Trabajo Práctico N° 1 - Programación de Videojuegos I

Fase de análisis:

Historia de usuario HO01:
Snake

Quién: Como jugador.
Qué: Quiero mover la serpiente por el tablero usando las teclas de dirección.
Para qué: Para poder crecer al comer animales y mantener el juego en marcha.

Tabla EPS

Entrada	Proceso	Salida
Pulsación de teclas (↑, ↓, ←, →)	Actualizar la dirección de movimiento de la serpiente.	Cambio de dirección en pantalla.
Posición actual de la cabeza de la serpiente	Calcular nueva posición sumando la dirección.	La cabeza se mueve en el tablero.
Lista de posiciones de la serpiente	Actualizar cuerpo siguiendo a la cabeza.	Serpiente se desplaza con todo su cuerpo.
Colisión con sí misma o bordes	Verificar condiciones de pérdida.	Fin del juego si hay colisión.

Historia de usuario HO02:
Animales

Quién: Como jugador.
Qué: Quiero que aparezcan animales (comida) en el tablero.
Para qué: Para que la serpiente pueda comerlos, crecer y aumentar el puntaje.

Tabla EPS

Entrada	Proceso	Salida
Posición aleatoria dentro del tablero	Generar un nuevo objeto Animal en esa posición.	Un animal aparece en pantalla.
Posición de la cabeza de la serpiente	Verificar si coincide con la del animal.	Detección de colisión con comida.
Colisión detectada	Destruir al animal y notificar al Snake que crezca.	Serpiente aumenta su tamaño y puntaje sube.

Estado del tablero	Comprobar que no se genere un animal en la misma posición de la serpiente.	Aparición válida de nuevos animales.

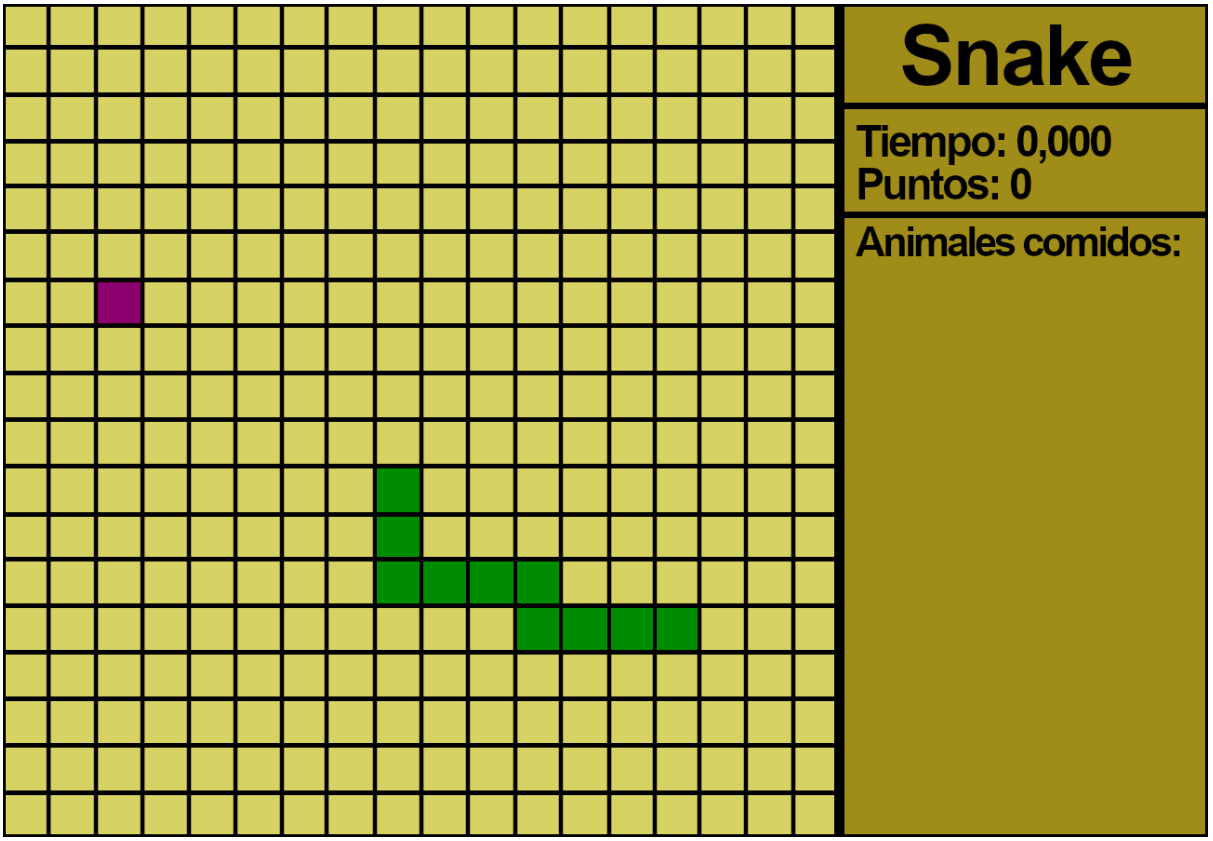
Historia de usuario HU03:
HUD

Quién: Como jugador.
Qué: Quiero ver mi puntaje y estado del juego en pantalla.
Para qué: Para saber cuántos puntos tengo y cuándo he perdido.

Tabla EPS

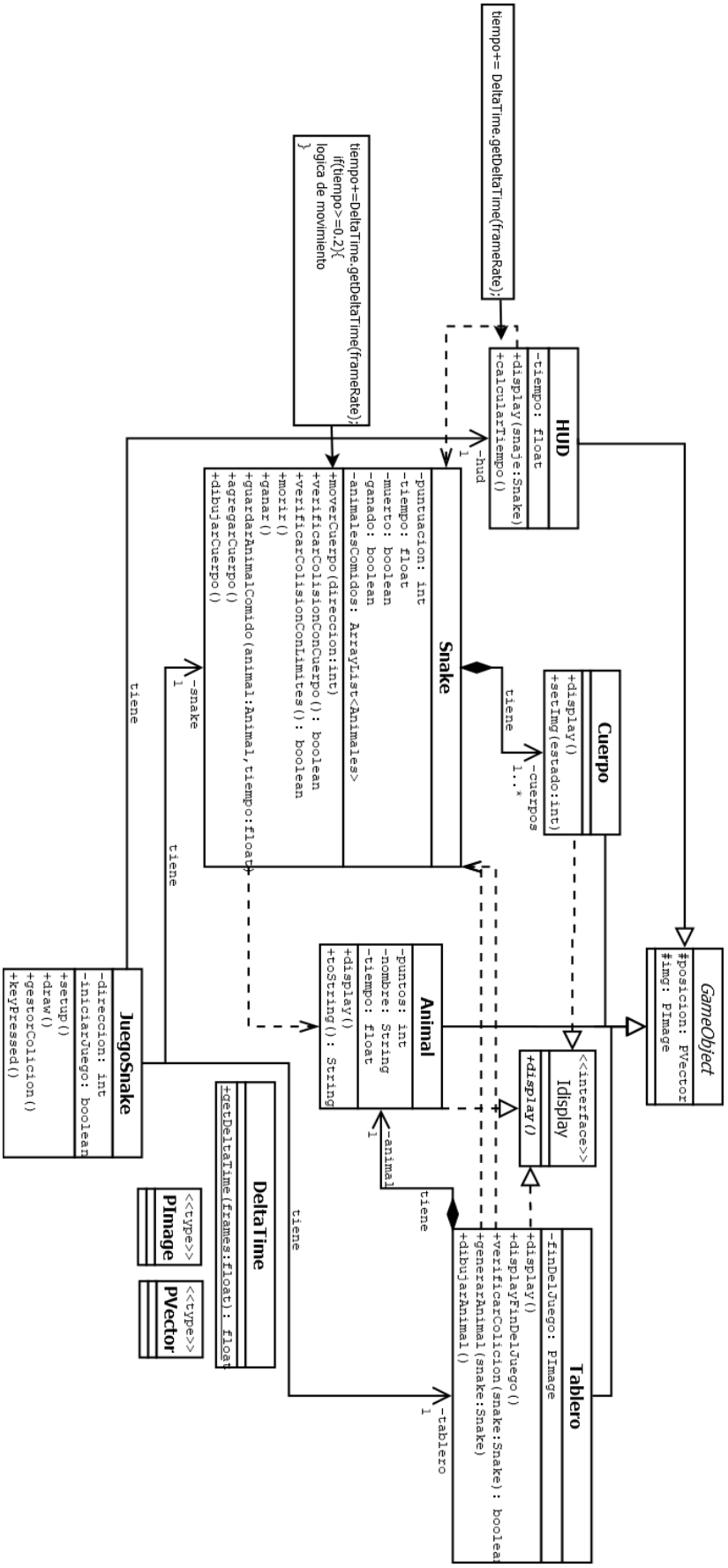
Entrada	Proceso	Salida
Variable de puntaje del juego	Actualizar texto en pantalla.	Puntaje visible al jugador.
Estado del juego (jugando / game over)	Determinar qué mensaje mostrar.	“Game Over” o puntaje final.
Posición en pantalla	Dibujar HUD en el lugar definido (esquinas, centro).	Información siempre visible al jugador.

Boceto:



Fase de diseño:

Diagrama de clases:



HU01: Snake

Objetivo central: Implementar la serpiente que se mueve en el tablero y crece con nuevos segmentos (Cuerpo).

Lista de Tareas

- ☐ Diseñar la estructura de la serpiente con cabeza y lista de segmentos de cuerpo.
- ☐ Implementar el movimiento de la cabeza según la entrada del teclado.
- ☐ Hacer que los segmentos sigan la posición de la cabeza (actualizar lista de Cuerpo).
- ☐ Implementar detección de colisión contra sí misma y contra los bordes del tablero.
- ☐ Probar en ejecución que la serpiente responde al input y se desplaza correctamente.

Criterios de Aceptación

- ☐ El jugador debe poder mover la serpiente en las 4 direcciones cardinales con el teclado.
- ☐ La serpiente debe moverse de forma continua, manteniendo la coherencia entre cabeza y cuerpo.
- ☐ Al chocar con su propio cuerpo o con los bordes, el juego debe terminar.
- ☐ Al crecer, los nuevos segmentos deben añadirse correctamente al final de la serpiente.

HU02: Animales

Objetivo central: Implementar animales como elementos recolectables que hacen crecer la serpiente y aumentan el puntaje.

Lista de Tareas

- ☐ Diseñar la clase Animal con atributos de posición y tamaño.
- ☐ Implementar generación de animales en posiciones aleatorias del tablero.
- ☐ Verificar que los animales no aparezcan sobre la serpiente (ni su cabeza ni su cuerpo).
- ☐ Detectar colisión entre la cabeza de la serpiente y un animal.
- ☐ Al producirse la colisión, eliminar el animal y notificar al Snake que debe crecer y sumar puntaje.
- ☐ Generar un nuevo animal tras cada recolección.

Criterios de Aceptación

- ☐ Siempre debe haber un animal visible en el tablero durante la partida.
- ☐ Cuando la serpiente toca un animal, este desaparece.
- ☐ La serpiente debe crecer en longitud al comer un animal.
- ☐ El puntaje debe aumentar en 1 (o la cantidad definida) al recoger un animal.

HU03: HUD

Objetivo central: Mostrar al jugador información sobre el estado del juego (puntaje y mensajes clave).

Lista de Tareas

- ☐ Diseñar la clase HUD con capacidad de mostrar puntaje y estado del juego.
- ☐ Implementar actualización visual del puntaje cada vez que se recolecta un animal.
- ☐ Implementar un temporizador que inicie junto con el juego y que limite el tiempo del juego a 60s
- ☐ Mostrar mensajes en pantalla al inicio y al final de la partida (ejemplo: “Game Over”).
- ☐ Definir posiciones fijas para los elementos del HUD (ejemplo: esquina superior para puntaje, centro para mensajes).
- ☐ Probar que el HUD se actualice en tiempo real mientras se juega.

Criterios de Aceptación

- ☐ El HUD debe mostrar siempre el puntaje actualizado del jugador.
- ☐ Debe mostrarse un mensaje claro cuando el juego termina.
- ☐ El HUD no debe interferir con la jugabilidad (no tapar la serpiente ni los animales).
- ☐ La información debe ser legible en todo momento.
- ☐ Si el contador llega a 60s el juego termina.

Documentación del código:

Clase JuegoSnake (Sketch)

```
// Variables principales del juego
private Snake snake;         // Objeto que representa la serpiente
private Tablero tablero;     // Objeto que maneja el tablero y los
animales
private HUD hud;             // Interfaz de usuario que muestra puntaje
y tiempo
private int direccion;       // Dirección actual de movimiento de la
serpiente
private boolean iniciarJuego; // Controla si el juego comenzó tras
presionar una tecla

// Método de inicialización del juego
public void setup(){
  size(1300,900);                    // Define el tamaño de la ventana
  snake = new Snake();               // Crea la serpiente
  tablero = new Tablero(new PVector()); // Crea el tablero con un animal
inicial
  hud = new HUD(new PVector(900,0));  // Crea el HUD en la posición
```

```
indicada
  frameRate(60);                        // Establece la tasa de
actualización
  iniciarJuego=false;                   // El juego comienza pausado
  direccion=0;                          // Dirección inicial (sin
movimiento)
}

// Método principal que se ejecuta en cada frame
public void draw(){
  tablero.display();                    // Dibuja el tablero
  snake.dibujarCuerpo();                // Dibuja la serpiente
  tablero.dibujarAnimal();              // Dibuja el animal en pantalla
  snake.moverCuerpo(direccion);         // Mueve la serpiente según la
dirección
  hud.display(snake);                   // Muestra el HUD actualizado
  gestorColicion();                     // Gestiona colisiones con
animales

  // Si el jugador empezó a moverse, se inicia el conteo del tiempo
  if(iniciarJuego){
    hud.calcularTiempo();
  }

  // Verifica condiciones de fin del juego
  if(snake.isMuerto() || snake.isGanado()){
    tablero.displayFinDelJuego();       // Muestra mensaje de fin
    noLoop();                           // Detiene el ciclo principal
  }

  // Verifica si la serpiente choca consigo misma o con los límites
  if (snake.verificarColisionConCuerpo() ||
snake.verificarColisionConLimites()) {
    snake.morir();
  }

  // Condición de victoria por tiempo (60 segundos)
  if(hud.getTiempo() >= 60){
    snake.ganar();
  }
}

// Método que gestiona las colisiones entre la serpiente y los animales
public void gestorColicion(){
  boolean colicion = tablero.verificarColicion(snake); // Detecta
colisión
```

```
  if(colicion){
    // Guarda información del animal comido (con tiempo incluido)
    snake.guardarAnimalComido(tablero.getAnimal(), hud.getTiempo());
    tablero.generarAnimal(snake);     // Genera un nuevo animal en el
tablero
    snake.agregarCuerpo();            // La serpiente crece
  }
}

// Manejo de entradas por teclado
public void keyPressed(){
  // Movimiento hacia la derecha (evita movimiento contrario inmediato)
  if(keyCode == RIGHT && direccion!= 2){
    direccion=1;
    iniciarJuego=true;
  }
  // Movimiento hacia la izquierda
  else if(keyCode == LEFT && direccion!=1){
    direccion=2;
    iniciarJuego=true;
  }
  // Movimiento hacia abajo
  else if(keyCode == DOWN && direccion!=4){
    direccion=3;
    iniciarJuego=true;
  }
  // Movimiento hacia arriba
  else if(keyCode == UP && direccion!=3){
    direccion=4;
    iniciarJuego=true;
  }

  // Si el juego terminó, se reinicia al presionar ENTER
  if(snake.isMuerto() || snake.isGanado()){
    if(keyCode == ENTER){
      setup(); // Reinicia el juego
      loop();  // Reactiva el bucle draw()
    }
  }
}
```

Clase Snake

```
// Clase Snake
// Representa a la serpiente controlada por el jugador.
```

```
// Maneja el movimiento, colisiones, crecimiento, estados
(muerto/ganado) y registro de animales comidos.
class Snake {

  // Atributos principales
  private int puntuacion;                          // Puntaje acumulado
por los animales comidos
  private float tiempo;                            // Controla el
intervalo de movimiento
  private boolean muerto;                          // Indica si la
serpiente ha muerto
  private boolean ganado;                          // Indica si la
serpiente ha ganado
  private ArrayList<Cuerpo> cuerpos = new ArrayList<Cuerpo>();        //
Segmentos que componen la serpiente
  private ArrayList<Animal> animalesComidos = new ArrayList<Animal>();
// Registro de animales comidos

  // Constructor: inicializa la serpiente con una cabeza en el centro
aproximado
  public Snake(){
    cuerpos.add(new Cuerpo(new PVector(width/2-250, height/2))); //
Posición inicial de la cabeza
    puntuacion = 0;   // Puntaje inicial
    tiempo = 0;       // Temporizador para el movimiento
  }

  // Método que mueve la serpiente en función de la dirección actual
  public void moverCuerpo(int direccion) {
    tiempo += DeltaTime.getDeltaTime(frameRate); // Acumula tiempo con
DeltaTime
    if(tiempo >= 0.2){ // Controla la velocidad (un movimiento cada 0.2
seg aprox.)

      // Los segmentos siguen la posición del anterior, de atrás hacia
adelante
      for (int i = cuerpos.size() - 1; i >= 1; i--) {
        cuerpos.get(i).setPosicion(cuerpos.get(i - 1).getPosicion());
      }

      // Copia la posición de la cabeza para actualizarla según la
dirección
      PVector cabezaPos = cuerpos.get(0).getPosicion().copy();

      // Movimiento de la cabeza según dirección
      switch (direccion) {
```

```
        case 1: // Derecha
          if(direccion != 2){
            cabezaPos.add(new PVector(50, 0));
            break;
          }
        case 2: // Izquierda
          if(direccion != 1){
            cabezaPos.add(new PVector(-50, 0));
            break;
          }
        case 3: // Abajo
          cabezaPos.add(new PVector(0, 50));
          break;
        case 4: // Arriba
          cabezaPos.add(new PVector(0, -50));
          break;
      }

      // Actualiza la posición de la cabeza
      cuerpos.get(0).setPosicion(cabezaPos);
      tiempo = 0; // Reinicia el temporizador
    }
  }

  // Verifica si la cabeza colisiona con el cuerpo (a partir del tercer
segmento)
  public boolean verificarColisionConCuerpo() {
    if (cuerpos.size() < 2) {
      return false;
    }
    for (int i = 2; i < cuerpos.size(); i++) {
      if
(cuerpos.get(0).getPosicion().equals(cuerpos.get(i).getPosicion())) {
        return true;
      }
    }
    return false;
  }

  // Verifica si la cabeza está fuera de los límites del tablero
  public boolean verificarColisionConLimites() {
    PVector posicionCabeza = cuerpos.get(0).getPosicion();

    if (posicionCabeza.x < 0 || posicionCabeza.x >= 900 ||
        posicionCabeza.y < 0 || posicionCabeza.y >= 900) {
      return true;
```

```
    }
    return false;
  }

  // Cambia las imágenes del cuerpo al estado de "muerto" y marca el
estado
  public void morir(){
    for(Cuerpo cuer : cuerpos){
      cuer.setImg(1);
      muerto = true;
    }
  }

  // Cambia las imágenes del cuerpo al estado de "ganador" y marca el
estado
  public void ganar(){
    for(Cuerpo cuer : cuerpos){
      cuer.setImg(2);
      ganado = true;
    }
  }

  // Registra un animal comido: se guarda, se anota el tiempo y se suman
sus puntos
  public void guardarAnimalComido(Animal animal, float tiempo){
    animalesComidos.add(animal);
    animal.setTiempo(tiempo = round(tiempo * 1000) / 1000.0); //
Redondea el tiempo a 3 decimales
    puntuacion += animal.getPuntos();
  }

  // Agrega un nuevo segmento al final del cuerpo (cuando se come un
animal)
  public void agregarCuerpo(){
    cuerpos.add(new
Cuerpo(cuerpos.get(cuerpos.size()-1).getPosicion()));
  }

  // Dibuja todos los segmentos de la serpiente
  public void dibujarCuerpo(){
    for(Cuerpo cuer : cuerpos){
      cuer.display();
    }
  }

  // Getters
```

```
  public ArrayList<Cuerpo> getCuerpos(){
    return cuerpos;
  }

  public ArrayList<Animal> getAnimalesComidos(){
    return animalesComidos;
  }

  public int getPuntuacion(){
    return puntuacion;
  }

  public boolean isMuerto(){
    return muerto;
  }

  public boolean isGanado(){
    return ganado;
  }
}
```

Clase Cuerpo

```
// La clase Cuerpo representa un segmento del cuerpo de la serpiente.
// Extiende de GameObject (tiene posición e imagen) e implementa
Idisplay (se puede dibujar en pantalla).
class Cuerpo extends GameObject implements Idisplay {

  // Constructor: recibe la posición inicial del segmento
  public Cuerpo(PVector posicion){
    super(posicion);               // Llama al constructor de GameObject
para inicializar la posición
    img = loadImage("cuerpo.png"); // Imagen por defecto del segmento
  }

  // Dibuja el segmento del cuerpo en su posición
  public void display(){
    image(img, posicion.x, posicion.y);
  }

  // Setter para actualizar la posición del segmento
  public void setPosicion(PVector posicion){
    this.posicion = posicion;
  }
```

```
  // Setter para cambiar la imagen del segmento según el estado del
juego
  public void setImg(int estado){
    if(estado == 1){
      img = loadImage("cuerpomuerto.png"); // Imagen cuando la serpiente
muere
    } else if(estado == 2){
      img = loadImage("cuerpoganar.png");  // Imagen cuando la serpiente
gana
    }
  }

  // Getter de la posición actual del segmento
  public PVector getPosicion(){
    return posicion;
  }
}
```

Clase Animal

```
// La clase Animal representa un objeto de comida en el tablero.
// Extiende de GameObject (tiene posición e imagen) e implementa la
interfaz Idisplay (debe poder dibujarse en pantalla).
class Animal extends GameObject implements Idisplay {
  // Atributos propios de un Animal
  private int puntos;       // Valor en puntos que otorga al ser comido
  private String nombre;    // Nombre identificador del animal
  private float tiempo;     // Tiempo en el que fue recogido por la
serpiente (si aplica)

  // Constructor: recibe la posición, la imagen, los puntos y el nombre
  public Animal(PVector posicion, PImage img, int puntos, String
nombre){
    super(posicion, img);   // Llama al constructor de GameObject para
inicializar posición e imagen
    this.puntos = puntos;   // Asigna valor en puntos
    this.nombre = nombre;   // Asigna nombre
    tiempo = 0;             // Inicializa tiempo en 0 (sin haber sido
recogido aún)
  }

  // Método display(): dibuja al animal en pantalla en su posición
  public void display(){
    image(img, posicion.x, posicion.y);
  }
```

```
  // Método toString(): devuelve un texto con información del animal
  public String toString() {
    return tiempo + " : " + nombre +
           " en " + (int)round(posicion.x) + ", " +
(int)round(posicion.y) +
           " - Puntos: " + puntos;
  }

  // Getter de la posición (utilizado para verificar colisiones)
  public PVector getPosicion(){
    return posicion;
  }

  // Getter de puntos (para sumarlos al puntaje al ser comido)
  public int getPuntos(){
    return puntos;
  }

  // Setter del tiempo en el que fue recogido
  public void setTiempo(float tiempo){
    this.tiempo = tiempo;
  }
}
```

Clase madre GameObjet

```
// Clase abstracta GameObject
// Sirve como clase base para todos los objetos del juego que tienen
posición e imagen.
// No se puede instanciar directamente (abstract), solo puede ser
extendida por otras clases.
abstract class GameObject {

  // Atributos comunes a todos los objetos del juego
  protected PVector posicion; // Posición del objeto en el tablero
  protected PImage img;       // Imagen asociada al objeto (sprite)

  // Constructor 1: inicializa solo con posición
  public GameObject(PVector posicion){
    this.posicion = posicion;
  }

  // Constructor 2: inicializa con posición e imagen
  public GameObject(PVector posicion, PImage img){
```

```
    this.posicion = posicion;
    this.img = img;
  }
}
```

Interfaz Idisplay

```
// Interfaz Idisplay
// Define un contrato para que cualquier clase que la implemente
// esté obligada a definir el método display(), encargado de dibujar en
pantalla.
interface Idisplay {

  // Método que deben implementar las clases que se pueden mostrar en
pantalla.
  public void display();
}
```

Clase Tablero

```
// Clase Tablero que extiende GameObject e implementa la interfaz
Idisplay.
class Tablero extends GameObject implements Idisplay {
    private Animal animal; // Variable para almacenar el animal presente
en el tablero.
    private PImage finDelJuego; // Imagen que se muestra al finalizar el
juego.

    // Constructor de la clase Tablero que recibe una posición como
PVector.
    public Tablero(PVector posicion) {
        super(posicion);
        img = loadImage("tablero.png"); // Carga la imagen del tablero.
        finDelJuego = loadImage("findeljuego.png"); // Carga la imagen
de fin del juego.
        generarAnimal(new Snake()); // Inicializa un animal en el
tablero con una serpiente.
    }

    // Método para mostrar la imagen del tablero en su posición.
    public void display() {
        image(img, posicion.x, posicion.y);
    }
```

```
    // Método para mostrar la imagen de fin del juego en una posición
desplazada.
    public void displayFinDelJuego() {
        image(finDelJuego, posicion.x + 100, posicion.y + 100);
    }

    // Método para verificar si la cabeza de la serpiente colisiona con
el animal.
    public boolean verificarColicion(Snake snake) {
        PVector cabezaPos = snake.getCuerpos().get(0).getPosicion(); //
Obtiene la posición de la cabeza de la serpiente.
        PVector animalPos = animal.getPosicion(); // Obtiene la posición
del animal.
        if (cabezaPos.equals(animalPos)) { // Compara si las posiciones
son iguales.
            return true;
        } else {
            return false;
        }
    }

    // Método para generar un nuevo animal en una posición válida en el
tablero.
    public void generarAnimal(Snake snake) {
        PVector nuevaPosicion;
        boolean posicionValida;
        do {
            nuevaPosicion = new PVector((int) random(0, 18) * 50, (int)
random(0, 18) * 50); // Genera una posición aleatoria en una cuadrícula
de 18x18.

            posicionValida = true; // Inicializa la bandera de posición
válida.
            if (animal != null &&
nuevaPosicion.equals(animal.getPosicion())) { // Verifica si la nueva
posición coincide con la del animal actual.
                posicionValida = false;
            }
            for (Cuerpo cuerpo : snake.getCuerpos()) { // Itera sobre
los cuerpos de la serpiente.
                if (nuevaPosicion.equals(cuerpo.getPosicion())) { //
Verifica si la nueva posición coincide con algún cuerpo.
                    posicionValida = false;
                    break;
                }
            }
```

```
        } while (!posicionValida); // Repite hasta encontrar una
posición válida.
        int tipo = (int) random(1, 4); // Genera un número aleatorio
para determinar el tipo de animal.
        if (tipo == 1) {
            animal = new Animal(nuevaPosicion, loadImage("insecto.png"),
3, "insecto"); // Crea un animal de tipo insecto.
        } else if (tipo == 2) {
            animal = new Animal(nuevaPosicion, loadImage("raton.png"),
1, "raton"); // Crea un animal de tipo ratón.
        } else if (tipo == 3) {
            animal = new Animal(nuevaPosicion, loadImage("pajaro.png"),
2, "pajaro"); // Crea un animal de tipo pájaro.
        }
    }

    // Método para dibujar el animal en el tablero.
    public void dibujarAnimal() {
        animal.display();
    }

    // Método getter para obtener el animal actual.
    public Animal getAnimal() {
        return animal;
    }

    // Método setter para establecer un nuevo animal.
    public void setAnimal(Animal animal) {
        this.animal = animal;
    }
}
```

Clase HUD

```
// La clase HUD representa la interfaz gráfica que muestra al jugador la
información del juego.
// Extiende de GameObject porque tiene posición e imagen de fondo
(HUD.png).
class HUD extends GameObject {

  // Atributo que guarda el tiempo transcurrido desde que inicia el
juego
  private float tiempo;

  // Constructor: inicializa posición, imagen y tiempo
```

```
  public HUD(PVector posicion){
    super(posicion);              // Asigna la posición usando el
constructor de GameObject
    img = loadImage("HUD.png");   // Carga la imagen del HUD
    tiempo = 0;                   // Inicializa el tiempo en cero
  }

  // Método que dibuja el HUD en pantalla
  // Recibe como parámetro la serpiente, para acceder a su puntaje y
animales comidos
  public void display(Snake snake){
    image(img, posicion.x, posicion.y); // Dibuja la imagen de fondo del
HUD

    // Configuración de texto
    fill(#352d00);   // Color marrón oscuro
    textSize(50);    // Tamaño grande para valores principales

    // Muestra la puntuación actual de la serpiente
    text(snake.getPuntuacion(), 1100, 212);

    // Muestra el tiempo transcurrido
    text(tiempo, 1100, 165);

    // Muestra la lista de animales comidos con sus datos
    textSize(20);   // Texto más pequeño para detalles
    int posY = 290; // Posición inicial en Y para la lista

    // Recorre la lista de animales comidos y los muestra uno debajo del
otro
    for(int i = 0; i < snake.getAnimalesComidos().size(); i++){
      text(snake.getAnimalesComidos().get(i).toString(), 910, posY);
      posY += 25; // Desplaza la posición vertical para el siguiente
texto
    }
  }

  // Método para calcular el tiempo transcurrido usando DeltaTime
  public void calcularTiempo(){
    tiempo += DeltaTime.getDeltaTime(frameRate);
  }

  // Getter del tiempo actual
  public float getTiempo(){
    return tiempo;
  }
```

```
}
```

Clase DeltaTime

```
// Clase auxiliar para calcular el deltaTime.
// Se declara como 'static' porque sus métodos no requieren instanciar
objetos.
static class DeltaTime {

  // Método estático que devuelve el deltaTime en segundos
  // Recibe como parámetro la cantidad de frames por segundo (fps).
  static float getDeltaTime(float frames){
    int framesPorSegundo = round(frames);   // Redondea los fps a un
entero
    float deltaTime = 1.0 / framesPorSegundo; // Calcula cuánto dura
cada frame (en segundos)
    return deltaTime; // Retorna el tiempo que tarda un frame
  }
}
```

Assets

Snake cuerpo

Animales

Tablero y HUD