Juegos inmersivos mediante color tracking

Curso 2018/2019

Markéta Kučerová (marku7@inf.upv.es)
Antonio Puchalt Alós (anpucal@inf.upv.es)

Sistemas Multimedia Interactivos e Inmersivos
Grado de Ingeniería informática
Universitat Politècnica de València

Abril 2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

Índice de figuras

Abstract

El objetivo de este trabajo ha sido diseñar y desarrollar tres simples juegos inmersivos utilizando Processing. Los juegos se juegan a través de una realidad aumentada simple que viene dada por la webcam y controlado con un mecanismo de rastreo de color. Empiezan en un menú común donde el jugador elige el juego al que jugar. La aplicación se controla completamente a mediante dos objetos de color bien diferenciados que son rastreados continuamente.

The objective of this work was to design and develop three simple immersive games using Processing. The games are played in webcam augmented reality and controlled by color-tracking mechanism. They start at one common menu where the player can choose a game to play. The application is fully controlled by two tracked colorful objects and allows upto two player to participate.

1. Introducción y objetivos

La idea principal fue realizar una realidad aumentada de manera que fuese sencilla, accesible y nuestra. Con un par de herramientas básicas que están al alcance de todos, como una cámara web y un objeto de color bien definido, hemos sentado el punto de partida para empezar a desarrollar nuestra aplicación. Para poder comunicarnos con la aplicación utilizaremos el color tracking.

Nuestro primer objetivo fue hacer una aplicación divertida y agradable, por tanto decidimos desarrollar juegos. Pensamos dos opciones: hacer un juego grande y algo más complejo, o hacer una serie de pequeños juegos. Teniendo así la posibilidad de probar más conceptos y explorar varias herramientas, elegimos hacer tres minijuegos, cada uno con su propia idea.

2. Desarrollo

2.1. Herramientas

Para desarrollar la aplicación con éxito durante un cuatrimestre, hemos elegido Processing, lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java. Processing se utiliza principalmente para desarrollar aplicaciones visuales artísticas, aplicaciones multimedia interactivas, juegos pequenos y diseño generativo. Se puede utilizar con Windows y UNIX igualmente, nosotros hemos utilizado Windows.

Originalmente Processing empezó con syntax basada en Java y con lexicón de primitivas gráficas inspiradas por OpenGL, Postscript, Design by Numbers y otros. Hoy día se puede utilizar con modos diferentes de, por ejemplo, Python y JavaScript. Su uso es fácil y hay disponibles muchas librerías que se pueden añadir fácilmente, esto lo hace perfecto para introducir y enseñar programación a los artistas y otros no-programadores. Existen muchos tutoriales tanto de texto como de video y se pueden encontrar muchos ejemplos de código de referencia.

Nosotros utilizamos el modo de Java con tres librerías: video, sound y minim. Las primeras dos, de video y sonido, sirven para operaciones muy básicas. La librería de video permite mostrar y procesar input de cámara web y la librería de sonido nos permitió añadir respuesta de audio para mejorar la experiencia inmersiva de usuario. La última librería Minim nos sirve para el análisis del sonido y la transformación en señales visuales.

2.2. Color tracking

Para empezar, encontramos algunos tutoriales y algoritmos de color tracking en processing y elegimos uno que nos parecía adecuado: The Coding Train - 11.5: Computer Vision: Color

Tracking (fig. 1). Para cada cuadro, el algoritmo encuentra el punto con el promedio de un color concreto. en este ejemplo el color se elige pulsando algún píxel de la imagen del video en la pantalla con el ratón. De esta manera, permite seguir un objeto de color intenso.

Figura 1. Color Tracking Tutorial.

En el segundo paso, extendemos el código para rastrear dos colores, el primero se elegía con el clic izquierda del ratón, el segundo con el click derecho. Del código original eliminamos todo lo que consideramos innecesario para nuestra aplicación - necesitamos rastrear solo los dos puntos promedios de los dos colores. Durante la transformación del código original, obtuvimos algunos resultados muy interesantes, como una aplicación de dibujo con objetos de color reales (fig. 2).

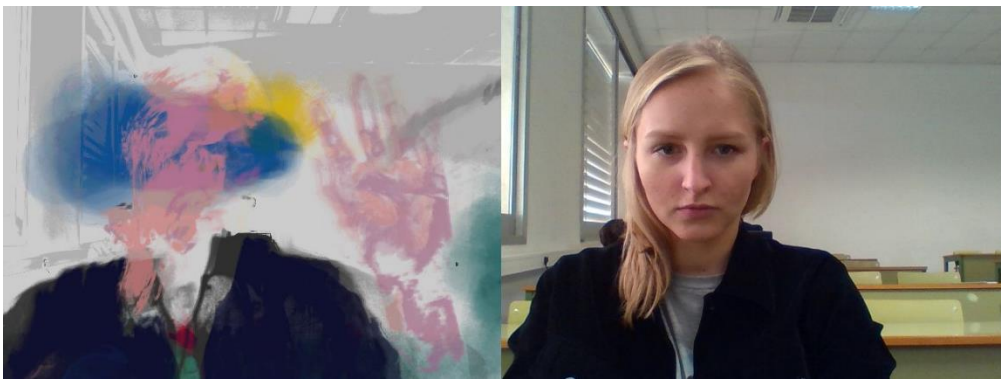


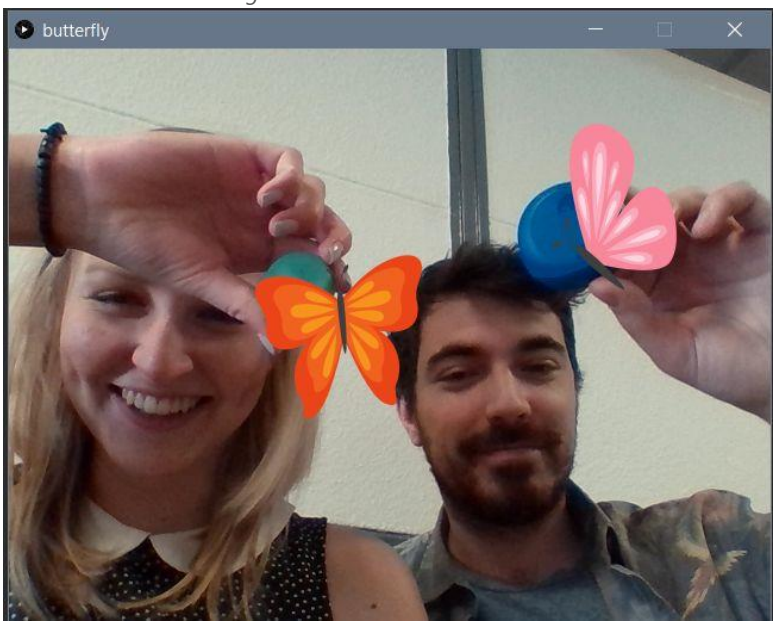
Figura 2. Dibujo con objeto de color.

Ya que nuestro objetivo era el de crear minijuegos fáciles de jugar y divertidos decidimos invertir la imagen que daba la cámara en pantalla para que fuese un efecto espejo. Ya que no tenía sentido que la imagen fuese al contrario de los movimientos del jugador.

Debido a problemas con la máxima resolución de imagen que podíamos obtener con processing junto con los cuadros por segundo decidimos escalar la imagen. Así obtuvimos una imagen que aunque inserta ciertos retardos en la visualización su visualización en pantalla, permite a los jugadores tener un tamaño de imagen considerablemente mejor para la experiencia dada la naturaleza de nuestros juegos.

Para representar el objeto de color en nuestro código, creamos objeto Catcher. Un catcher tiene unas coordenadas definidas por el punto de color promedio seleccionado previamente y una forma predefinida dada por el fichero SVG importado. El método más importante de catcher es intersect(), que asegura la interacción con otros objetos.

Figura 3. Realidad aumentada.



2.3. Pong

Para empezar, encontramos un código muy sencillo de single-player pong (https://www.openprocessing.org/sketch/47481/). Reemplazamos el fondo con el imagen de la cámara, asociamos la raqueta con el catcher y añadimos otro catcher para hacerlo dual-player. Creamos una clase Ball y le añadimos una imagen SVG, para que parezca más realista. Finalmente añadimos sonido.

Tuvimos que hacer algunos cambios de dificultad y velocidad, programar la lógica para jugar y ganar. Nos sirvió como un buen comienzo para empezar con Processing y para probar hacer la lógica general de los juegos.

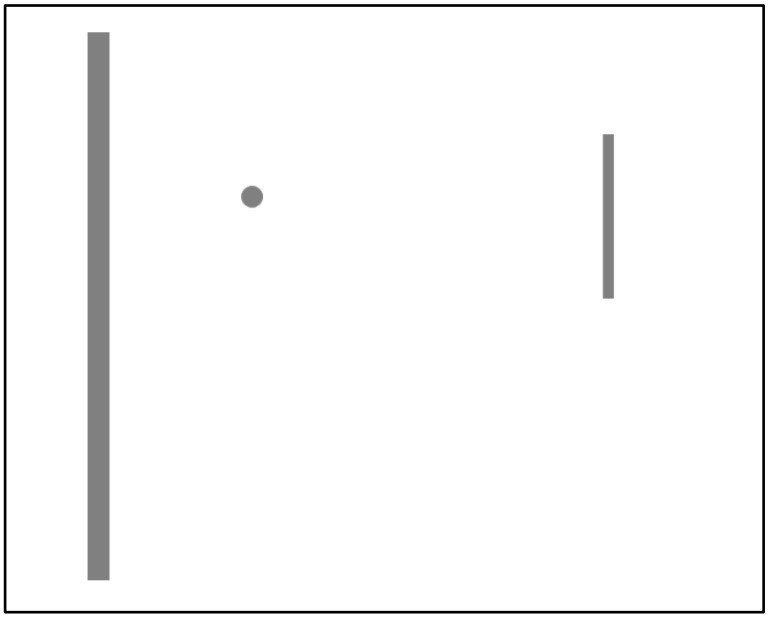


Figura 4. Simple pong.

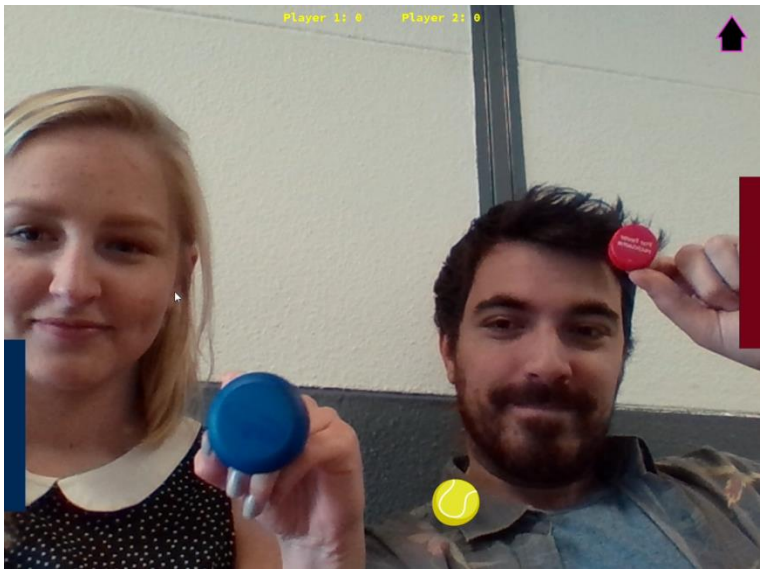


Figura 5. Pong.

2.4. Falling Drops

El juego de falling drops también se basa en codigo de juego muy sencillo que encontramos online: Learning Processing - Rain Catcher (http://learningprocessing.com/examples/chp10/example-10-10-rain-catcher-game). De manera similar a Pong, cambiamos el fondo y asociamos al objeto catcher del juego con nuestro objeto catcher. Para hacer el juego más interesante, extendemos la clase Drop, para que existan tres tipos de Drop: drop de primer jugador, drop de segundo jugador y bomba. Añadimos el segundo jugador, formas de SVG y sonido. Además, hemos optimizado algunos métodos, especialmente la parte de que una gota sea "cogida" y removerla del ArrayList (fig. 6 y 7, en el código original solo se mueve afuera de la pantalla, pero sigue siendo un elemento del ArrayList).

```
// Move and display all drops
for (int i = 0; i < totalDrops; i++ ) {
  drops[i].move();
  drops[i].display();
  if (catcher.intersect(drops[i])) {
    drops[i].caught();
  }
}
```

```
// If the drop is caught
void caught() {
  // Stop it from moving by setting speed equal to zero
  speed = 0;
  // Set the location to somewhere way off-screen
  y = -1000;
}
```

Figura 6. Rain catcher código original.

Tuvimos que decidir sobre el score y medir el tiempo, sobre cómo se puede ganar o perder. Un jugador gana 20 puntos cuando coge su gota, pierde 10 puntos cuando coge la otra gota (del otro jugador) y pierde un punto de vida cuando coge una bomba. El juego termina después tiempo predefinido (mostrado por la barra de progresso abajo, fig. 8) o cuando un de los jugadores pierde todas sus vidas.

```
for ( int i = 0; i < drops.size(); i++ ) {

  Drop d = drops.get(i);

  if (d.y > height + 100) {
    // removing elements that are no longer visible makes the iterations shorter
    drops.remove(i);
    i--;
    continue;
  }

  d.move();
  d.display();

  if ( catcher1.intersect(d) ) { // caught by the first one
    d.caught();
    if ( d.isOfType( DropType.DROP_1) ) {
      player1.addScore(DROP1_VAL);
    } else if (d.isOfType( DropType.DROP_2)) {  // if caughts other players drop, looses half of its value
      player1.substractScore(DROP2_VAL/2);
    } else {  // if hits bomb, looses lives
      player1.looseLife();
    }
  } else if ( catcher2.intersect(d) ) { // caught by the second one
    d.caught();
    if ( d.isOfType( DropType.DROP_2) ) {
      player2.addScore(DROP2_VAL);
    } else if (d.isOfType( DropType.DROP_1)) { // if caughts other players drop, looses half of its value
      player2.substractScore(DROP1_VAL/2);
    } else { // if hits bomb, looses lives
```

Figura 7. Rain catcher código cambiado.

En esta etapa de desarrollo un elemento muy importante fue programado: la barra de progresso (fig. 9). Al principio se solo utilizó para mesurar el tiempo del juego, pero después fue muy beneficioso para la interacción con botones sin contacto físico.

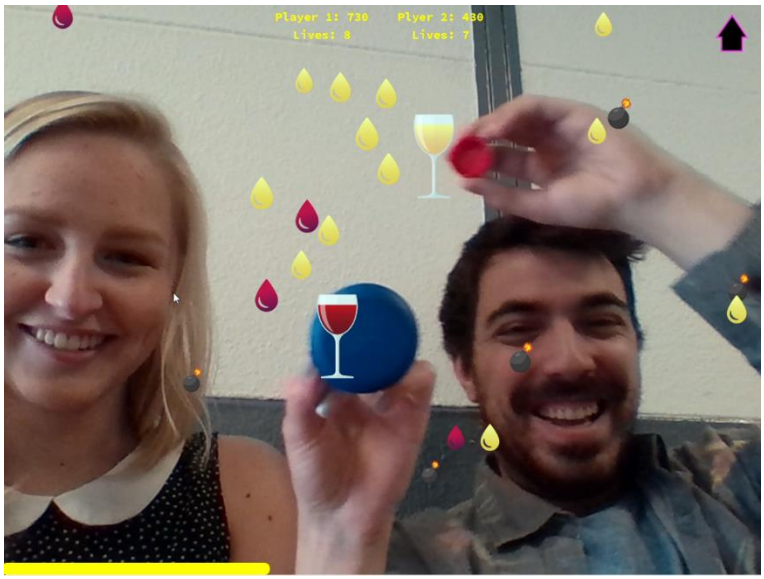


Figura 8. Falling Drops.

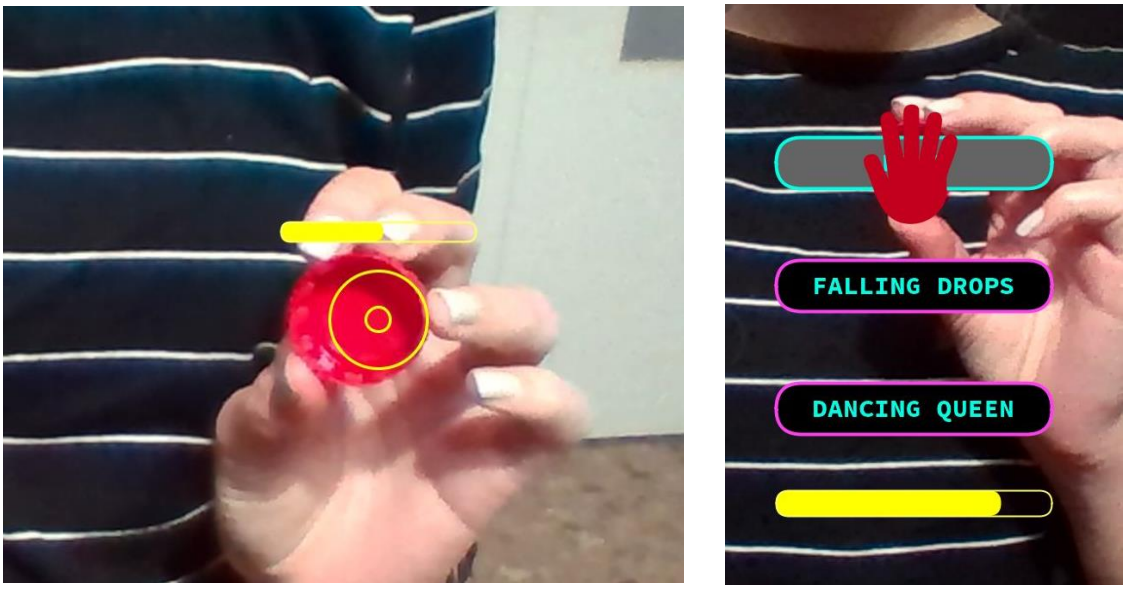


Figura 9. Barra de progresso.

2.5. Dancing Queen

El desafío de Dancing Queen fue encontrar una buena librería para análisis del sonido y convertir el ritmo de una canción en señales visuales. En librería Minim existe metodo de BeatDetect (http://code.compartmental.net/minim/beatdetect_class_beatdetect.html). La documentación de esta librería con sus ejemplos (fig. 10) nos ayudó a desarrollar algoritmo, que llena círculos sobre el ritmo de alguna canción.

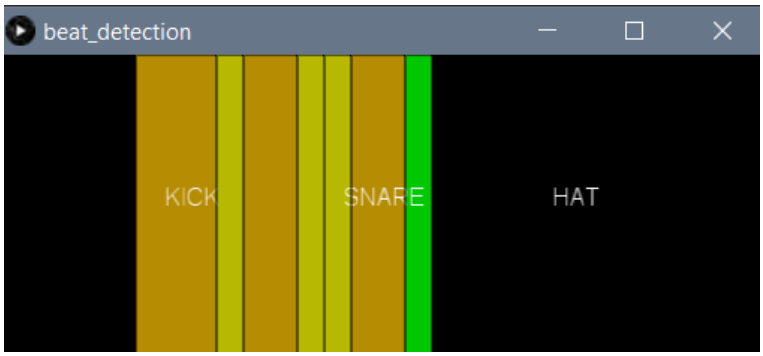


Figura 10. Beat detection.

Eliminamos todo innecesario del algoritmo original y lo integramos en nuestra aplicación. Tuvimos que cambiar los parámetros del beat detection muchas veces para obtener resultados jugables. La versión final del juego no es demasiado fácil, pero es muy sencillo cambiar número de los círculos o las constantes que afectan a la dificultad y velocidad del juego.

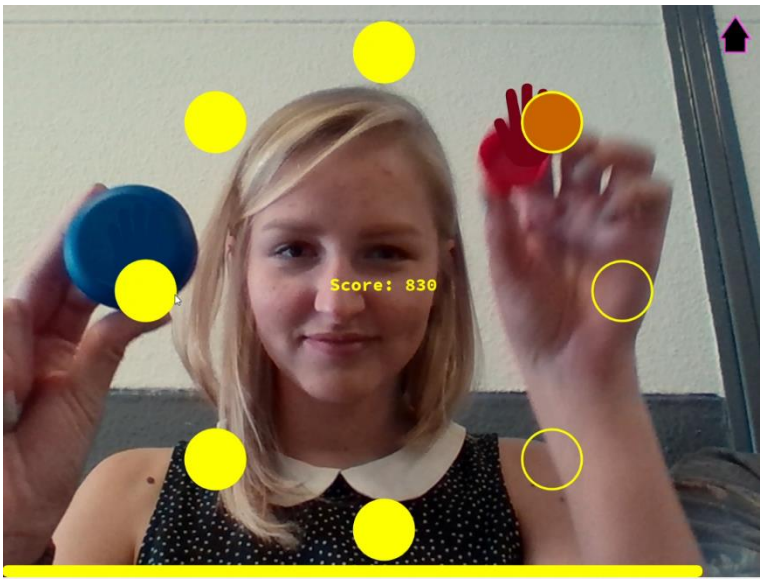


Figura 11. Dancing Queen.

Para mejorar la experiencia de usuario mostramos abajo una barra de progresso que reflecta longitud de la canción elegida y además cuando el jugador toca correctamente un círculo lleno, la color amarillo del círculo se cambia a naranja (fig. 11). En el centro el jugador puede ver su score que se aumenta con cada círculo bien tocado.

2.6. Integración

Un otro desafío grande vino con la integración de los tres juegos en un juego completo. Tuvimos que desarrollar una manera de seleccionar color sin fisicamente tocar el teclado o el ratón.

Así pues desarrollamos un clase encargada de realizar esta función. La forma de interactuar es sencilla, hay dos círculos en la imagen que indican la zona donde debe colocarse el objeto que se pretende utilizar como controlador del juego. Se debe mantener este objeto durante un periodo de tiempo en esa zona hasta que se nos indique que se ha capturado el color correctamente. Una vez ambos jugadores hayan elegido su color, se procederá a la selección del juego.

La idea base del selector de color automático descrito anteriormente es bastante sencilla. Se toma el color de uno de los primeros frames que captura la cámara como color base, así cuando un objeto de un color lo suficientemente diferente se coloque en esa misma posición de la imagen se empezará a capturar el color. Hay un tiempo de margen, para que no se escoja un color por equivocación como por ejemplo por un cambio de luza repentina.

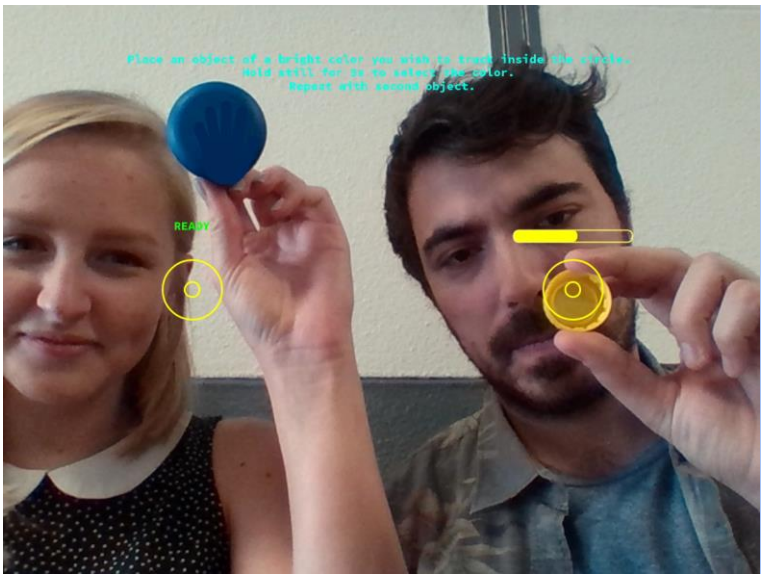


Figura 12. Selección de color.

La manera de controlar el menú se inspira en juegos inmersivos modernos: hay que poner el catcher sobre el botón y mantenerlo allí durante 3 segundos. Una barra de progreso se aparece para demostrar el tiempo que queda de los 3 segundos. El mismo tipo de botones y interacción se utiliza también cuando un juego termina y los opciones de ir atrás o continuar aparecen (fig. 13). Además, cuando un juego está activo, aparece un botón en forma de casa para volver al menú principal.

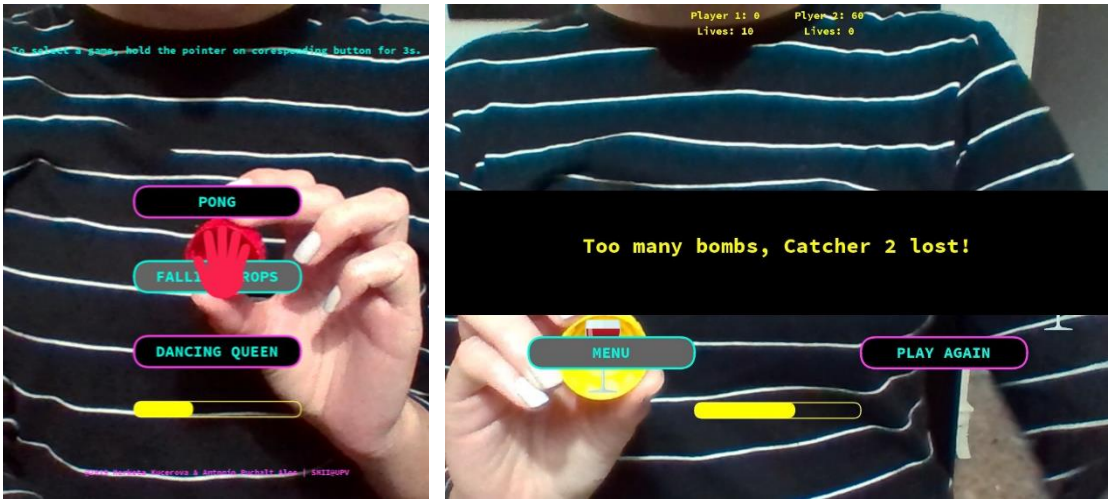


Figura 13. Interacción.

En el código la selección de juego se controla con una variable gameState y constantes diseñadas para esta variable (fig. 14). Al principio después de ejecutar el código, estamos en estado de "GET_COLORS" y después continuamos al "MENU" y a una de los 3 juegos. Dentro cada juego el estado del juego se controla por playState.

```
// game constants
// there are two types
// this one is for playState of each game
final int GAME_OFF = -11;
final int GAME_ON = 11;
final int GAME_LOST = 12;
final int GAME_OVER = 13;
final int GAME_START = 14;

// this type is for gameState of the whole system
final int GET_COLORS = 100;
final int MENU = 101;
// these have to be normal 0,2,3, because they also function as array indexes in the menu part
final int PONG = 0;
final int FALLING_DROPS = 1;
final int DANCING_QUEEN = 2;
```

Figura 14. Game state control.

Somos conscientes de las limitaciones y las condiciones en las que deben jugarse nuestros juegos para que sea una experiencia satisfactoria. El punto principal es el cambio de la luz a lo largo del tiempo de juego, ya que es necesario un fondo de un color bien definido y una luz ambiente estable para que se puedan rastrear de forma eficiente los colores con los que se pretende jugar.

Por otro lado, hemos tenido bastantes problemas para poder llevar a cabo la creación de un archivo ejecutable que contuviese todo el juego, hasta el punto de que nos ha sido imposible solucionarlo para la entrega de este proyecto. Como solución provisional hemos considerado la opción de que el usuario que esté interesado en jugar a nuestro juego deberá descargar Processing en su máquina e instalar las librerías necesarias. Recomendamos la versión 3.5.3 de processing pues es en la que se ha desarrollado íntegramente el juego. Las librerías necesarias son Sound, Video y Minim que pueden ser descargadas fácilmente desde

el entorno de processing. Este proceso está explicado más detalladamente en el archivo léeme.txt que se encuentra en TrabajoSMI -> theGame.

3. Conclusión

Durante el desarrollo de este juego hemos aprendido a trabajar con nuevas herramientas y plantear nuevos enfoques. Los principales retos a los que nos hemos enfrentado han sido tanto un nuevo entorno de desarrollo como aprender a superar sus limitaciones y sacar partido de sus ventajas. Al principio elegimos trabajar con código ya existente (código libre, código de tutoriales) mientras que al final fuimos capaces de desarrollar nuestras propias clases y algoritmos desde cero, como pueden ser "AutomaticColorSelector" y "ProgressBar".

El conocimiento aprendido durante este proyecto con processing nos puede servir en un futuro para el desarrollo de pequeños proyectos interactivos. Podemos considerar también posibles ampliaciones o mejoras a nuestro juego.

Como conceptos a mejorar podemos identificar los siguientes:

- Afinar más la selección automática de color para que no falle tanto y sea más estable con cambios de luz.
- La creación del ejecutable para que pueda ser portable.
- Automatizar la selección de la resolución de la cámara para así facilitar la portabilidad.
- Tratar de mejorar la jugabilidad propia de cada juego añadiendo reglas y modos de juego.
- Tratar de mejorar la atmósfera y la experiencia del jugador mediante mejores efectos de sonido o visuales.

Ya que nuestro principal objetivo era intentar realizar un proyecto inmersivo, divertido y crear una experiencia de usuario decente. Hemos probado nuestro juego con amigos y conocidos. Así que, como conclusión final diremos que la satisfacción de haber aprendido nuevas habilidades ha sido realzada al ver a nuestros amigos jugando entre ellos, pasando un buen rato y disfrutando de la experiencia.

4. Bibliografía

[1] "Learning Processing," *Learning Processing 2nd Edition*, fecha de consulta 15 abril 2019, en http://learningprocessing.com/.

[2] PROCESSING FOUNDATION, "Language Reference (API) \ Processing 3 ," *Processing*, fecha de consulta 15 abril 2019, en https://processing.org/reference.

[3] QUARTZ, D., "Minim Documentation," *Compartmental*, fecha de consulta 15 abril 2019, en http://code.compartmental.net/minim/.

[4] SHIFFMAN, D., "Rain Catcher Game," *Learning Processing 2nd Edition*, fecha de consulta 15 abril 2019, en http://learningprocessing.com/examples/chp10/example-10-10-rain-catcher-game.

[5] LOVATO, G., "Simple Pong," *Open Processing*, fecha de consulta 15 abril 2019, en https://www.openprocessing.org/sketch/47481/.