PROYECTO DDS

PacMan UPV

Marcos Manuel Gallego Ferrer
Alfredo Moises Boullosa Ramones

Índice.

Descripción del proyecto.

Introducción.

El proyecto que hemos escogido para su desarrollo, tratando de implementar varios de los patrones vistos en el aula, ha sido el conocido juego recreativo “Pac-Man”.

Antes de centrarnos en los principales puntos a desarrollar en el trabajo, conozcamos un poco de su historia y de su funcionamiento.

Pac-Man es un videojuego de tipo arcade creado por el diseñador de videojuegos Toru Iwatani, propiedad de la empresa Namco. Surgió a principio de los años 80, concretamente el 21 de mayo, y en apenas semanas, se convirtió en un videojuego viral, de gran éxito y expansión por todos los países. Es por ello que es el mas exitoso de todos los tiempos, ya que destrono al juego de Space Invaders.

En España, este juego es conocido como el comecocos, y fue introducido en los 80’s mediante la plataforma Atari 2600.

¿En qué consiste el juego?

El protagonista de nuestro videojuego, es un circulo amarillo, al que le falta un sector, a modo de boca. Dicho circulo, recorre una serie de laberintos, que están sembrados de unos pequeños puntos conocidos como Pac-Dots, puntos mayores y otros premios con forma de fruta, que añaden una serie de comportamientos o estados al personaje. El objetivo de nuestro personaje, es acabar con todos estos puntos. ¿Parece fácil no? Para nada, ya que este laberinto está custodiado por los conocidos fantasmas, que trataran de impedir que nuestro personaje complete con vida el nivel, persiguiéndole por todo el recorrido. Son conocidos por los siguientes nombres, que denotan su comportamiento:

- Shadow (Blinky) → Es como la sombra del personaje, está continuamente detrás de el.
- Speedy (Pinky) → Su destino es 2 bloques por delante de Pacman.
- Bashful (Inky) → Su algoritmo de persecución utiliza la ubicación de Blinky y Pacman. Su nombre, Bashful, significa timido, esto se deba a que si Inky está con Blinky, sus movimietnos son como los de Blinky, pero si Pacman esta de por medio Inky tiende a alejarse de Pacman.

Pokey (Clyde) → Como su nombre indica(Poke-Pinchar) se mueve de una forma de ataque-retirada al pasar cierto rango de distancia de Pacman.

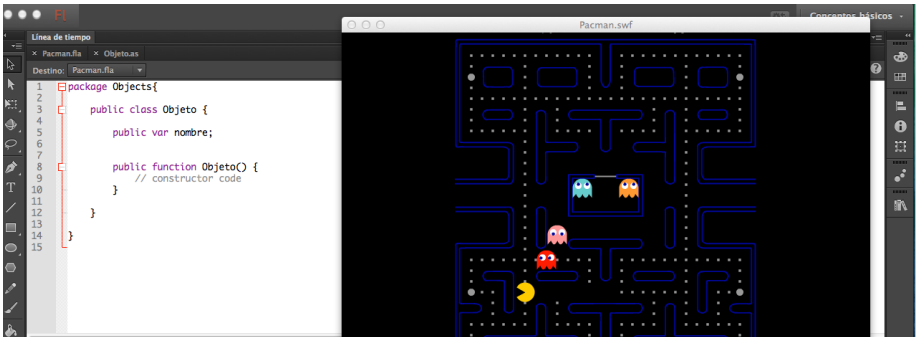
Estos fantasmas son de colores diferentes. Además, en las esquinas del tablero, encontramos cuatro puntos mas grandes de lo normal, conocidos como Power Pellets o "Pildoras", y que al comérselas Pacman, invierten la dirección de los fantasmas y los ralentiza, adema de darle la habilidad a Pacman de poder comérselos durante unos segundos, y estos han de volver del punto de partida, para poder restaurar su estado normal, y poder volver a perseguirnos. Este tiempo en el que el fantasma es vulnerable a nuestros ataques, decrece de manera que se aumenta los niveles del juego.

Tecnología utilizada.

La plataforma que hemos escogido para desarrollar el proyecto es Adobe Flash CC, de la compañía Adobe Systems Inc. Se trata de una aplicación de creación y manipulación de gráficos vectoriales, con el añadido de que nos permite el manejo de dichos gráficos mediante código, mediante el lenguaje ActionScript. Flash es básicamente un estudio de animación programable, la cual, como toda animación, se basa en los fotogramas. El hecho de que sea programable a nuestro parecer, la hace la herramienta perfecta, debido a la facilidad de uso que representa el "dibujar" los objetos que van a ser empleados en la interfaz de nuestra aplicación. Además, una de las ventajas de utilizar está plataforma, es que es capaz de exportar los proyectos a una gran variedad de plataformas, una vez nuestro proyecto esté finalizado y todo compile como nosotros esperamos. Es capaz de crear ejecutables para MAC OSX, Windows, ejecutables en HTML, app's para IOS y Android… En definitiva, es una herramienta muy poderosa, y muy útil en nuestro caso.

En este caso, Adobe Flash requiere que la programación se haga en ActionScript, lo que le puede añadir dificultad al proyecto, si no se está familiarizado con este lenguaje de programación. Actualmente, corre con la versión 3.0. Este lenguaje es un lenguaje orientado a objetos, los cual nos ha beneficiado, ya que el lenguaje que comúnmente empleamos es Java. Por tanto, la única dificultad en la comprensión del lenguaje es la sintaxis que difiere respecto a Java.

Por estas razones, hemos decidido desarrollar nuestro proyecto en Flash, ya que herramientas como Eclipse o Netbeans, no soportan este tipo de lenguaje, y la facilidad y versatilidad que tiene la combinación del ActionScript con la animación por dibujo en la interfaz de desarrollo. Es decir, con unos sencillos pasos, como es dibujar un circulo, darle el color deseado, un nombre y unos parámetros concretos, podemos hacer de el nuestro personaje del juego, dotándole de los respectivos métodos de movimiento, para que responda a nuestras pulsaciones en el teclado.



Diseño arquitectónico.

El diseño arquitectónico que sigue la aplicación es “sencillo”. Al tratarse de un videojuego, está orientado totalmente a la interacción entre el usuario, y el programa en ejecución, que es percibido por el usuario a través de la pantalla.

La aplicación, mediante los patrones creacionales que mas adelante presentaremos, inicializa todos los elementos que forman parte de la interfaz, y mediante la interacción del usuario mediante el teclado, recibe como feedback la interacción continua en la pantalla, y también a modo de audio, ya que en el juego se ha implementado el conocido “waka-waka” que se produce al desplazar el personaje por la interfaz.

No hemos considerado necesaria la implementación de una capa de persistencia, ya que no supone una mejoría en la aplicación. Si que es verdad, que mediante una aplicación que implementara persistencia, podríamos almacenar pares usuario-puntuación, reflejando las mejores partidas jugadas.

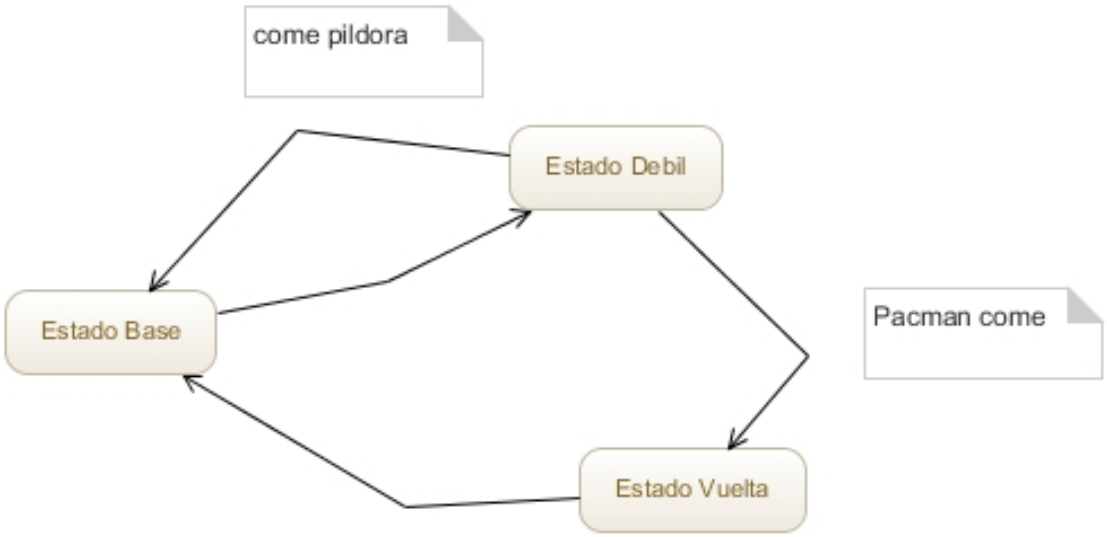
Otra posible ampliación de la aplicación, sería un modo multi-jugador, en el que se pudiera o competir por puntuación mas alta contra otro jugador, o modo superviviente en el mismo interfaz, pero desde las dos computadoras en acción. Es decir, necesitaríamos un protocolo para que las dos aplicaciones se comunicaran a tiempo real y con coherencia entre si, por lo que se necesitaría la habilitación de un servidor capaz de manejar ambos flujos de datos.

Patrones de diseño aplicados.

Patrones de Comportamiento

Patrón de Estados.

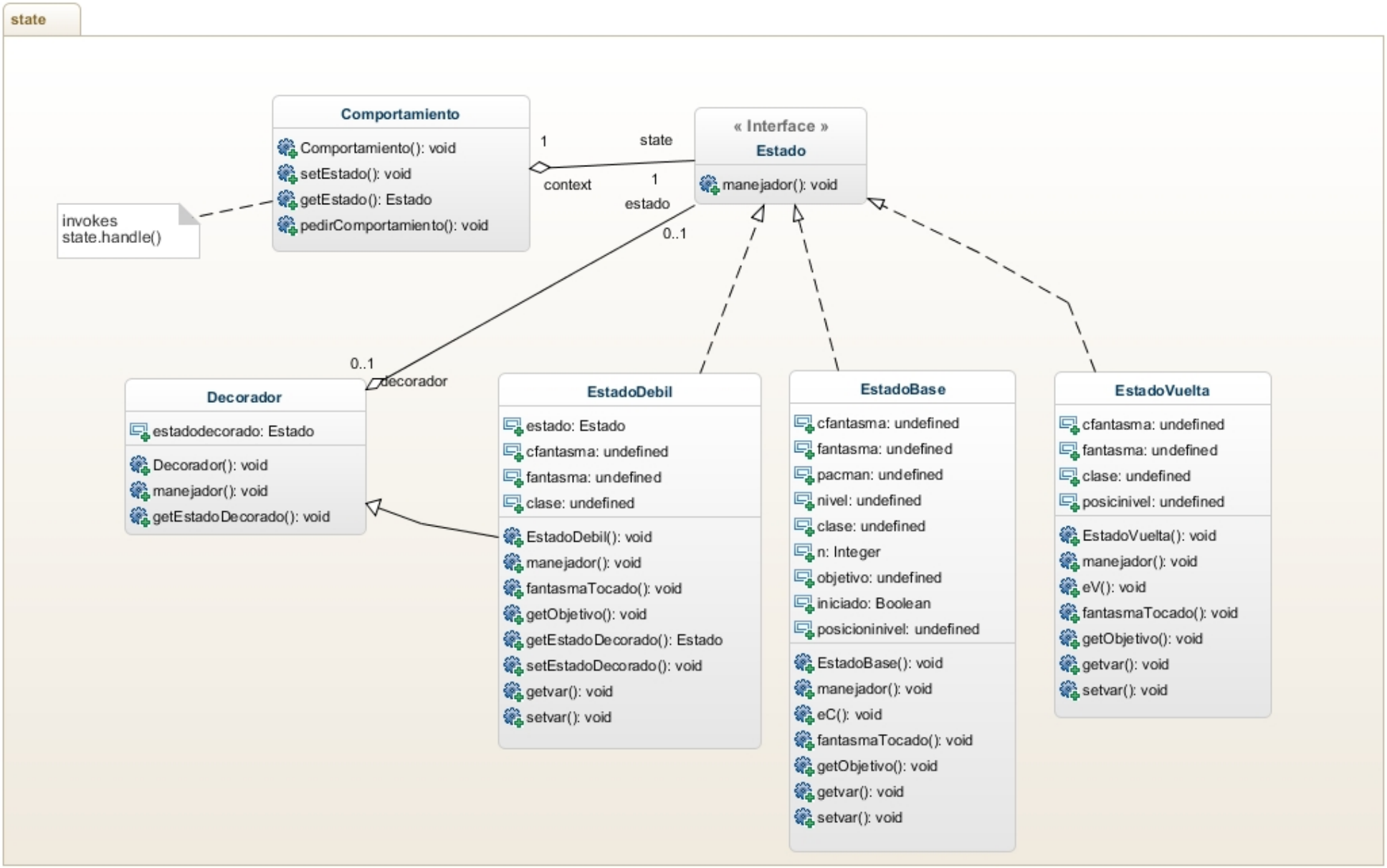
Hemos implementado este patrón, junto con el patrón decorador, para de esta manera, poder reflejar los cambios de comportamiento que tienen los personajes del juego al interactuar en la partida. El decorador es el que se encarga de añadir o quitar el estado débil al estado base. El siguiente diagrama nos muestra los diferentes estado por los que puede cambiar el juego:



La partida empieza en un estado normal, que corresponde a la clase EstadoBase, en la que se encarga de perseguir a nuestro personaje.

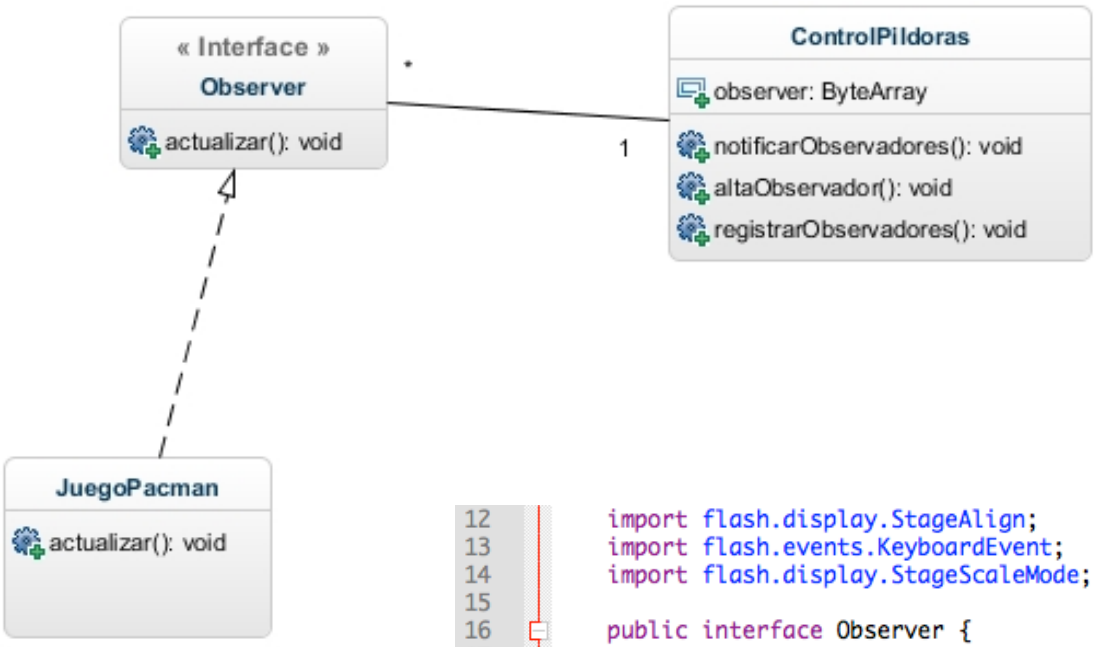
Si nuestro personaje come una de las píldoras, nuestro personaje puede comer a los fantasmas y obligar que estos vuelvan a casa para retomar su estado inicial, o trascurrido un corto periodo de tiempo, si no son comidos, regresan al estado inicial. En este caso, el estado en el que pueden ser comido corresponde a EstadoDebil, y si realmente son comidos, pasan a EstadoVuelta.

El esquema general de las clases está a continuación:


state
Comportamiento
Comportamiento(): void
setEstado(): void
getEstado(): Estado
pedirComportamiento(): void
invokes
state.handle()
1
state
context
1
estado
0..1
« Interface »
Estado
manejador(): void
0..1
decorador
Decorador
estadodecorado: Estado
Decorador(): void
manejador(): void
getEstadoDecorado(): void
EstadoDebil
estado: Estado
cfantasma: undefined
fantasma: undefined
clase: undefined
EstadoDebil(): void
manejador(): void
fantasmaTocado(): void
getObjetivo(): void
getEstadoDecorado(): Estado
setEstadoDecorado(): void
getvar(): void
setvar(): void
EstadoBase
cfantasma: undefined
fantasma: undefined
pacman: undefined
nivel: undefined
clase: undefined
n: Integer
objetivo: undefined
iniciado: Boolean
posicioninivel: undefined
EstadoBase(): void
manejador(): void
eC(): void
fantasmaTocado(): void
getObjetivo(): void
getvar(): void
setvar(): void
EstadoVuelta
cfantasma: undefined
fantasma: undefined
clase: undefined
posicinivel: undefined
EstadoVuelta(): void
manejador(): void
eV(): void
fantasmaTocado(): void
getObjetivo(): void
getvar(): void
setvar(): void

Observer

Para que todos los objetos, estén al tanto de que es lo que está ocurriendo en cada momento, y actúen según sus respectivos comportamientos, como puede ser el que pacman coma una píldora, y todos los fantasmas tengan que cambiar de estado y de dirección, al ser avisados por dicho patrón. Por ejemplo, para resumir la modelización del patrón, hemos definido en el siguiente esquema la estructura que sigue por ejemplo el control de las píldoras, y como es notificado.




```
12      import flash.display.StageAlign;
13      import flash.events.KeyboardEvent;
14      import flash.display.StageScaleMode;
15
16      public interface Observer {
17
18          /*public function Observer() {
19              // constructor code
20          }*/
21          function actualizar();
22
23      }
24
25  }
26
```

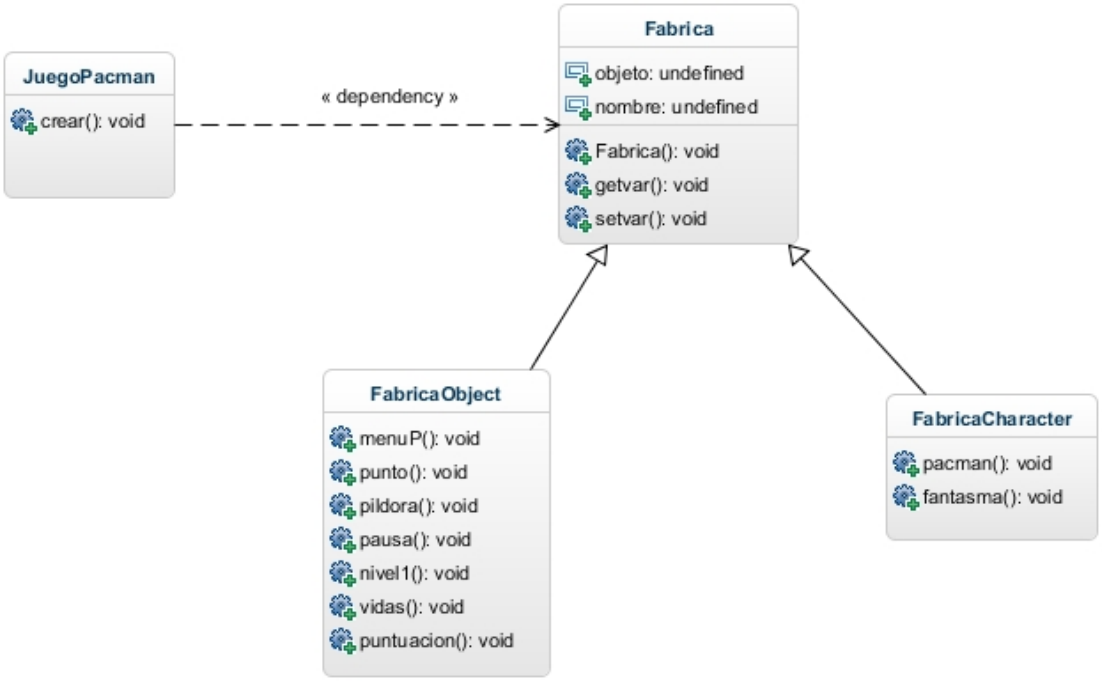
```
public function ControlPildoras(){
    observers=new Array;
}

public function registrarObservador(observador){
    if(!searchArrayVar(observers,observador))
        observers=addArrayVar(observers,observador);
}

public function altaObservador(observador){
    if(searchArrayVar(observers,observador))
        observers=removeArrayVar(observers,observador);
}

public function notificarObservadores(){
    for(var i=0;i<observers.length;i++){
        observers[i].actualizar();
    }
}
```

Patrones creacionales

Fabrica

Como nuestra aplicación, para su inicialización necesita de una gran cantidad de objetos, que son los que interactúan entre si, y le dan la funcionalidad al juego, el patrón mas claro a implementar era el fabrica.

El esquema, nos muestra como desde JuegoPacman, mandamos mediante el método crear, el cual nos devolverá el objeto que hayamos mandado crear, ordenamos a la fabrica que nos cree dicho objeto. Fabrica lo hemos divido en 2. Una fabrica para los elementos del juego, como puede ser los puntos, las píldoras, el menú… etc. Que corresponde con FabricaObject, y FabricaCharacter, que es la encargada de construir los objetos pacman, y los objetos fantasma.



Refactorización aplicada.

Código duplicado

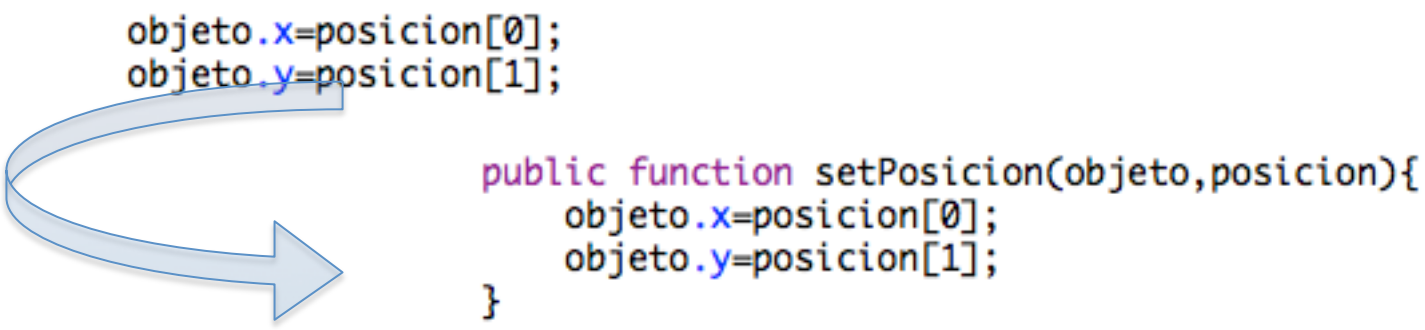
En la clase JuegoPacman podíamos encontrar el siguiente código, que se repetía al perder y al subir el nivel, por lo que como solución, se ha compactado el código común en un método llamado "eliminartodo()", que se ejecutará en cada uno de los métodos correspondientes a perder → "gameover()" y subir nivel →"aumentarnivel()".

```
cpacman.eliminar();
cfantasma1.eliminar();
cfantasma2.eliminar();
cfantasma3.eliminar();
cfantasma4.eliminar();
eliminarPuntos();
clasep.removeChild(vidas0);
clasep.removeChild(puntuacion0);
clasep.removeChild(nivel);

clasep.removeEventListener("enterFrame",tr);
escenario.removeEventListener(KeyboardEvent.KEY_DOWN,KeyDown);
```

```
public function eliminarObjetos(){
    cpacman.eliminar();
    cfantasma1.eliminar();
    cfantasma2.eliminar();
    cfantasma3.eliminar();
    cfantasma4.eliminar();
    eliminarPuntos();
    clasep.removeChild(vidas0);
    clasep.removeChild(puntuacion0);
    clasep.removeChild(nivel);

    clasep.removeEventListener("enterFrame",tr);
    escenario.removeEventListener(KeyboardEvent.KEY_DOWN,KeyDown);
}
```

```
public function gameover(){
    eliminarObjetos();
    escenario.frameRate=18;
    clasep.crear("menuP",this);
    clasep.setPosicion(clasep.getvar("menuP"),posicioni);
    clasep.setvar("cmenuP",new MenuPrincipal(clasep.getvar("menuP"),escenario,clasep));
}
```

También, al iniciar cada objeto, se repetía el código de inicialización para cada objeto, por lo que se creó el método "setPosicion(objeto, arrayPosicion[])", de esta manera se consigue simplificar la inicialización de los objetos.



Simplificar las llamadas a métodos y Data Clumps

Puesto que la posición x e y se utilizaban siempre juntas y eran llamadas por métodos como "setPosicion()" todas las posiciones x e y se unieron en arrays como:

```
public var posicioni=[512,382]; //Posicion inicial
public var posicioninivel;      //Posicion inicial nivel
public var posicionipacman;     //Posicion inicial pacman
public var posicionifantasma1;  //Posicion inicial fantasma1
public var posicionifantasma2;  //Posicion inicial fantasma2
public var posicionifantasma3;  //Posicion inicial fantasma3
public var posicionifantasma4;  //Posicion inicial fantasma4
```

Los cuales se inician luego, y son llamados en el método "setPosicion()" . De esta manera "setPosicion(objeto, posición[])", en vez de "setPosicion(objeto, x, y)", simplificando de esta manera la llamada.

Simplificar las llamadas a métodos

El método "crear()" pedía 4 argumentos, puesto que según el objeto se podían requerir de 2 a 4 argumentos, pero se ha simplificado de manera que se necesitan 2 argumentos para todos los casos, por lo que la llama al método pasa de ser: "crear(objeto, padre, variable1, variable2)" → "crear(objeto, padre)".

Agregar método a clase

En la clase Fantasma:

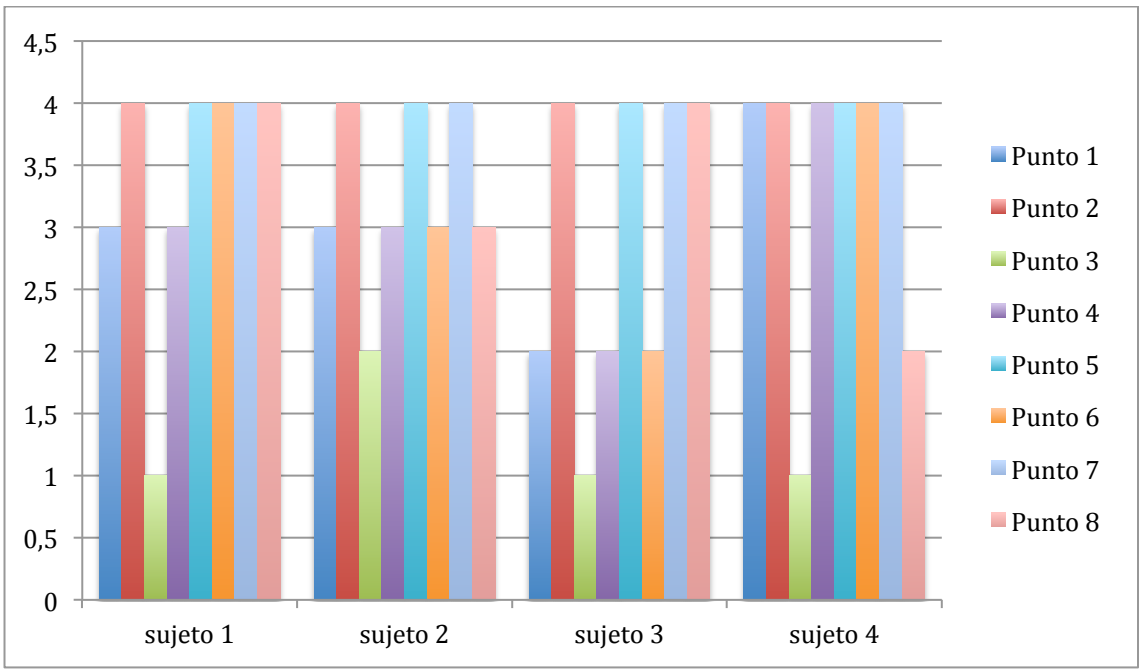
```
public function setDir(dirs){
    if(dirs=="arriba"){
        dir="arriba";
        velx=0;
        vely=vel;
        if(character.currentFrame<5) character.ojos.gotoAndStop("arriba");
    }else if(dirs=="abajo"){
        dir="abajo";
        velx=0;
        vely=-vel;
        if(character.currentFrame<5) character.ojos.gotoAndStop("abajo");
    }else if(dirs=="derecha"){
        dir="derecha";
        velx=vel;
        vely=0;
        if(character.currentFrame<5) character.ojos.gotoAndStop("derecha");
    }else if(dirs=="izquierda"){
        dir="izquierda";
        velx=-vel;
        vely=0;
        if(character.currentFrame<5) character.ojos.gotoAndStop("izquierda");
    }
}
```

Este código se ejecutaba dentro de un método que se ejecutaba en cada fotograma, así que para mejorar la estructura y optimizar el código se crea un método aparte para que solo se ejecute ese código cuando se necesite y no en cada fotograma.

Pruebas / Testing de la aplicación.

Al no tratarse de un proyecto desarrollado en Eclipse o Netbeans, no disponemos de la tecnología de JUnit para poder diseñar pruebas para verificar su correcto funcionamiento, por lo que, al tratarse de un videojuego, en la que la parte visual y su manejo es primordial para el éxito de la aplicación, hemos decidido realizar un tipo de "test" para que sea rellenado por un grupo de usuario, escogidos al azar, y de esta manera conocer sus opiniones, y los puntos en los cuales puede cojear la aplicación, o que mejoras se les podría introducir.

La plantilla a rellenar se encuentra en la siguiente pagina. El estudio lo hemos realizado sobre 5 personas, por lo que los datos no son de gran valor, por que el estudio se debería de realizar sobre un grupo de población mas extenso para que estos datos se adecuen a la verdadera opinión general sobre nuestra aplicación. El siguiente grafico muestra los resultados obtenidos.



El apartado Punto se refiere a cada una de las preguntas que se realiza en el test, donde el valor 1 significa totalmente desacuerdo, y 4 totalmente de acuerdo.

Una de las sugerencias que nos ha llamado la atención, y seria bastante llamativa, seria la posibilidad de adaptar un joystick, como se jugaba originalmente a este juego.

		Totalmente en desacuerdo	Un poco	Parcialmente	Totalmente de acuerdo
	PUNTUACION/OPINION.	1	2	3	4
1	La interfaz es llamativa.				
2	Es fácil de utilizar.				
3	Me ha costado poco entender su funcionamiento.				
4	Es entretenido				
5	No he encontrado aparentemente errores. (En caso negativo, describirlos en observaciones).				
6	Se la recomendaría a un amigo.				
7	Se adapta perfectamente a mi plataforma. (Windows, MAC OSX, navegador web).				
8	Sus controles son sencillos, para cualquier tipo de usuario.				

Observaciones/ Sugerencias:

DATOS IMPORTANTES

EDAD : **SEXO:**

¿Juegas de normal a juegos?:

MANUAL DE USUARIO

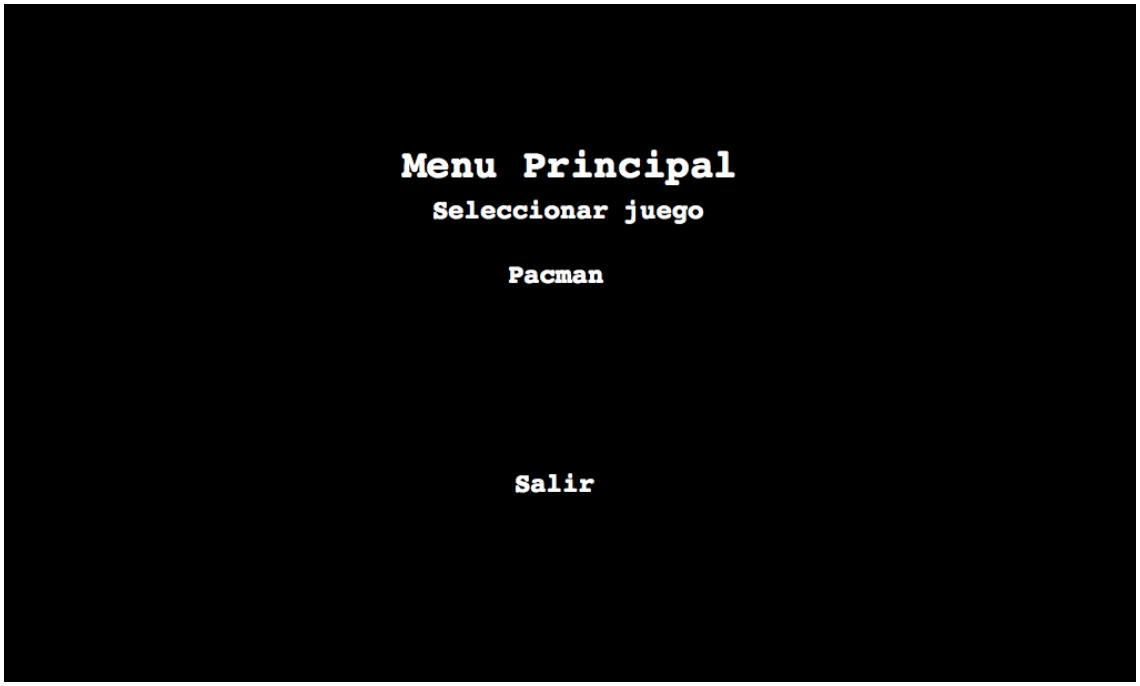
Índice

Requisitos y adaptabilidad del sistema

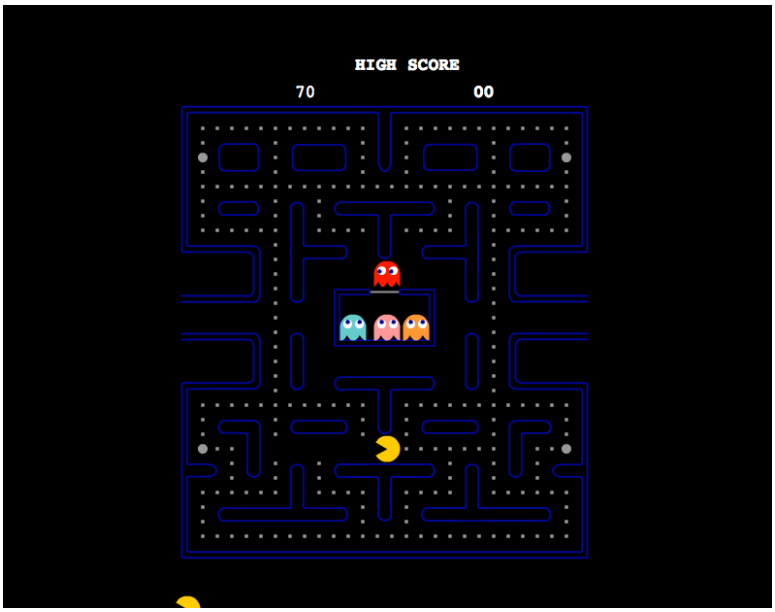
El videojuego no requiere de ningún tipo de instalación. Es ejecutable tanto en Windows, MAC OSX, y cualquier navegador (en el caso que se ejecute desde el fichero html).

Pantalla principal

Al iniciar la aplicación, nos aparecerá el menú de la aplicación, en el cual debemos de pulsar la tecla <space> o <Enter> para iniciar el juego.

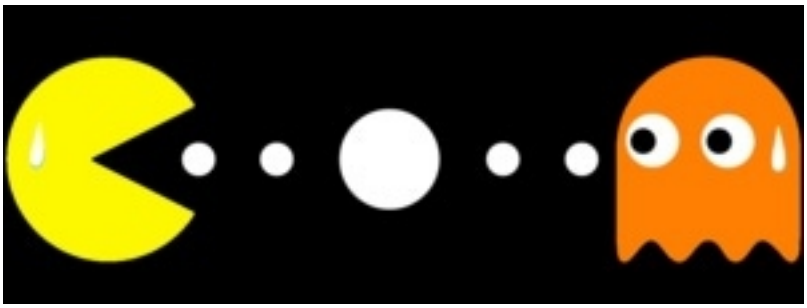


Al acceder a la pantalla de nuestro juego, tendremos ante nosotros el videojuego en pleno funcionamiento, como podemos observar:



La finalidad del juego consiste en superar cada nivel, sin ser atrapado por los fantasmas que van rondando por el laberinto, pero para ello, hay que recoger todos los puntos y las píldoras de cada tablero. Eso si, si somos atrapados, disponemos de tres vidas, una vez acabas, el juego termina, y volvemos al menú principal.

Pero no solo los fantasmas pueden comer a nuestro personaje. Si comemos una de las píldoras del tablero, que se diferencia por ser de mayor tamaño que los puntos que hay repartidos por el tablero, los fantasmas entran en un estado de pánico, en el que son vulnerables, y les podemos atacar, de manera que si son alcanzados por nuestra mordedura, y deben regresar al centro del tablero, desde donde empiezan la partida, para volver al estado inicial, y que puedan volver a atacarnos.



Controles del sistema

Para controlar a nuestro personaje solo disponemos de 4 movimientos:

- Arriba: flecha superior del teclado. O tecla W.
- Abajo: flecha inferior del teclado. O tecla S.
- Derecha: flecha derecha del teclado. O tecla D.
- Izquierda: flecha izquierda del teclado. O tecla A.