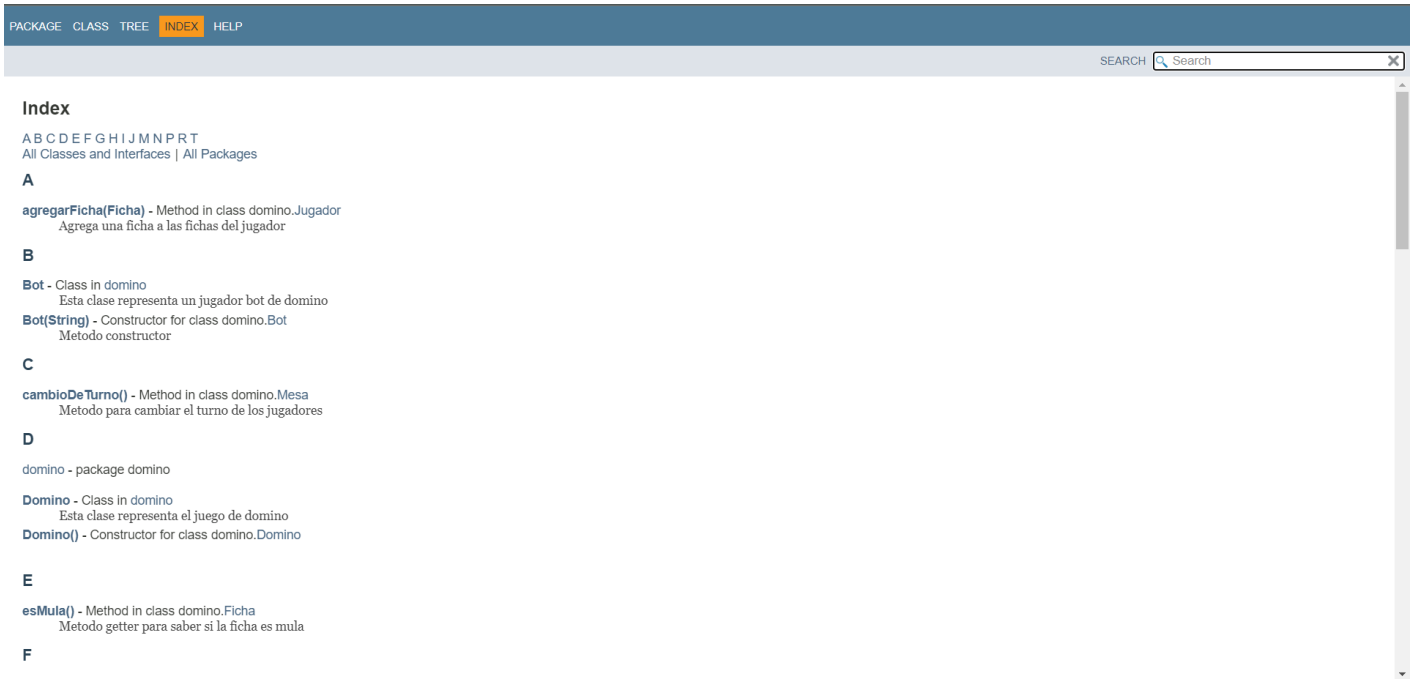

Figura 22: Índice alfabetico

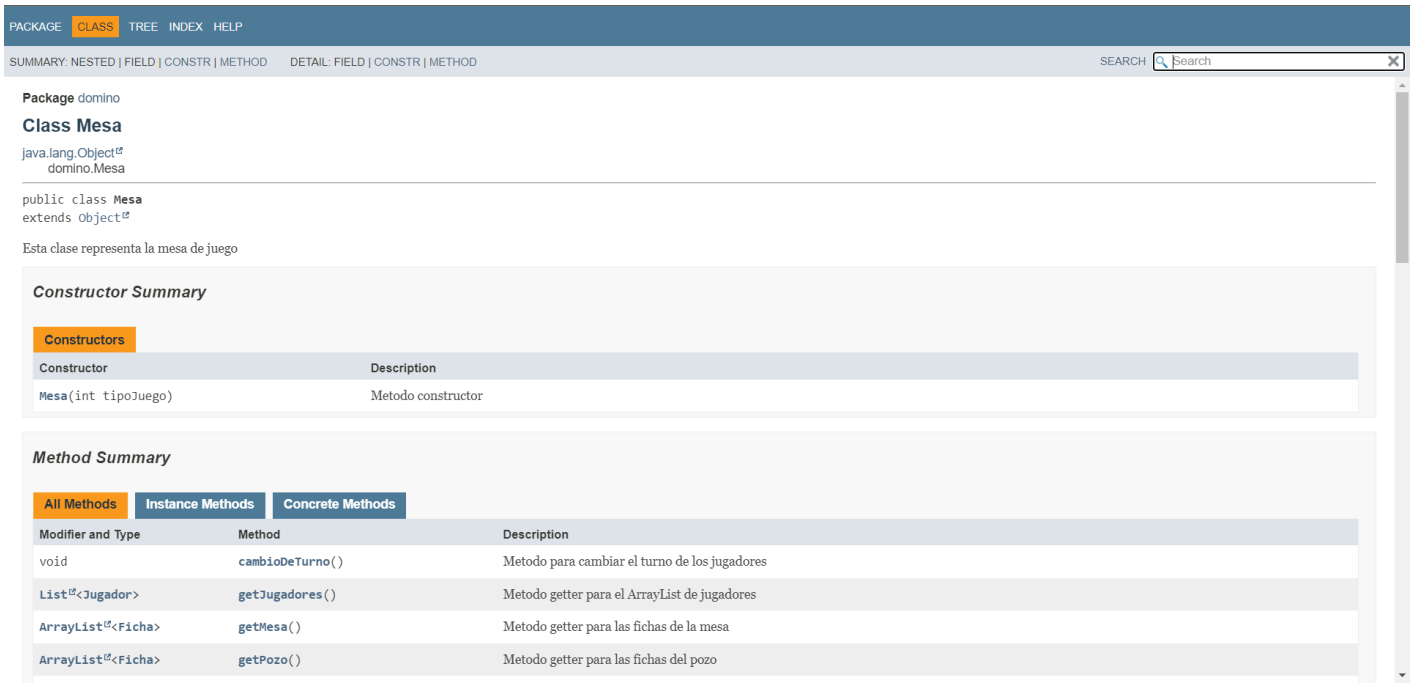

Figura 23: Página de la clase *Mesa*