Clase SuikaGame

Inicio

Método principal();

Crear ventana con el título "SUIKA GAME";

Establecer operación por defecto al cerrar la ventana a EXIT_ON_CLOSE;

Establecer la ventana como no redimensionable;

Establecer el color de fondo de la ventana a (213, 137, 54);

Crear panelJuego como un nuevo GamePanel;

Añadir panelJuego a la ventana;

Empaquetar la ventana;

Establecer la ubicación de la ventana en el centro;

Hacer la ventana visible;

Iniciar el juego en panelJuego;

Fin del Método

Fin de la Clase

Clase GamePanel extiende JPanel e implementa Runnable

Inicio

Constantes ANCHO = 1280, ALTO = 720, FPS = 60;

Declarar hiloJuego como Thread;

Declarar managerJuego como ManagerJuego;

Declarar fondo como Image;

Constructor GamePanel()

Establecer el tamaño preferido a (ANCHO, ALTO);

Establecer el color de fondo a (213, 137, 54);

Establecer el layout a null;

// Se agregan los controles de juego

Añadir KeyListener con nuevos Controles;

Establecer focusable a true;

Inicializar managerJuego como nuevo ManagerJuego;

Intentar

Leer la imagen de fondo desde el archivo "C:\\Users\\gusta\\Imágenes\\Sprites\\fondo2.jpeg";

Capturar

Imprimir la traza de la excepción;

Fin del Intentar

Fin del Constructor

Método iniciarJuego()

```
    Inicializar hiloJuego como nuevo Thread con this;
    Iniciar hiloJuego;
Fin del Método

Método run()
    Declarar intervalo = 1000000000/FPS;
    Declarar delta = 0;
    Declarar tiempoPrevio = System.nanoTime();
    Declarar tiempoActual;

    Mientras hiloJuego no sea null
        tiempoActual = System.nanoTime();

        delta += (tiempoActual - tiempoPrevio) / intervalo;
        tiempoPrevio = tiempoActual;

        Si delta >= 1
            Llamar al método actualizar();
            Llamar al método repaint();
            delta--;
        Fin del Si
    Fin del Mientras
```

Fin del Método

Método actualizar()

Llamar al método actualizar de managerJuego;

Fin del Método

Método paintComponent(graficos)

Llamar al método paintComponents de la superclase con graficos;

Si fondo no es null

Dibujar la imagen de fondo en graficos;

Fin del Si

Declarar graficos2 como Graphics2D con graficos;

Llamar al método dibujar de managerJuego con graficos2;

Fin del Método

Fin de la Clase

Clase ManagerJuego

Inicio

Constantes ANCHO = 360, ALTO = 550;

Variables estáticas xIzquierda, xDerecha, yArriba, yAbajo;

```
// Fruta
Declarar fruta como Fruta;
Constantes posInicialX, posInicialY;

// Siguiente fruta
Declarar siguienteFruta como Fruta;
Constantes posInitSiguienteX, posInitSiguienteY;

Variable estática frutasCaidas como Lista;

Variable estática caidaFruta = 60; // Intervalo de caída de la fruta

Constructor ManagerJuego()
    // Se dibuja el contenedor de las frutas
    xIzquierda = (GamePanel.ANCHO / 2) - (ANCHO / 2);
    xDerecha = xIzquierda + ANCHO;
    yArriba = 50;
    yAbajo = yArriba + ALTO;

    posInicialX = xIzquierda + (ANCHO / 2) - fruta.diametro; // La fruta se crea en la mitad del eje horizontal del contenedor
    posInicialY = yArriba + fruta.diametro; // Y en la parte superior del contenedor
```

```
    posInitSiguienteX = xIzquierda - 200;
    posInitSiguienteY = yArriba + 50;
    fruta = elegirFruta();
    fruta.establecerPosicion(posInicialX, posInicialY);

    siguienteFruta = elegirFruta();
    siguienteFruta.establecerPosicion(posInitSiguienteX, posInitSiguienteY);
Fin del Constructor

// Método para elegir una fruta al azar
Método privado elegirFruta()
    Declarar fruta como Fruta;
    Declarar random como un número aleatorio entre 0 y 4;

    Si random es 0
        fruta = nuevo Datil con la ruta de la imagen;
    Sino Si random es 1
        fruta = nuevo Cotoperi con la ruta de la imagen;
    Sino Si random es 2
        fruta = nuevo Mamey con la ruta de la imagen;
    Sino Si random es 3
```

```
      fruta = nuevo Cereza con la ruta de la imagen;
    Sino Si random es 4
      fruta = nuevo Pumalaca con la ruta de la imagen;
    Fin del Si

    Devolver fruta;
Fin del Método

Método actualizar()
    Si fruta.frutaActiva es falso
      frutasCaidas.agregar(fruta);

      fruta = siguienteFruta;
      fruta.establecerPosicion(posInicialX, posInicialY);
      siguienteFruta = elegirFruta();
      siguienteFruta.establecerPosicion(posInitSiguienteX, posInitSiguienteY);
    Sino
      fruta.actualizar();
    Fin del Si
Fin del Método

Método dibujar(graficos2)
```

```
    // Dibujar contenedor
    graficos2.setColor(Color.yellow);
    graficos2.setStroke(new BasicStroke(4f));
    graficos2.drawRect(xIzquierda - 4, yArriba + 40, ANCHO + 8, ALTO + 8);

    // Dibujar cuadrado para mostrar la siguiente fruta a caer
    Declarar x = xIzquierda - 220;
    Declarar y = yArriba + 40;
    graficos2.drawOval(x, y, 100, 100);

    // Se dibuja la fruta
    Si fruta no es null
      fruta.dibujarFruta(graficos2);
    Fin del Si

    // Se dibuja la siguiente fruta
    siguienteFruta.dibujarFruta(graficos2);

    //frutasCaidas.dibujarFrutas(graficos2);
  Fin del Método
Fin de la Clase
```

Clase Fruta

```
// Atributos

Variables posX, posY;

Variable estática diametro = 30;

Declarar imagen como BufferedImage;

Variable contadorCaida = 0;

Variables booleanas colisionIzquierda, colisionDerecha, colisionFondo, frutaActiva;


Constructor Fruta(rutaImagen)
  Intentar
    Leer la imagen desde la rutaImagen y asignarla a this.imagen;
  Capturar
    Imprimir la traza de la excepción;
  Fin del Intentar
Fin del Constructor


Método establecerPosicion(posX, posY)
  this.posX = posX;
  this.posY = posY;
Fin del Método


// Método para actualizar la posición y movimiento de la fruta

Método actualizar()
```

```
Incrementar contadorCaida;

Llamar al método comprobarColision();

// Mover la fruta
Si Controles.botonIzquierda es verdadero
   Si colisionIzquierda es falso
      Restar diametro a posX;
   Fin del Si
   Asignar falso a Controles.botonIzquierda;
Fin del Si
Si Controles.botonDerecha es verdadero
   Si colisionDerecha es falso
      Sumar diametro a posX;
   Fin del Si
   Asignar falso a Controles.botonDerecha;
Fin del Si

Si colisionFondo es verdadero
   Asignar falso a frutaActiva;
Sino
   Si contadorCaida es igual a ManagerJuego.caidaFruta
```

```
        Si colisionFondo es falso
            Sumar diametro a posY;
            Asignar 0 a contadorCaida;
        Fin del Si
    Fin del Si
Fin del Si
Fin del Método

Método comprobarColision()
    // Se establecen las variables de colisión en falso
    Asignar falso a colisionDerecha, colisionIzquierda, colisionFondo;

    // Se verifican las colisiones con el contenedor

    // Lado Izquierdo
    Si posX es menor que ManagerJuego.xIzquierda
        Asignar verdadero a colisionIzquierda;
    Fin del Si
    // Lado Derecho
    Si posX más diametro es mayor que ManagerJuego.xDerecha
        Asignar verdadero a colisionDerecha;
    Fin del Si
```

// Fondo

Si posY es mayor que ManagerJuego.yAbajo

Asignar verdadero a colisionFondo;

Fin del Si

Fin del Método

Método dibujarFruta(graficos2)

// Creamos una nueva imagen que será un círculo perfecto del tamaño correcto

Declarar imagenCircular como un nuevo BufferedImage con diametro, diametro, BufferedImage.TYPE_INT_ARGB;

Declarar g como los gráficos de imagenCircular;

// Dibujamos la imagen de la fruta dentro del círculo

g.setClip(new Ellipse2D.Float(0, 0, diametro, diametro));

g.drawImage(this.imagen, 0, 0, diametro, diametro, null);

g.dispose();

// Dibujamos la imagen circular en el círculo

graficos2.drawImage(imagenCircular, posX, posY, null);

Fin del Método

Fin de la Clase

Clase Datil extiende Fruta

Constructor Datil(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Fin del Constructor

Fin de la Clase

Clase Pumalaca extiende Fruta

Constructor Pumalaca(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 1.8;

Fin del Constructor

Fin de la Clase

Clase Cotoperi extiende Fruta

Constructor Cotoperi(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 1.2;

Fin del Constructor

Fin de la Clase

Clase Cereza extiende Fruta

Constructor Cereza(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 1.6;

Fin del Constructor

Fin de la Clase

Clase Coco extiende Fruta

Constructor Coco(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 3.2;

Fin del Constructor

Fin de la Clase

Clase Kiwi extiende Fruta

Constructor Kiwi(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 2.2;

Fin del Constructor

Fin de la Clase

Clase Mango extiende Fruta

Constructor Mango(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 2.8;

Fin del Constructor

Fin de la Clase

Clase Parchita extiende Fruta

Constructor Parchita(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 2.6;

Fin del Constructor

Fin de la Clase

Clase Patilla extiende Fruta

Constructor Patilla(rutaImagen)

Llamar al constructor de la superclase con rutaImagen;

Multiplicar diametro por 4;

Fin del Constructor

Fin de la Clase

Clase Nodo

// Atributos

Declarar valor como Fruta;

Declarar siguiente como Nodo;

// Método Constructor

Constructor Nodo(valor)

this.valor = valor;

this.siguiente = null;

Fin del Constructor

// Métodos Getters y Setters

Método getValor()

Devolver valor;

Fin del Método

Método setValor(valor)

this.valor = valor;

Fin del Método

Método getSiguiente()

Devolver siguiente;

Fin del Método

Método setSiguiente(siguiente)

this.siguiente = siguiente;

Fin del Método

Fin de la Clase

Clase Lista

// Atributos

Declarar cabeza como Nodo;

Declarar tamano como int;

// Método Constructor

Constructor Lista()

Asignar null a cabeza;

```
    Asignar 0 a tamano;
Fin del Constructor

// Método para verificar si la lista está vacía
Método estaVacia()
    Devolver si cabeza es igual a null;
Fin del Método

// Método para agregar una fruta a la lista
Método agregar(valor)
    // Se crea un nuevo nodo
    Declarar nuevo como un nuevo Nodo con valor;

    Si la lista está vacía
        Asignar nuevo a cabeza;
    Sino
        Declarar temp como cabeza; // Se crea un nodo temporal

        Mientras el siguiente de temp no sea null
            Asignar el siguiente de temp a temp; // Se sigue avanzando al siguiente nodo
        Fin del Mientras
```

```
        Asignar nuevo al siguiente de temp; // Se agrega el nuevo nodo con su valor al final de la lista
      Fin del Si
      Incrementar tamano; // Se incrementa el tamano de la lista
    Fin del Método

    // Método para obtener la cabeza de la lista
    Método getCabeza()
      Devolver cabeza;
    Fin del Método

    // Método para eliminar una fruta de la lista
    Método eliminar(valor)
      Si la lista está vacía
        Retornar;
      Fin del Si

      Si el valor de cabeza es igual a valor
        Asignar el siguiente de cabeza a cabeza;
        Decrementar tamano;
        Retornar;
      Fin del Si
```

Declarar actual como cabeza;

Mientras el valor del siguiente de actual es igual a valor

Asignar el siguiente del siguiente de actual al siguiente de actual;

Decrementar tamano;

Retornar;

Fin del Mientras

Asignar el siguiente de actual a actual;

Fin del Método

Fin de la Clase

Clase Controles implementa KeyListener

Variables estáticas booleanas botonIzquierda, botonDerecha;

Método keyTyped(e) {}

Método keyPressed(e)

Declarar codigo como el código de la tecla presionada en e;

Si codigo es igual a KeyEvent.VK_LEFT

Asignar verdadero a botonIzquierda;

Fin del Si

Si codigo es igual a KeyEvent.VK_RIGHT

Asignar verdadero a botonDerecha;

Fin del Si

Fin del Método

Método keyReleased(e) {}

Fin de la Clase