UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Artes Visuales & Multimedia
Máster Oficial· UPV

MÁSTER UNIVERSITARIO EN ARTES VISUALES Y MULTIMEDIA

TRABAJO FINAL DE MÁSTER. Anexos.

Interfaces, diversidad funcional e interacción lúdica.

Caso : Tetris Escape. Prototipo de juego de control por sonido sin reconocimiento del habla

Trabajo presentado por:
Dña. Ainhoa Salas Richarte

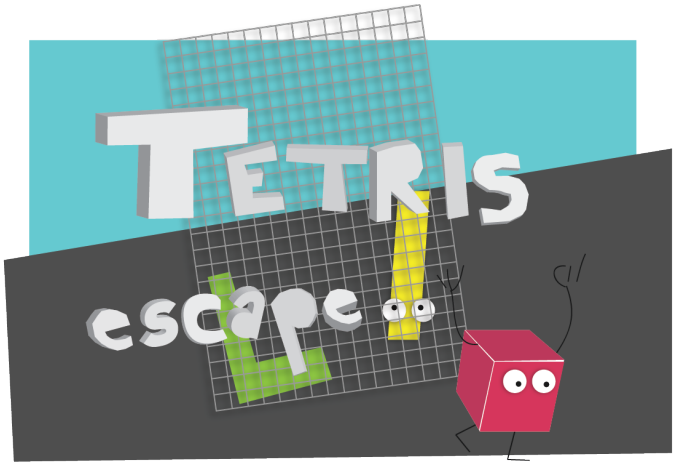
Dirigido por:
Dr. Moisés Mañas Carbonell
Dr. Carlos Manuel García Miragall

VALENCIA, septiembre de 2016

Índice de contenidos

Anexos

1. Bocetado, planteamiento y análisis funcional del programa



Estado 0

pantalla de carga.

- el sistema se está inicializando.

- se pasa al Estado 1 de forma automática, sin input del usuario.

Caso de uso

- Al usuario se le presenta una imagen con el tema del juego. No se espera nada de este.

Estado 1

pantalla calibración.

- el sistema espera la entrada de sonido en un lapso concreto de tiempo

- sino recibe input sonoro en el primer lapso y repite la operación dos veces más

- si no recibe input sonoro en ninguno de los tres intervalos define un valor de input por defecto

- si recibe una entrada en alguno de los tres periodos pasa a procesar ese input y no repite el lapso para la recepción

- los datos introducidos llenan un búfer de un tamaño concreto, el sistema hace una media con los datos de ese búfer. Este dato “media” se almacena en una variable global

- se pasa al Estado 2 cuando obtenemos un dato para esa variable, ya se el procesado a partir del input o el que definimos por defecto

- de esta fase tenemos que obtener una clase calibración

Caso de uso

- Al usuario se le presentan unas instrucciones. Debe producir un sonido con un volumen que le resulte natural, no muy alto ni muy bajo.

- debe hacerlo en un periodo concreto de tiempo que se marca con un barra de carga. La barra es una visualización del periodo de tiempo en que el sistema está “escuchando” al usuario

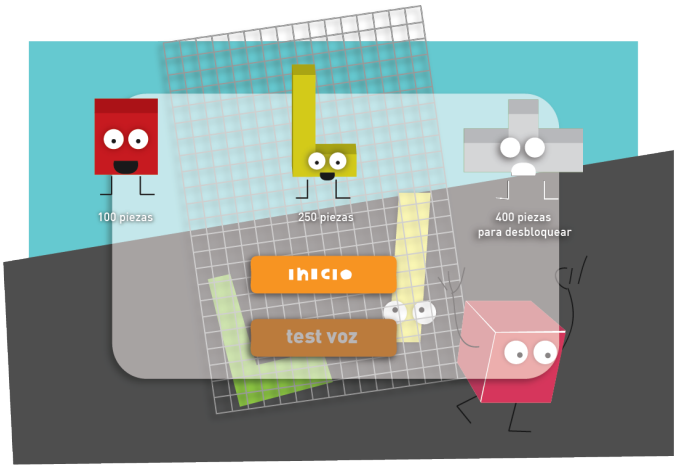
Requisitos funcionales

- función que habilite por un periodo concreto de tiempo la entrada de sonido

- crear búfer que pueda almacenar todos los datos que pueden entrar durante el periodo de recepción de datos

- comprobar si el búfer se ha llenado. Si no se ha llenado, repetir la función que habilita la entrada de sonido; si se ha llenado pasar a función que calcula la media de los datos obtenidos. De estos habría que limpiar a lo mejor los valores que sean muy bajos, ya que pueden ser ruido ambiente

- si se ha obtenido el valor medio del búfer de datos sonoros, almacenarlo en una variable global y pasar al siguiente estado



Estado 2

pantalla inicio del juego.

- se presenta al usuario una pantalla con un menú de opciones. Una de las opciones está resaltada.

- el sistema está constantemente esperando un input sonoro

- si recibe un input lo suficientemente largo como para llenar un búfer de X longitud, se cambia la opción preseleccionada, es decir, en la interfaz gráfica se resalta la siguiente opción

- si recibe un input corto, que no llene un búfer, la opción resaltada se selecciona

- esa selección es lo que produce el cambio de estado. En este caso, depende de la selección, se puede volver al Estado 1 o pasar al Estado 3

Caso de uso

- Al usuario se le presenta un menú de opciones. En los casos en los que haya menú de opciones la interacción se basa en sonidos largos para cambiar la opción resaltada y sonido corto para confirmar la selección

- dos posibles opciones: iniciar el juego o volver a la calibración por si no ha podido calibrarla o por si la calibración que ha hecho no le convence o no se le ha ajustado bien

Requisitos funcionales

- función selección en menú de opciones.

    - el sistema está a la escucha de forma continua. Si los datos entrantes no superan un umbral establecido como ruido ambiente los datos no son almacenados.

    - si hay una entrada de sonido con un volumen más o menos similar al almacenado como “media” en el estado anterior se almacena en un búfer

    - la longitud de este búfer se compara con parámetros preestablecidos (definidos por el diseñador) de longitud y se cataloga según pertenezca a un grupo o a otro como “sonido largo” o “sonido corto”

- función de selección por casos

    - se alimenta de las variables definidas en la función anterior

    - tiene una serie de funciones que se corresponden con las opciones que puede elegir el jugador. Sería “boton1” y “boton2”, por defecto, cuando se le diese la orden, el sistema elegiría “boton1”

    - si entra “sonido largo” (en la interfaz gráfica se resaltaría el botón siguiente, pero no sé cómo se traduciría esto a lo que estaría ocurriendo a nivel interno) la opción por defecto que es “boton1” cambia y pasa a ser “boton2”

    - si entra “sonido corto” (es una confirmación de la selección que se cambia con “sonido largo” pero no sé cómo se podría dejar preseleccionada una opción a nivel interno) el sistema ejecuta “boton1” o “boton2” dependiendo de cuál sea la opción que tiene el sistema en ese momento por defecto

- función “boton1” produce un cambio de estado, en este caso deriva al Estado 3

- función “boton2” produce también un cambio de estado, deriva al Estado 1

Estado 3

dentro del juego.

- en este estado se van presentando aleatoriamente diferentes pantallas al usuario de forma continua

- el sistema está a la espera de input constantemente

- el tipo de input esperado en cada una de las pantallas o mini juegos se define en cada mini juego, pero siempre tendrán que ver con el volumen del sonido, por lo tanto, los inputs recibidos en las pruebas se compararán con el valor que hayamos obtenido en el Estado 1

- Hay dos formas de cambiar de estado: llegando a “game over”, esto ocurre cuando perdemos las pruebas; de esta forma se pasa al Estado 5; o podemos pasar al Estado 4 que sería la pausa, en cualquier momento, del Estado 3. Esto ocurre cuando el sistema recibe un input largo en cualquier momento del Estado 3

Caso de uso

- Al usuario se le presentan pruebas de forma aleatoria dependiendo del nivel en el que se encuentre. Si las pasa se siguen presentando pruebas, si las pierde va perdiendo vidas, si pierde tres vidas es “game over”

- Las interacciones requeridas dependen de cada mini juego, pero siempre tienen que ver con el volumen, más alto o menos que el que el mismo usuario ha establecido como “normal” o medio al inicio de la aplicación

Requisitos funcionales

- existe un array con una serie de variables que aluden a mini juegos, mini estados o mini funciones

- función que elige una variable aleatoria de ese array. Hay un requisito que dice que no se puede escoger la misma variable dos veces seguidas, por lo tanto, la variable escogida se almacena para que, al hacer la selección nueva, el sistema las compare, si son diferentes, se lanza la función escogida, y la variable se almacena en el puesto de la anterior; si son iguales se vuelve a realizar el proceso de selección.

- cada uno de esos mini juegos tiene sus propias funciones. Se ejecutan y al acaba devuelven al sistema una variable “gana” o “pierde”

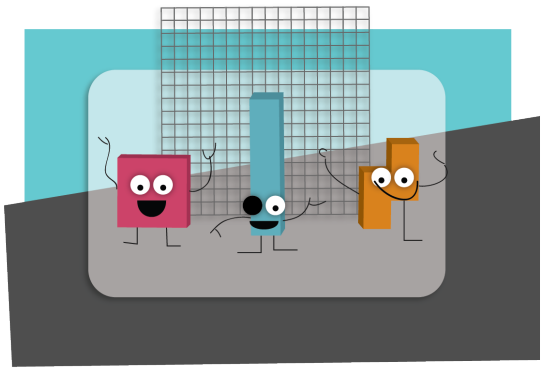
    - si el juego devuelve “gana” se suma una “estrella” o “pieza” al recuento general de estrellas o piezas y se lanza la “pantalla de recuento n”

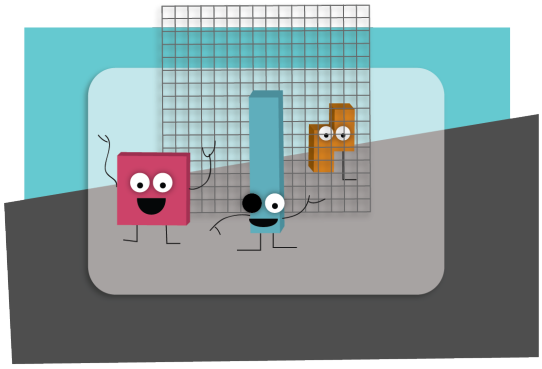
    - si el juego devuelve “pierde” no se suman estrellas, se resta una unidad a la variable vidas. A continuación se comprueba la variable vidas, si es igual a 3, se presenta la “pantalla de recuento 3”, si es igual a 2, se presenta la “pantalla de recuento 2”, si es igual a 1, se presenta la “pantalla de recuento 1”, si es igual a 0 se pasa al Estado 5 “game over”

    - después de lanzar la “pantalla de recuento n” se espera n segundos y se vuelve a la función que elige la siguiente prueba

- (habría que hacer una variable que sea un contador de partidas jugadas o de juegos jugados para que se fuera aumentando la dificultad de las partidas si el jugador juega muchas partidas sin perder las vidas)

- lo ideal para que el jugador entendiese perfectamente que ha perdido una vida sería hacer una animación de la ficha yendo a la cárcel, pero será para trabajo futuro





*

*pantallas de recuento

Estado 4

pausa

- a este estado solo se puede acceder desde el Estado 3

- pone en pausa el Estado 3 y presenta un menú de opciones al usuario. La dinámica es la misma que en el resto de menús de opciones, la interacción se produce a través de inputs cortos o largos

- los estados a los que podemos llegar desde este son el Estado 1, el Estado 2 o continuar en el Estado 3

Caso de uso

- Al usuario se le presenta un menú de opciones. Puede continuar el juego, volver a la pantalla de inicio o a la pantalla de calibración

Requisitos funcionales

- durante todo el Estado 3 se sigue comprobando la longitud de los sonidos, es decir, cómo de largos son los búfers. Si en algún momento se crea uno muy largo se pasa a este estado

- ¿¿cómo se pone en pausa un estado?? cómo se mantiene lo que esté sucediendo para poder continuar como lo habíamos dejado

- en este estado se presenta de nuevo un menú de opciones, por lo tanto utilizaremos la función selección en menú de opciones.

- “boton1” reanuda el Estado 3 por donde lo habíamos dejado

- “boton2” nos lleva al Estado 1 (pantalla de calibración)

- “boton3” nos lleva al Estado 2 (pantalla inicio del juego)

Estado 5

game over.

- a este estado solo se puede acceder desde el Estado 3

- se presenta al usuario un menú de opciones. Se sigue la dinámica de menú de opciones para la interacción

- desde aquí podemos acceder al Estado 2 o reiniciar el Estado 3

Caso de uso

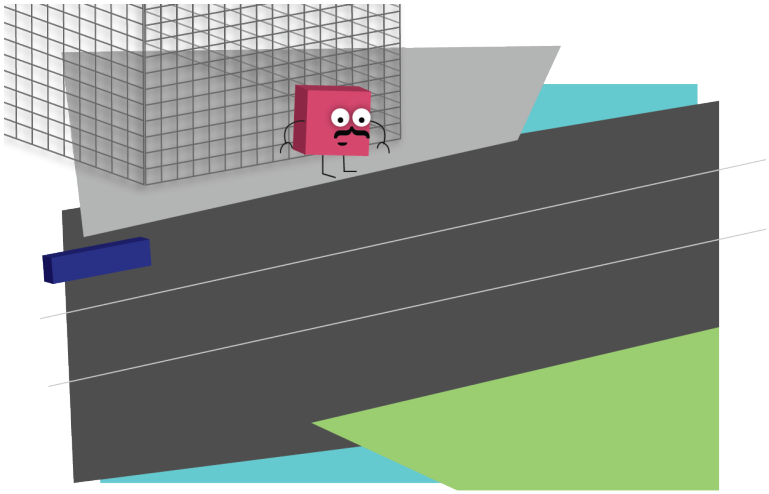
- Game over, el usuario ha perdido todas las vidas. Puede volver a iniciar el juego directamente desde aquí o volver a la pantalla de inicio

Requisitos funcionales

- en este estado se presenta de nuevo un menú de opciones, por lo tanto utilizaremos la función selección en menú de opciones

- “boton1” nos lleva al Estado 2 (pantalla inicio del juego)

- “boton2” nos lleva al Estado 1 (pantalla de calibración)



Mini juego 1

frogger.

- a este estado solo se puede acceder desde el Estado 3

- el sistema espera sonidos cortos y concretos, el volumen similar el de “media”.

- con cada entrada de sonido, sea larga o corta, el objeto mueve una posición hacia delante

- si llega al otro lado de la pantalla gana la prueba

- si colisiona con algun otro objeto pierde la prueba o si se agota el tiempo sin haber llegado a la meta, pierde la prueba

Caso de uso

- el usuario interactua con un sonido “medio” para mover al personaje, para mover al personaje otra vez debe interrumpir la emisión de sonido y volver a emitir, con una emisión continua, el personaje, solo se mueve una vez

Requisitos funcionales

- la función mover personaje frogger.

    - requiere de un input sonoro con un volumen similar al de “media” para actuar

- el movimiento del objeto se produce una sola vez al recibir el input sonoro, aunque siga habiendo sonido, si no ha habido un momento de silencio, el objeto no se mueve otra vez

- función mover enemigos

- objetos que avanzan en una dirección concreta y a una velocidad continua de un lado a otro del escenario. Cambian su dirección en X y ligeramente en Y.

- habrá que crear variable contador y clase objeto y que cada x tiempo se cree un objeto y luego haga el recorrido concreto de ese objeto

- todos los objetos tienen colisión, si colisionan con el personaje se acaba la prueba, el usuario pierde

- cuenta atrás

- unas milesimas después de iniciarse la prueba se inicia la cuenta atrás (se dejan unas milésimas para qeu el jugador pueda leer las instruccuiones)

- si se agota el tiempo y el personaje no ha llegado a la meta definida se acaba la prueba, el usuario pierde

- si el personaje llega a la meta definida el usuario gana la prueba,a nivel local de este mini estado se envía la variable “gana frogger” y así se reprocuce la animación “gana frogger animacion” y para el nivel global se devuelve la variable “gana”, el Estado 3 se encarga de realizar las funcions relacionadas con “gana”; si se pierde la prueba, a nivel local se envía la variable “pierdefrogger” y así se reprocuce la animación “pierdefrogger animacion” y para el nivel global se devuelve se devuelve la variable “pierde”, de nuevo, el Estado 3 se encarga de realizar las funcions relacionadas con “pierde”

2. URL para visualización del gameplay

Se puede ver un gameplay en la siguiente dirección https://youtu.be/WnyUSI83ee4

3. Enlace para descarga del juego

Mac OSx : https://www.dropbox.com/s/tdnb0tbnw6bax8x/application.macosx.zip?dl=0

Windows 32 bits:
https://www.dropbox.com/s/eps9jnfz1x7r9uu/application.windows32.zip?dl=0

Windows 64 bits:
https://www.dropbox.com/s/f33t6fk06h53hcz/application.windows64.zip?dl=0

4. Código del programa Tetris Escape

```
// Clase principal
import ddf.minim.spi.*;
import ddf.minim.signals.*;
import ddf.minim.*;
import ddf.minim.analysis.*;
import ddf.minim.ugens.*;
import ddf.minim.effects.*;

InterpreteII interprete2;
InterpreteSonido interprete;
Estados estado;
Cronometro c;

Minim minim;

void setup()
{
  size(800, 600, P3D);
  frameRate(120);

  minim = new Minim(this);
```

```
  interprete2 = new InterpreteII();
  interprete = new InterpreteSonido(minim);

  estado = new Estados();

  c = new Cronometro();

}

void draw()
{
  background (255);

  interprete2.detectaSonido();
  interprete.detectaSonido();

  estado.cambiarEstado();
  estado.dibujarEstado();

}

// Clase cronómetro
class Cronometro
{
    //--------Atributos-----------//
  int tiempoInicio;
  boolean enMarcha;

    //--------constructor-----------//
  Cronometro ()
  {
    enMarcha = false;
  }

    //--------métodos-----------//
  void arrancarCrono()
  {
    enMarcha = true;
    tiempoInicio = millis();
  }

  void pararCrono()
  {
    enMarcha = false;
  }

  int tiempoTrasncurrido()
  {
      int tiempoActual = 0, tiempoTranscurrido = 0;
      if (enMarcha)
        {  tiempoActual = millis();
           tiempoTranscurrido = (tiempoActual - tiempoInicio)/1;
```

```
      }
      return tiempoTranscurrido;
  }}

// Clase estados
class Estados
{
   //--------Atributos----------//
   int est = 0;
   PImage e0, e1, e2, e3, e4, e5;
   int boton;
   Juego1 juego;

   //--------constructor----------//
   Estados()
   {
    est = 0;
    boton = 1;

    e0 = loadImage("e-0.jpg"); //pantalla intro
    e1 = loadImage("e-1.jpg"); //pantalla calibracion
    e2 = loadImage("e-2.jpg"); //pantalla inicio
    e3 = loadImage("e-3.jpg"); //pantalla aux juego
    e4 = loadImage("e-4-transparente-01.png"); //pantalla pausa
    e5 = loadImage("e-5.jpg"); //pantalla game over
   }

   //--------métodos----------//
   /* cambios estado */
   void cambiarE1()
   {
    est = 1;
    c.arrancarCrono();
   }

   void cambiarE2()
   {
    est = 2;
   }

   void cambiarE3()
   {
    est = 3;
    juego = new Juego1();
   }

   void cambiarE4()
   { // estado de pausa
    est = 4;
   }

   void cambiarE5()
```

```
{ // game over
  est = 5;
}

void cambiarEstado()
{
// para controlar con teclado
//int tipo = interprete2.getTipoSonido();
//int duracion = interprete2.getLongitudSonido();

//control con sonido, recoge la variable que se genera en la clase interprete
//int tipo = interprete.getTipoSonido();
int duracion = interprete.getLongitudSonido();

  if (est == 0)
  {
    int m = millis();
    if (m >= 3000){
      cambiarE1();
      //c.arrancarCrono();
    }

  }
  if (est == 1)
  {
    int t = c.tiempoTrasncurrido();
    println(t);
    if(t >= 4000)
   {
     cambiarE2();
   }

  }
  if (est == 2)
  {
    if(boton == 1 && duracion == 1)
    {
      cambiarE3();
    } else if (boton == 2 && duracion == 1)
    {
      cambiarE1();
    }else if (duracion == 2 && boton == 1)
      {
         boton = 2;
      } else if (duracion == 2 && boton == 2)
        {
          boton = 1;
        }

    println ("duracion = "+duracion);

    }
```

```
    if (est == 3)
    {
      if (duracion == 2)
      {
        cambiarE2();
      }
      if (juego.esFinJuego())
      {
        cambiarE5();
      }
    }

}

/* acciones en estado*/
void dibujarEstado()
{
    //println(est);
  if (est == 0)
  {
    image(e0, 0, 0);
    e0.resize (800,0);
  }
  if (est == 1)
  {
    image(e1, 0, 0);
    e1.resize (800,0);

    float t = c.tiempoTrasncurrido();
    float m = map(t, 0, 4000, 0, 250);
    // dibujar la barra de tiempo
    noStroke();
    fill (90, 90, 90);
    rect (270, 305, 250, 40);

    fill (58, 250, 158);
    rect (270, 305, m, 40);

    //habría que recoger aquí las cosas para almacenar variable media de sonido
  }
  if (est == 2)
  {
    image(e2, 0, 0);
    e2.resize (800,0);

    //coloca recuadro remarcar botón
    if (boton == 1)
    {
      stroke(255, 133, 18);
      strokeWeight(4);
      noFill();
      rect(width/2 -120, 325, 200, 50);
```

```
      } else if (boton == 2)
      {
        stroke(255, 133, 18);
        strokeWeight(4);
        noFill();
        rect(width/2 -120, 401, 200, 50);
      }
    }
    if (est == 3)
    { //se dibuja el juego y se va actualizando la posición de los personajes y demás
      juego.actualizar();
      juego.dibujar();
    }
    if (est == 4)
    {
      image(e4, 0, 0);
      e4.resize (800,0);
    }
    if (est == 5)
    {
      image(e5, 0, 0);
      e5.resize (800,0);
    }
  }
}

// Clase Interprete_sonido
class InterpreteSonido
{
 //---Atributos---//
 Minim m_minim;
 AudioInput in;

 float volumen;
 int tipoSonido, numeroBuffer, ultimoBuffer, longSonido;

 //---constructor---//
 InterpreteSonido(Minim minim)
 {
  m_minim = minim;
  in = m_minim.getLineIn();

  numeroBuffer = 0;
  ultimoBuffer = 0;
 }

 //---métodos---//

 // estos metodos son para capturar y enviarlas variables del tipo o duracion del sonido
 int getTipoSonido()
 {
  return tipoSonido;
```

```
}

int getLongitudSonido()
{
 return longSonido;
}

// este método sería para la calibración del estado 1
float getVolumen()
{
 return volumen;
} // no sé si enviar la variable volumen o la variable m que ya está mapeada

// este método es para diferenciar los tipos y la duracion del sonido
void detectaSonido()
{
 //el mix.level() mide los niveles de audio y hace una RMS (media cuadrática) del búfer
 volumen = in.mix.level();
 float m = map(volumen, 0, 1, 1, 10);

 /*para ver si el sonido es alto bajo o silencio.
 esto tiene que ver con las variables tipoSonido y son:
 =0 si hay silencio, =1 si es bajo, =2 si es alto.
 en una versión en la que incluyesemos la calibración del volumen,
 la cifra con la que se compara la variable m debería ser una variable que
 se estableciese en relación con la información que obtuviesemos en la pantalla calibración*/
 if (m >= 2.0 && m <= 3.0)
 { // sonido bajo
  tipoSonido = 1;
  delay (600);
  m = 0;
  delay(400);
 } else if (m >= 3.0 && m <= 10.0)
 { // sonido alto
  tipoSonido = 2;
  delay(600);
  m = 0;
  delay(400);
 } else
 { // silencio
  tipoSonido = 0;
 }

 /* Para saber si el sonido es corto o largo.
 Tomamos como referencia las variables tipoSonido.
 Mientras el sonido sea diferente a silencio, hay una variable que va aumentando (numeroBuffer).
 Cuando hay un silencio esta variable se almacena en ultimoBuffer y se vuelve a poner a 0
 despuén se compara este dato que hemos almacenado y se determina que si es menor que 5, el último
 sonido antes del silencio ha sido corto, si es mayor que 5 ha sido largo */
 if (tipoSonido == 1 | tipoSonido == 2)
 {
  numeroBuffer ++;
```

```
    } else if (tipoSonido == 0)
    {
      ultimoBuffer = numeroBuffer;
      numeroBuffer = 0;
    }
    if (ultimoBuffer >=1 && ultimoBuffer <= 3)
    { // sonido corto
      longSonido = 1;
    } else if (ultimoBuffer >= 3)
    { // sonido largo
      longSonido = 2;
    } else
    {
      longSonido = 0;
    }
  }
}

// Clase Interprete_II
class InterpreteII
{
  //--------Atributos-----------//
  float volumen;
  int tipoSonido, longSonido;

  //--------Constructor-----------//
  InterpreteII()
  {
    tipoSonido = 0;
    longSonido = 0;
    volumen = 3.0;
  }

  //--------Métodos-----------//
  int getTipoSonido() {
   return tipoSonido;
  }

 int getLongitudSonido(){
   return longSonido;
 }

 void detectaSonido()

 {
   if (keyPressed)
   {
       if (key == 'a' || key == 'A')
       {
         // corto y bajo
         longSonido = 1;
         tipoSonido = 1;
```

```
}

if (key == 'b' || key == 'B')
{
  //largo y bajo
  longSonido = 2;
  tipoSonido = 1;
}

if (key == 'c' || key == 'C')
{
  //corto y alto
  longSonido = 1;
  tipoSonido = 2;
}
if (key == 'd' || key == 'D')
{
  //largo y alto
  longSonido = 2;
  tipoSonido = 2;
}
if (key == 'e' || key == 'E')
{
  //silencio
  longSonido = 0;
  tipoSonido = 0;
}
 if (key == 'f' || key == 'F')
{
  //larguísimo bajo
  longSonido = 3;
  tipoSonido = 1;
  volumen = 3.0;
}
 if (key == 'g' || key == 'G')
{
  //larguísimo alto
  longSonido = 3;
  tipoSonido = 2;
  volumen = 9.0;
}
if (key == 'h' || key == 'H')
{
  //tiempo excedido
  longSonido = 4;
  tipoSonido = 1;
}
if (key == 'i' || key == 'I')
{
  //tiempo excedido
  longSonido = 4;
```

```
        tipoSonido = 2;
       }
   }else
   {
    longSonido = 0;
    tipoSonido = 0;
   }
 }
}

// Clase Juego1
class Juego1
{
  // ---- atributos ---- //
  PImage fondo;
  Protagonista protagonista;
  Villano v1;
  Villano2 v2;
  Villano3 v3;
  float colision1, colision2, colision3;
  boolean finJuego;

  // ---- constructor ---- //
  Juego1()
  {
   fondo = loadImage("e-3-vacio.jpg");
   protagonista = new Protagonista();
   finJuego = false;

   v1 = new Villano();
   v2 = new Villano2();
   v3 = new Villano3();

  }

  // ---- métodos ---- //

  void actualizar()
  {
   colisiones();
   protagonista.personajeYposicion();


  }
  void dibujar()
  {
   image(fondo, 0, 0);
   fondo.resize(800,0);
   protagonista.dibujar();

   v1.dibujarVillano();
   v2.dibujarVillano2();
```

```
   v3.dibujarVillano3();

   protagonista.mover();
  }

  boolean esFinJuego()
  { return finJuego;
  }

  void colisiones()
  {
   // --- cogemos la posición en X y en Y del personaje protagonista --- //
   int protaX = protagonista.getPosicionX();
   int protaY = protagonista.getPosicionY();

   // --- y el valor de la variable posicion -- //
   int posicion = protagonista.getPosicion();

   // --- cogemos la posición en X y en Y de los antagonistas --- //
   int vill1X = v1.getPosicionV1x();
   float vill1Y = v1.getPosicionV1y();

   int vill2X = v2.getPosicionV2x();
   float vill2Y = v2.getPosicionV2y();

   int vill3X = v3.getPosicionV3x();
   float vill3Y = v3.getPosicionV3y();

   //------ calcula la distancia entre el protagonista y el villano ------//
   colision1 = dist(protaX, protaY, vill1X, vill1Y);
   colision2 = dist(protaX, protaY, vill2X, vill2Y);
   colision3 = dist(protaX, protaY, vill3X, vill3Y);

    // ------ colisiones ------ //
    if ((posicion == 2 && colision1 <= 130) || (posicion == 3 && colision2 <= 130) || (posicion == 4 &&
colision3 <= 130))
    {
     finJuego = true;
    }
  }
}

// Clase Protagonista
class Protagonista
{
  // ---- atributos ---- //
  PImage personaje;
  int posicion, posX, posY;

  // ---- constructor ---- //
  Protagonista()
  {
```

```
  personaje = loadImage("prota.png");

  posicion = 0;
}

// ---- métodos ---- //
void personajeYposicion()
{
  // --- determina las posiciones del personaje --- //
  if (posicion == 1)
  {
    posX = 170;
    posY = 70;
  }else if (posicion == 2)
  {
    posX = 230;
    posY = 130;
  }else if (posicion == 3)
  {
    posX = 300;
    posY = 200;
  }else if (posicion == 4)
  {
    posX = 380;
    posY = 280;
  }else if (posicion == 5)
  {
    posX = 470;
    posY = 370;
  }
}
void dibujar()
{
  // --- dibuja al personaje en alguna de las posiciones anteriores --- //
  image(personaje, posX, posY);
  personaje.resize (150, 0);
}

// -- envíaposición en X y en Y --//
int getPosicionX()
{
  return posX;
}

int getPosicionY()
{
  return posY;
}

// -- envía el valor de la variable posicion -- //
int getPosicion()
{
```

```
  return posicion;
 }

 int mover()
 {
  //int tipo = interprete2.getTipoSonido();
  //int duracion = interprete2.getLongitudSonido();

  int duracion = interprete.getLongitudSonido();

  if (duracion == 1)
  {
   posicion += 1;
  }
  return posicion;//creo que no hace falta
 }
}

//Clase Villano1

class Villano
{
 // ---- atributos ---- //
 PImage villano;
 int posV1x;
 float posV1y;

 // ---- constructor ---- //
 Villano()
 {

  villano = loadImage("villano.png");

  posV1x = width;
  posV1y = 90;
 }
 // ---- métodos ---- //

 void dibujarVillano()
 {
  image(villano, posV1x, posV1y);
  villano.resize(150, 0);

  if (posV1x >= -villano.width)
  {
   posV1x -= 4;
   posV1y += 0.8;
  }else
  {
   posV1x = width;
   posV1y = 90;
  }
```

```
  }

  int getPosicionV1x()
  {
    return posV1x;
  }

  float getPosicionV1y()
  {
    return posV1y;
  }
}

// Clase Villano2

class Villano2
{
  // --- atributos --- //
  PImage villano2;
  int posV2x;
  float posV2y;

  // --- constructor --- //
  Villano2()
  {
    villano2 = loadImage("villano2.png");

    posV2x = -villano2.width;
    posV2y = 370;
  }

  // --- métodos --- //
  void dibujarVillano2()
  {
    image(villano2, posV2x, posV2y);
    villano2.resize(150,0);

    if(posV2x <= width)
    {
      posV2x += 4;
      posV2y -= 0.8;
    }else
    {
      posV2x = -villano2.width;
      posV2y = 370;
    }
  }

  int getPosicionV2x()
  {
    return posV2x;
  }
```

```
  float getPosicionV2y()
  {
    return posV2y;
  }
}

// Clase Villano3
class Villano3
{
  // ---- atributos ---- //
  PImage villano3;
  int posV3x;
  float posV3y;

  // ---- constructor ---- //
  Villano3()
  {
    villano3 = loadImage("villano3.png");

    posV3x = width;
    posV3y = 290;
  }

  // ---- métodos ---- //
  void dibujarVillano3()
  {
    image(villano3, posV3x, posV3y);
    villano3.resize(150,0);

    if(posV3x >= -villano3.width)
    {
      posV3x -= 3;
      posV3y += 0.6;
    }else
    {
      posV3x = width;
      posV3y = 290;
    }
  }

  int getPosicionV3x()
  {
    return posV3x;
  }

  float getPosicionV3y()
  {
    return posV3y;
  }
}
```