

Práctica de Programación Orientada a Objetos

Nombre: Juan Paulo

Apellidos: D'Urbano

Email: jdurbano1@alumno.uned.es

Telefono: 647901897

Análisis de la aplicación

Para la realización del trabajo se optó por un desarrollo iterativo ya que este se adapta mejor al problema porque permite desarrollo rápido partiendo de un prototipo rápido que proporciona una idea del aspecto que tendrá y lo que va a hacer permitiendo que el software puede ir creciendo en función de las necesidades que se necesiten.

1. Análisis de requisitos

En esta primera fase se hace un análisis para identificar las clases partiendo de los requisitos que debe cumplir el software y tener una idea del aspecto del diseño:

Clases:

- R1: analizando el requisito 1 se identifican las clases : **Pantalla, Jugador, Punto**
- R2: analizando el requisito 2 se identifican las clases : **Pacman**
- El jugador controlará a Pacman por tanto podría haber una relación de herencia entre las clases jugador y Pacman
- R3: analizando el requisito 3 se identifican las clases : **Fantasma**
- R4: analizando el requisito 4 se identifican las clases : **Teclado**
- R5: analizando el requisito 5 se identifican las clases : **Mapa**
- R8: analizando el requisito 8 se identifican las clases : **Blinky, pinky, Clyde** , al ser estos también fantasmas se identifica una relación de herencia entre estas clases y la clase fantasma
- R9: analizando el requisito 9 se identifican las clases: una clase Controlador que controle las acciones entre los fantasmas y Pacman. También tiene que controlar las colisiones entre los fantasmas
- R10, R11: analizando el requisito 10 y 11 se identifican las clases : **puntoChico, puntoGrande**, al ser estos también puntos se identifica una relación de herencia entre estas clases y la clase Punto
- El mapa por donde se mueve Pacman tendrá paredes : **Pared**
- Habrá una clase inicial para iniciar la aplicación

Campos de clase:

Teniendo las clases identificadas se analizará los campos que debe tener cada clase partiendo de los requisitos que se deben cumplir.

- **Jugador:**
 - **Vidas** : vidas del jugador
 - **Puntuación**: puntos que acumula el jugador
 - **Teclado**: teclas con las que el jugador juega al juego
- **Pacman:**
 - **Mapa**: mapa por donde se mueve Pacman
 - **set<puntos> puntos**: lista de puntos comidos por Pacman

- **set <fantasmas> fantasmasComidos** : lista de fantasmas azules comidos por pacman
 - **posición** : posición en el mapa de pacman
- **Fantasma:**
 - **Azul** : campo que determina si un fantasma es azul o no
 - **Valor** : valor en puntos de un fantasma (un fantasma azul comido vale 100 puntos)
 - **posición** : posición en el mapa del fantasma
- **Mapa:**
 - **Set<Puntos> puntos**: lista de puntos del laberinto
 - **Set<Pared> paredes**: lista de paredes del laberinto
 - **Límite**: límites del mapa
- **Punto**
 - **posicion** : posición del punto en el mapa
- **puntoChico:**
 - **valor**: valor en puntos de un puntoChico (un punto chico comido vale 10 puntos)
- **puntoGrande:**
 - **tiempo** : tiempo que dura la habilidad de un punto grande (un punto grande comido por pacman permite que los fantasmas sean azules durante 5 segundos)
- **Pared:**
 - **Posicion** : posición de la pared en el mapa
- **Teclado:**
 - **Pausa**: indica que se pulso la tecla de pausa
 - **Empezar**: indica que se pulso la tecla empezar
 - **Mueve**: indicara que se pulso algunas de las flechas de dirección
- **Controlador:**
 - **Empezar**: indica que debe empezar el juego
 - **Fin**: indica si es el fin del juego
 - **Pacman**: el controlador ejecuta acciones sobre pacam (como comprobar si este colisiona con algún fantasma por ejemplo, o si come puntos, etc)
 - **Set <Fantasmas>**: el controlador ejecuta acciones sobre los fantasmas (como comprobar si estos colisionan entre ellos por ejemplo, o si se tiene que poner en modo azul , etc)
- **Pantalla:**
 - **Controlador**: de donde se obtendrán los objetos para mostrar en pantalla
 - **Ancho**: ancho de la pantalla
 - **Alto**: alto de la pantalla

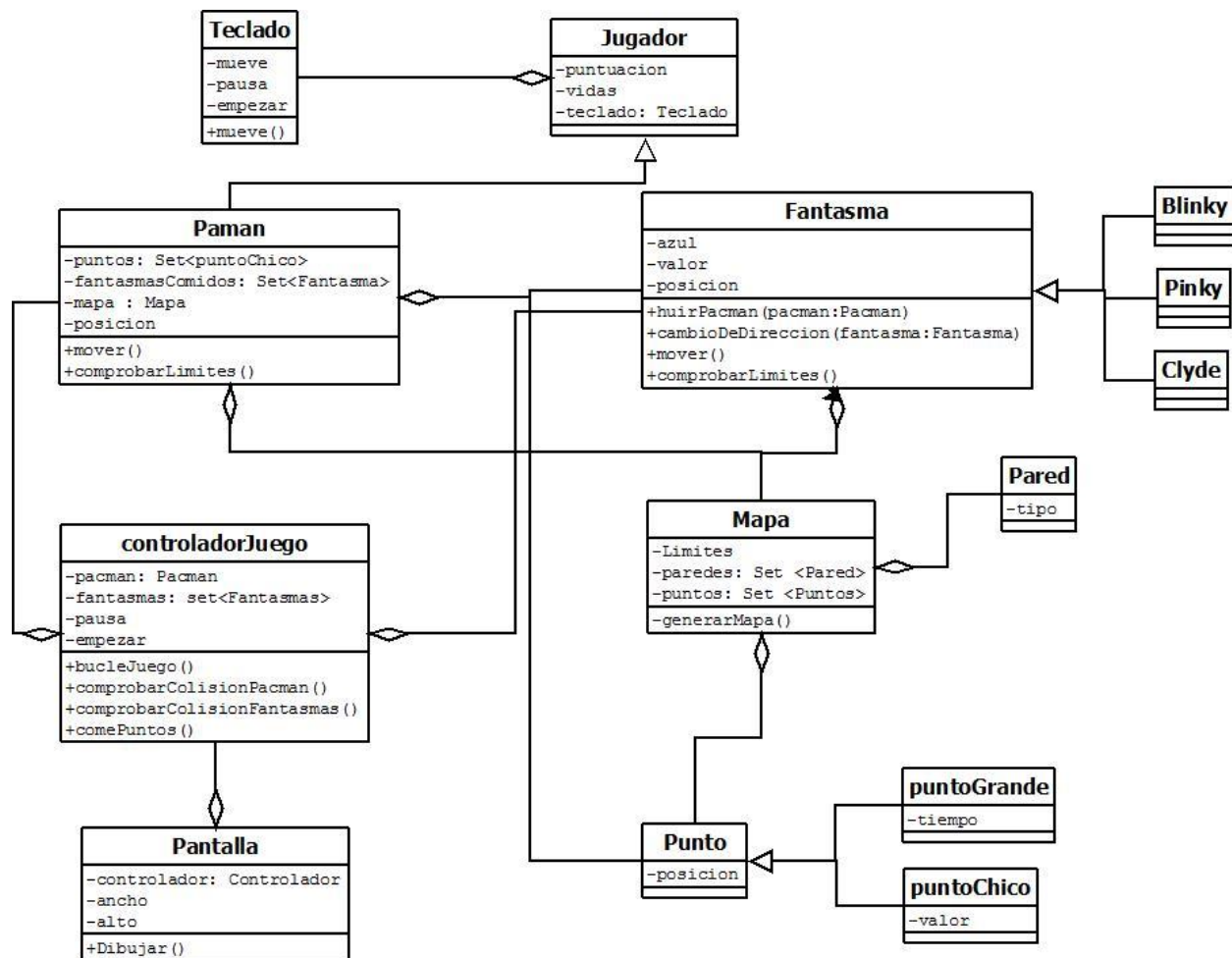
Métodos de clases:

Teniendo las clases identificadas y campos se analizara los métodos que debe tener cada clase partiendo de los requisitos que se deben cumplir.

- **Pacman:**
 - **Mover()** : Mueve a pacman
 - **comprobarLimites()**: comprueba que no superen se los límites del mapa antes de mover al personaje

- **Fantasma:**
 - **Mover():** mueve al fantasma
 - **huirPacman():** hace que el fantasma escape de pacman
 - **cambioDeDireccion():** hace que el fantasma cambie de dirección
 - **comprobarLimites():** comprueba que no superen se los límites del mapa
- **Mapa:**
 - **generarMapa():** se encarga de armar el laberinto con paredes y puntos
- **Teclado:**
 - **mueve():** indica que se pulso la tecla y que el jugador debe moverse
- **Controlador:**
 - **comprobarColisionPacman():** comprobar si pacman colisiona con algún fantasma
 - **comprobarColisionFantasmas():** comprobar si los fantasmas colisionan entre ellos
 - **comePuntos():** comprueba si pacman come puntos
 - **bucleJuego():** ejecuta el bucle del juego
- **Pantalla:**
 - **Dibujar():** dibuja los componentes del juego en pantalla

A partir del primer análisis de los requisitos se identifican, a nivel abstracto, las primeras clases con sus campos y métodos. Este primer diseño da una idea general de cómo se puede desarrollar la aplicación y es esta la base desde el cual se desarrolla toda la aplicación.



- Primer diseño de clases a nivel abstracto , el desarrollo de la aplicación se hará siguiendo este diseño

Luego del primer análisis se decide implementar un patrón de diseño modelo-vista-controlador porque se considera que es un patrón que se ajusta perfectamente a las necesidades ya que este permite mantener actualizada la vista y el modelo por medio del controlador

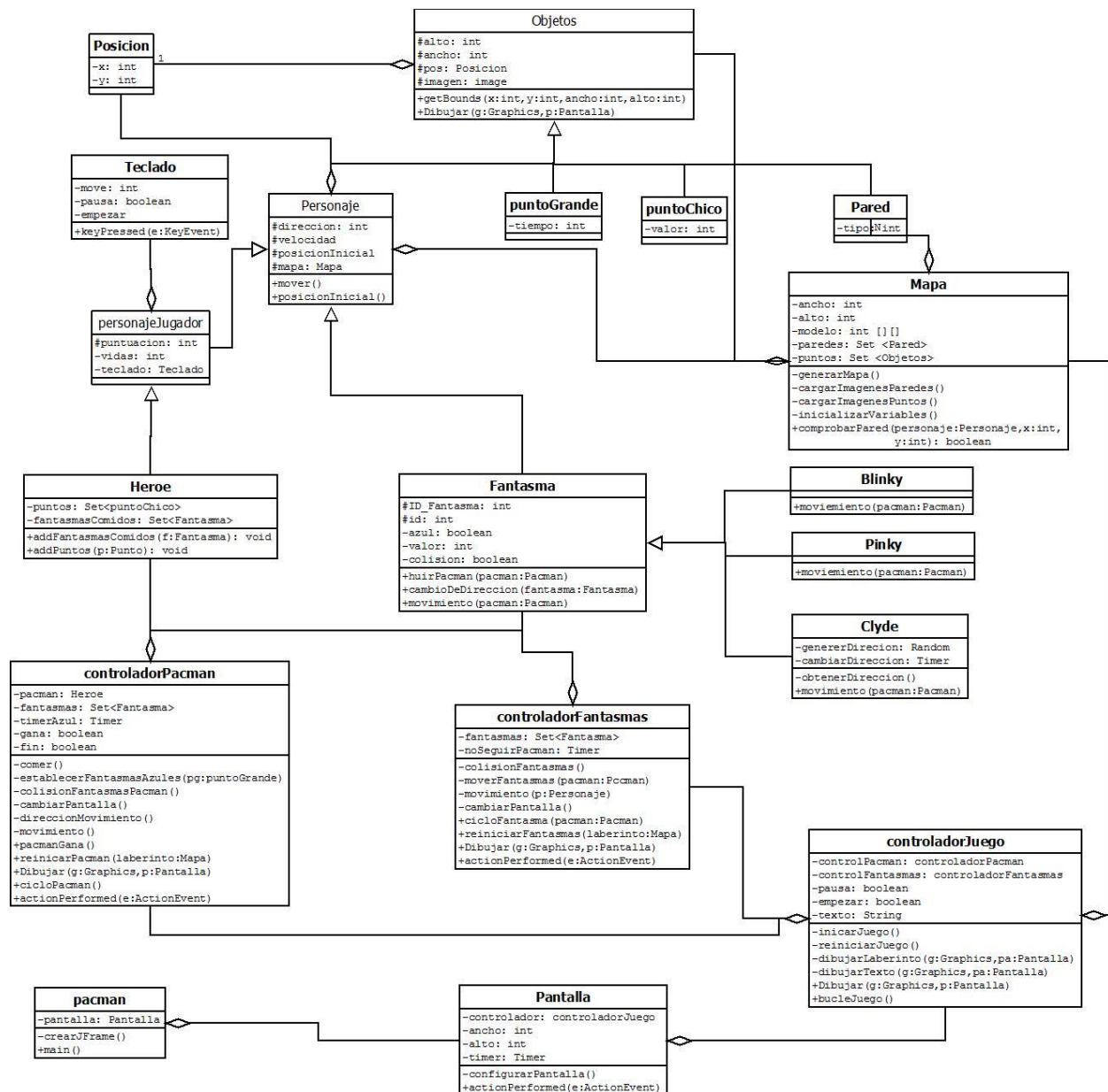
Diseño de Clases

1.1. Identificación y refinamiento de las clases partiendo del diseño principal

Partiendo del diseño de la primera fase se hace un segundo análisis profundizando más en los requisitos y en el diseño de clases intentando identificar relaciones de herencia y si las clases existentes se pueden dividir en más clases así como las funcionalidades de cada clase. Con esto se busca dividir las funcionalidades en clases bien identificadas buscando el fácil mantenimiento y la reutilización de las mismas.

Esta fase se explica que realiza cada una de las clases (con sus campos y métodos) del diseño

Diagrama de Clases:



- Por espacio en el diagrama se omiten los métodos selectores y mutadores

Clases, campos y métodos:

- **Objetos:** de esta clase heredaran todas las subclases que se tengan que dibujar en el tablero de juego. Tiene las características básicas que tendrá cualquier objeto que se quiera dibujar en pantalla
 - **Campos**

- **# Ancho:** ancho del objeto
 - **# Alto:** alto del objeto
 - **# imagen:** imagen del objeto
 - **# posicion:** posición del objeto en la pantalla
- **Métodos:**
 - **+getBounds(int x, int y, int ancho, int alto)** : Devuelve el el objeto como rectángulo. Se hará uso de este método cuando se necesiten comprobar colisiones
 - **+Dibujar(Graphics g, Pantalla p):** dibuja el objeto en pantalla(estè método es llamado desde el controlador del juego)
 - **+getAncho():** Devuelve el ancho que ocupa un objeto
 - **+getAlto():** Devuelve el alto que ocupa un objeto
- **Posición:** como los objetos que se mostraran en pantalla tendrán una posición en la misma, esta clase contendrá las coordenadas x e y de la pantalla.
 - **Campos**
 - - **X:** coordenada x
 - - **Y:** coordenada y
 - **Metodos:**
 - **+ getY()** : Devuelve la coordenada y
 - **+ SetY(int y):** modifica la coordenda y
 - **+ getX()** : Devuelve la coordenada x
 - **+ SetX(int x):** modifica la coordenada y
- **Personaje:** los personajes del juego (fantasmas y pacman) tiene unas propiedades similares que son: posición, velocidad, posición inicial y Mapa. También tienen un comportamiento similar que es mover. Por tanto la clase pacman y fantasmas pueden heredaran de esta. Esta clase al ser un objeto que se va dibujar en la pantalla heredara de la clase Objetos
 - **Campos**
 - **#dirección:** código de dirección en la que se mueve un personaje (izquierda, derecha, arriba, abajo)
 - **#velocidad:** velocidad a la que se mueve un personaje
 - **#posicionInicial:** posición desde la que empieza en el juego
 - **#mapa:** mapa por el que se mueve el jugador
 - **Métodos:**
 - **#comprobarLimites():** comprueba que los personajes no superen el límite del mapa
 - **+posicionInicial():** hace que el personaje vuelva a su posición inicial
 - **+Mover():** hace que se mueva el personaje
 - **+ getDireccion():** obtiene el código de dirección en la que se mueve el personaje
 - **+ setDireccion(int direccion)** : modifica el código de dirección en la que se mueve el personaje
 - **+ getVelocidad():** devuelve la velocidad de movimiento del personaje

- **+ setVelocidad(int velocidad):** Modifica la velocidad de movimiento de un personaje
 - **+ getPosicionInicial():** devuelve la posición inicial del personaje
 - **+ setPosicionInicial(Posicion posicionInicial):** modifica la posición inicial del personaje

 - **+ getMapa():** Devuelve el mapa por el que se mueve el personaje
 - **+ setMapa(Mapa mapa) :** Modifica el mapa por el que se mueve el personaje

- **personajeJugador:** como un jugador controlara a pacman se crea esta clase que contiene características básicas que tiene un personaje controlado por una persona, estos son: dispositivo que permita mover al personaje (teclado), puntuación del jugador, vidas de las que dispone el jugador. Esta clase heredara de la clase personaje
 - **Campos**
 - **puntuacion:** puntuación del jugador
 - **vidas:** vidas de las que dispone el jugador
 - **teclado:** que implementa los controles del teclado que utilizara el jugador
 - **Metodos:**
 - **+ getPuntuacion():** Devuelve la puntuación del personaje que controla una persona. Este método es redefinido en la clase Heroe
 - **+ setPuntuacion(int puntuacion):** modifica la puntuación del personaje que controla una persona
 - **+ getTeclado():** Devuelve el componente teclado que utiliza el jugador para controlar el juego
 - **+ getVidas():** Devuelve las vidas del personaje
 - **+ setVidas(int puntuacion):** modifica las vidas del personaje

- **Heroe:** Esta clase es la que un una primera instancia se nombró como pacman, se cambia el nombre ya que la clase principal de la aplicación tiene que tener ese nombre. Esta clase es la que tendrá las características y actualizara la posicion de este. Hereda de la clase personajeJugador. Esta clase es la que definirá a pacman.
 - **Campos**
 - **- Set<Puntos> puntos:** lista con los puntos comidos por pacman
 - **- set <fantasmas> fantasmasComidos :** lista de fantasmas azules comidos por pacman
 - **Métodos:**
 - **+ addFantasmasComidos():** añade fantasmas comidos a la lista de fantasmas
 - **+ addPuntos():** añade puntos a la lista de puntos comidos
 - **+ Mover():** actualiza la posición de pacman en el mapa
 - **+ getPuntos() :** Devuelve la lista de los puntos comidos por pacman

- **+ getFantasmasComidos():** Devuelve la lista de los fantasmas azules comidos por pacman
- **Fantasma:** Esta clase es la que tendrá las características y movimientos básicos de los fantasmas. Hereda de la clase personaje. Esta clase es la que definirá un fantasma general, los distintos tipos de fantasmas heredaran de esta clase.
 - **Campos**
 - **# ID_Fantasma:** Este es un campo static que se utiliza como contador para generar los ids de los fantasmas
 - **# Id:** id de un fantasma
 - **- azul:** para establecer si el fantasma es azul o no
 - **- colision :** para indicar si un fantasma ha colisionado con otro
 - **- valor:** Valor en puntos de un fantasma
 - **Métodos:**
 - **+huirPacman():** hace que el fantasma escape de pacman
 - **+cambioDeDireccion():** hace que el fantasma cambie de dirección
 - **+ movimiento():** Este método es el que establece el movimiento característico de cada fantasma, se redefine en las subclases que hereden de fantasma para dar un movimiento específico a cada uno
 - **+ Mover():** actualiza la posición del fantasma en el mapa
 - **+ isAzul() :** Metodo que indica si un fantasma es azul o no
 - **+setAzul(boolean azul):** método que cambia al estado azul o normal a un fantasma
 - **+ equals (Fantasma fantasma):** Indica si 2 fantasmas son el mismo
 - **+ getId() :** devuelve el identificador de un fantasma
 - **+ getValor():** devuelve el valor en puntos de un fantasma
 - **+ isColision():** indica si un fantasma colisiona o no con otro fantasma
 - **+ setColision(boolean colision):** cambia el estado de colisión de un fantasma
- **Blinky:** Clase del fantasma Blinky, este fantasma sigue a pacman por el laberinto intentando colisionar con el primero verticalmente y luego horizontalmente. Extiende a la clase Fantasma.
 - **Métodos:**
 - **+ movimiento (Heroe pacman):** Implementación del movimiento característico del fantasma blinky. Método que el código de dirección de movimiento , para que intente colisionar con pacman primero lo busca verticalmente y luego horizontalmente
- **Pinky:** Clase del fantasma Pinky, este fantasma sigue a pacman por el laberinto intentando colisionar con el primero, horizontalmente y luego verticalmente. Extiende a la clase Fantasma.

- **Métodos:**
 - **+ movimiento (Heroe pacman):** Implementación del movimiento característico del fantasma pinky. Método que asigna el código de dirección de movimiento , para que intente colisionar con pacman primero horizontalmente y luego verticalmente
- **Clyde:** Clase del fantasma Clyde, este fantasma no sigue a pacman por el laberinto si no que deambula por el sin rumbo, cambia su dirección aleatoriamente cada cierto tiempo. Extiende a la clase Fantasma.
 - **Campos:**
 - - **genererDireccion:** generador de nuemeros aleatorios
 - - **cambiarDireccion:** timer que indica cada que tiempo cambia de dirección
 - - **direccionAleatoria:** contiene un código de dirección generado aleatoriamente
 - **Métodos:**
 - - **obtenerDireccion():** genera una dirección aleatoria
 - **+ movimiento (Heroe pacman):** Implementación del movimiento característico del fantasma pinky. Método que asigna la dirección de movimiento , para que intente colisionar con pacman primero horizontalmente y luego verticalmente
- **Teclado:** Clase que implementa los controles del teclado y da funcionalidad a las teclas del juego
 - **Campos**
 - - **Pausa:** indica que se pulso la tecla de pausa
 - - **Empezar:** indica que se pulso la tecla empezar
 - - **move:** guarda el código de dirección según la tecla pulsada
 - **Metodos:**
 - **+ getMove():** Devuelve el código de dirección de movimiento. Esta dirección se utiliza para modificar la dirección del jugador
 - **+ setMove(int move):** modifica el código de dirección del movimiento
 - **+ keyPressed(KeyEvent e):** Método que se ejecutara al pulsar una tecla, evaluara si la tecla pulsada es alguna de las permitidas en el juego. Si las teclas pulsadas son las flechas de dirección, se modificara el código de dirección de movimiento.
 - **+ isPausa() :** indica si el juego debe ponerse en pausa o no
 - **+ setPausa(boolean pausa):** modifica la variable pausa , que indica si el juego debe o no estar en pausa
 - **+ isEmpezar() :** indica si el juego debe empezar o no
 - **+ setEmpezar(boolean empezar):** modifica la variable empezar , que indica si el juego debe o no empezar

- **Mapa:** Clase que genera el laberinto por donde se moverá pacman y los fantasmas, se encargara de crear y contener los objetos dibujables del laberinto
 - **Campos**
 - - **Set<Puntos> puntos:** lista de puntos del laberinto
 - - **Set<Pared> paredes:** lista de paredes del laberinto
 - - **Modelo[][]:** modelo del mapa del cual se generara el mapa de juego
 - - **ancho:** ancho del mapa
 - - **alto:** alto del mapa
 - **Métodos:**
 - - **generarMapa():** se encarga de crear el laberinto con paredes y puntos
 - - **+comprobarPared(Personaje personaje, int x, int y) :** comprueba si el personaje pasado como parametro colisiona con una pared
 - + **getParedes():** Devuelve la lista paredes que contiene el laberinto
 - + **getPuntos():** Devuelve la lista de puntos que contiene el laberinto
 - + **getAncho():** Devuelve el ancho que ocupa el mapa
 - + **getAlto():** Devuelve el alto que ocupa el mapa
- **puntoChico:** Esta clase hereda de la clase Objeto y define las características de un punto chico del laberinto. Como la clase padre punto solo tiene un campo posición se decide no incluirla porque la clase Objetos ya define ese campo:
 - **Campos:**
 - - **valor:** valor en puntos de un puntoChico (un punto chico comido vale 10 puntos)
 - **Métodos:**
 - + **getValor():** Devuelve el valor del punto
- **puntoGrande:** Esta clase hereda de la clase Objeto y define las características de un punto grande del laberinto
 - **Campos:**
 - - **tiempo :** tiempo que dura la habilidad de un punto grande (un punto grande comido por pacman permite que los fantasmas sean azules durante 5 segundos)
 - **Métodos:**
 - + **getTiempo():** Devuelve el tiempo que dura la habilidad de convertir a los fantasmas en azules
 - + **SetTiempo(int tiempo):** modifica el tiempo que dura la habilidad de convertir a los fantasmas en azul
- **Pared:** Esta clase se encarga de crear una pared del laberinto y establecer su posición en el mismo. También indica el tipo de pared que es (esquina inferior, superior, etc.) mediante un código. Esta clase Hereda de la clase Objetos
 - **Campos**
 - - **tipo:** código que indica el tipo de pared (esquina inferior, superior, etc.).
 - **Métodos:**

- + **getTipo()**: Devuelve el tipo de pared
- **Pared**: Esta clase se encarga de crear una pared del laberinto y establecer su posición en el mismo. También indica el tipo de pared que es (esquina inferior, superior, etc.) mediante un código. Esta clase Hereda de la clase Objetos
 - **Campos**
 - - **tipo**: código que indica el tipo de pared (esquina inferior, superior, etc.).
 - **Métodos**:
 - + **getTipo()**: Devuelve el tipo de pared
- **controladorJuego**: Clase que controla el bucle del juego, y mantiene actualizada la parte del modelo a través de los controladores de pacman y l fantasmas, y a la vista mediante el método dibujar. Esta clase mantiene actualizada la vista y el modelo
 - **Campos**:
 - - **controlPacman**: controlador de pacman
 - - **controlFantasmas**: controlador fantasmas
 - - **empezar**: indica si empieza el juego
 - - **pausa**: indica si el juego está en pausa
 - - **texto**: texto que se muestra en pantalla tanto al iniciar la partida como cuando termina
 - **Metodos**:
 - - **iniciarJuego()**: inicia los controladores del juego para que inicialicen los componentes del juego
 - - **reiniciarJuego()**:reinicia los controladores del juego para que reinicien los componentes del juego
 - - **dibujarLaberinto(Graphics g,Pantalla pa)**: Dibuja el laberinto
 - - **dibujarTexto(Graphics g,Pantalla p)**: Dibuja la parte de texto del juego
 - + **bucleJuego()** : Método que contiene las acciones que debe realizar el juego en cada iteración. Este método se ejecutara en cada golpe de timer en la clase pantalla
 - + **getControlPacman()** : devuelve el controlador de pacman
 - + **getControlFantasmas()**: Devuelve el controlador de los fantasmas
 - + **Dibujar(Graphics g,Pantalla p)** : llama al método dibujar que llamara al método dibujar de los controladores de pacman y los fantasmas
 -
- **controladorPacman**: Clase que controla las acciones y actualiza el estado de pacman durante el juego. Esta clase es utilizada desde la clase controladorJuego.
 - **Campos**:
 - - **pacman**: pacma que mueve el jugador
 - - **fantasmas**: lista de fantasmas que mueve el ordenador
 - - **timerAzul**: timer con el tiempo que duran los fantasmas en modo azul
 - - **gana**: indica si el jugador gana o no la partida
 - - **fin**: indica si es el fin de la partida
 - **Metodos**:

- - **comer()**: realiza la función de pacman de comer puntos según si este colisiona con algún punto
 - - **establecerFantasmasAzules(puntoGrande pg)**: pone a los fantasmas en el modo azul
 - - **colisionFantasmasPacman()**: comprueba si pacman colisiona con algún fantasma
 - - **direccionMovimiento()**: Asigna la dirección de movimiento a pacman según la tecla pulsada por el jugador
 - - **movimiento()**: Actualiza la posición de pacman
 - + **pacmanGana()**: establece si pacman gana o no la partida
 - + **cicloPacman()**: acciones que realiza el controlador por cada iteración del bucle del juego para actualizar el estado de pacman
 - + **reiniciarPacman(Mapa laberinto)**: vuelve a inicializar los valores de pacman al estado inicial
 - + **Dibujar(Graphics g, Pantalla p)** : llama al método dibujar del objeto Heroe
 - + **getPacman()**: Devuelve un objeto Heroe
 - + **isGana()**: indica si pacman gana o no el juego
 - + **setGana(boolean gana)**: modifica la variable gana que establece si pacman gana o no la partida
 - + **isFin()**: indica si es el fin de la partida o no (dependiendo de si pacman colisiona con algún fantasma o no)
 - + **setFin(boolean fin)**: modifica la variable fin que establece si es el fin o no de la partida
 - + **actionPerformed(ActionEvent e)**: este método se ejecuta cuando termine el tiempo del modo azul de los fantasmas devolviendo a los fantasmas a su modo normal y estableciendo la velocidad de movimiento de pacman a 1
- **controladorFantasmas**: Clase que controla las acciones y actualiza el estado de los fantasmas durante el juego. Esta clase es utilizada desde la clase controladorJuego. La colisión entre fantasmas realiza un cambio de dirección entre fantasmas durante un periodo un tiempo determinado
 - **Campos**:
 - - **fantasmas**: lista de fantasmas que mueve el ordenador
 - - **noSeguirPacman**: timer que indica el tiempo de duración de un cambio de dirección cuando colisionan 2 fantasmas
 - **Metodos**:
 - - **colisionFantasmas()**: comprueba si colisionan los fantasmas y establece el comportamiento según colisionen o no
 - - **moverFantasmas(Heroe pacman)**: modifica el código de dirección de los fantasmas según el comportamiento de cada uno en el momento de llamar al método
 - - **movimiento(Personaje p)**: Actualiza la posición del personaje pasado como parámetro si no choca con una pared
 - + **cicloFantasma(Heroe pacman)** : acciones que realiza el controlador por cada iteración del bucle del juego para actualizar el estado de los fantasmas

- + **reiniciarFantasmas(Mapa laberinto)** : vuelve a inicializar los valores de los fantasmas al estado inicial
 - + **Dibujar(Graphics g,Pantalla p)** : llama al método dibujar del objeto fantasma
 - + **getFantasmas()**: Devuelve la lista de fantasmas del juego
 - + **addFantasma(Fantasma fantasma)** : añade fantasmas a la lista de fantasmas del juego
 - + **actionPerformed(ActionEvent e)**: este método se ejecuta cuando termina el tiempo que dura el cambio de dirección de fantasmas cuando estos colisionan
- **Pantalla:** clase que contiene el panel donde se dibujara los componentes del juego y se ejecutara el bucle del juego
 - **Campos:**
 - - **controlador:** controlador del juego
 - - **ancho:** ancho de la pantalla
 - - **alto:** alto de la pantalla
 - - **timer:** timer que se llamara para ejecutar el bucle del juego y actualizar la
 - **Métodos:**
 - - **configurarPantalla()**: configura la pantalla de juego
 - + **paint(Graphics g)** : redefinido para dibujar los distintos componentes del juego
 - + **actionPerformed(ActionEvent e)**:método que se ejecuta con cada golpe de timer y que llama al método bucleJuego del controlador y repintara la pantalla
 - + **getAncho()**: Devuelve el ancho de la pantalla
 - + **getAlto()**: Devuelve el alto de la pantalla
 - + **setAncho(int ancho)**: modifica el ancho de la pantalla
 - + **setAlto(int alto)**: modifica el alto de la pantalla
 -
- **pacman:** Clase que crea un JFrame que contiene el tablero del juego y que contiene el método main
 - **Campos:**
 - - **pantalla:** pantalla del juego
 - **Metodos:**
 - - **crearJFrame()**: crea el jframe que contendrá la pantalla de juego
 - + **main (String [] args)**: método min para iniciar el juego, crea una instancia de la clase pacman.

Estrategias implementadas y decisiones de diseño establecidas

Relaciones de herencia

Se intenta buscar desde el diseño hacer uso de los mecanismos de herencia y poder diseñar clases reutilizables y fáciles de mantener.

Las clases Objetos, Personje y personajeJugador, se diseñaron desde el punto de vista de que también puedan utilizarse para otros proyectos de tipo video juego 2d en los que un personaje tenga que moverse por un mapa ya que estos tendrán las mismas características. Estas 3 clases se deciden crear como abstractas ya que no se crearan instancias de las mismas.

La clase que tengan que dibujarse en pantalla heredan de la clase objetos ya que esta tiene los campos y métodos para dicha función. Las clases que heredan de la clase objeto y que serán dibujables en la pantalla son: Personaje, Pared, puntoChico, puntoGrande

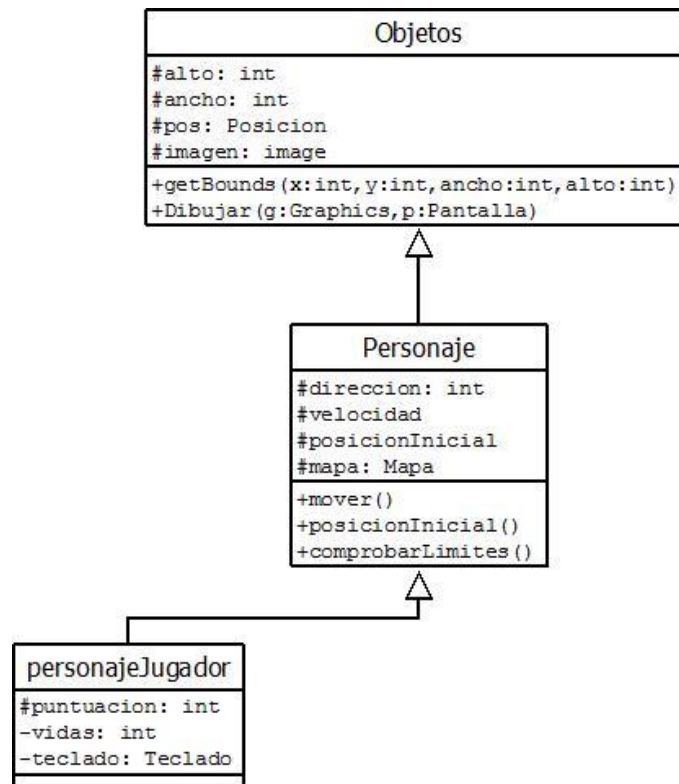
Para Cualquier personaje de un juego de estas características pude crearse una clase que herede de la clase Personaje ya que define las funciones básicas (campos y métodos) de un personaje para que pueda moverse por un mapa y dibujarse en pantalla

Para un personaje que controle un usuario por medio del teclado se podría heredar de la clase personajeJugador porque esta clase contiene aparte de toda la funcionalidad de un personaje (hereda de esta clase) campos y métodos para mostrar la puntuación de un jugador , las vidas , y el teclado para jugar.

También se podría implementar herencia para la clase mapa ya que ahora la clase mapa esta implementada con el laberinto de pacman y las clases que hereden de personaje siempre tendrán que utilizar como mapa el laberinto de pacman, esto se pude solucionar haciendo que la clase mapa sea una superclase con las características y funcionalidades básicas de cualquier mapa y clases que heredan de mapa solo tendrían añadir las propiedades y funcionalidades específicas de un mapa en concreto. De esta forma se solucionaría el problema de que las clases que hereden de Personaje se limiten a utilizar el laberinto de pacman. **Esto se deja como una mejora.**

Para la clase teclado se podría hacer lo mismo que con la clase mapa ya que ahora está implementado para usar el teclado y para unas teclas concretas. **Esto se deja como una mejora.**

Con estas 2 mejoras las clases Personje y personajeJugador serían completamente reutilizables al no depender de un mapa específico y de un teclado con una configuración específica.





La clase Heroe hereda de la clase personajeJugador. Al utilizar el mecanismo de Herencia en esta clase solo hizo falta añadir las características y funcionalidades básicas para el personaje pacman.


La clase Fantasma hereda de la clase Personaje. Al utilizar el mecanismo de Herencia en esta clase solo hizo falta añadir las características y funcionalidades básicas para los fantasmas en general, en donde cada fantasma específico hereda de esta clase y solo hizo falta añadir el movimiento específico de cada fantasma.

Estrategias implementadas

Creación del laberinto

Para la creación del laberinto se utiliza un modelo de tabla (se implementa mediante una matriz) de 22 filas x 21 columnas de 30 x 30 pixeles, donde cada par columna-fila lleva un código para identifica paredes, puntos chicos y puntos grandes. Se asignan números negativos para las paredes, el número 1 para puntos chicos y el número 2 para puntos grandes. Este modelo indica en que la posición donde debe ir ubicada cada uno de los objetos del laberinto

Código	Tipo de objeto del laberinto	Imagen
números < 0	Paredes	
1	Punto chico	

2	Punto Grande	
---	--------------	---

Ejemplo del modelo del laberinto:

-1	-5	-5	-5	-5	-5	-5	-5	-5	-5	-8	-5	-5	-5	-5	-5	-5	-5	-5	-2
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

- Aquí solo muestra una sola fila , pero el modelo completo sigue este patrón para 22 filas y 21 columnas

El laberinto contiene una lista de paredes y otra lista de puntos, estas lista se llenan recorriendo el array con el modelo del laberinto y dependiendo del código se almacena en un lista u otra (si código < 0 se almacena en la lista de paredes, si es código >0 se almacena en la lista de puntos). De esta forma cuando tanto pacman como los fantasmas tengan que comprobar las colisiones con las paredes solo tendrán que utilizar esta lista de paredes.

Para mostrar el laberinto en pantalla, se recorren las lista de paredes y puntos y se imprimen en pantalla según su posición (cada pared y punto indica su posición y su imagen)

Movimiento de los personajes

Pacman

El movimiento de pacman se basó en el movimiento del juego original: se elige una dirección y pacman se mueve en esa dirección mientras no choque con ninguna pared, también se reconoce el próximo movimiento de pacman, por ejemplo si el jugador mueve a pacman a la derecha y decide mover arriba pero no puede porque hay una pared, se guarda en memoria el movimiento arriba para que cuando no haya pared este cambie de dirección.

Cuando se quiera mover a pacman se llamara al método mover para actualizar la posicion de pacman, este a su vez llamara la metodo direccionMovimiento() para asignar el codigo direccion de movimiento

Se utilizan 1 variable, una que indica la dirección en la que se está moviendo pacman (dirección, que se encuentra en la clase Personaje porque clase que define a pacman hereda de esta)

Se utiliza un código de dirección para indicar si el personaje se mueve a la izquierda, a la derecha, abajo o arriba. Este código es.

1=izquierda, 2=derecha ,3=arriba,4=abajo

Cada vez que el jugador pulse una tecla el controlador recogerá el valor de la tecla pulsada (un codigo de direccion) y se asignara el código de dirección a pacman siempre que en el momento de pulsar la tecla no choque con ninguna pared.

El desplazamiento de pacman se hace en función de la velocidad, lo que se hace es sumar o restar a las coordenadas x e y la velocidad dependiendo de hacia donde se mueva pacman.

Fantasmas

El movimiento de los fantasmas sigue un patrón similar al de pacman, es decir, se asigna un código de dirección que indica a donde se mueve el fantasma. Este código es:

1=izquierda, 2=derecha, 3=arriba, 4=abajo

Cuando se quiera mover a un fantasma se llama al método mover(), el método actualiza la posición del fantasma según el código de dirección.

Los fantasmas tienen tipos de movimientos generales que hacen que cambien de dirección cuando colisionan entre ellos y que huyan de pacman cuando están en modo azul. Luego cada fantasma en particular tiene su propio movimiento característico. Estos métodos son los que asignan el código de dirección correspondiente para poder actualizar su posición

Movimientos generales:

Movimiento huir de pacman.

Este movimiento se implementa en un método llamado huirPacman(Heroe pacman) en la clase fantasma y lo que hace es asignar un código de dirección dependiendo de donde este pacman.

El método huirPacman(Heroe pacman) se expresa de la siguiente forma:

huirPacman(Heroe pacman) {

 Si (coordenada x fantasma < coordenada x pacman){

 Si (si no hay pared en direccion = 1) {

 direccion = 1

 }

 }

 Si (coordenada x fantasma > coordenada x pacman) {

 Si (si no hay pared en direccion = 2) {

 direccion = 2

 }

 }

 // Se hace la misma comprobación para la coordenada y del fantasma

}

Movimiento colisión de fantasmas.

Este movimiento se implementa en el método `cambioDeDireccion(Fantasma fantasma)` de la clase `fantasma` y lo que hace es asignar un código de dirección dependiendo en donde este el fantasma recibido como parámetro. El método es similar al método `huirPacman(Heroe pacman)`.

Movimientos Especificos:

Cada fantasma implementa el metodo `movimiento(Heroe pacman)` , en este método se implementa el movimiento característico de cada fantasma. Lo que hace el método es asignar el código de dirección según su comportamiento

Blinky:

Buscará colisionar con pacman. Para acercarse a pacman calculará la distancia medido en filas y columnas e intentará primero acercarse verticalmente y luego horizontalmente.

El método movimiento para blinky se expresa:

```
movimiento(Heroe pacman) {  
    //primero intenta acercarse verticalmente  
    Si (coordenada y fantasma < coordenada y pacman ){  
        Si (si no hay pared en dirección = 4) {  
            direccion = 4 //se acerca verticalmente por arriba  
        }  
    }  
    Si (coordenada x fantasma > coordenada x pacman) {  
        Si (si no hay pared en dirección = 3) {  
            direccion = 3 // se acerca verticalmente por abajo  
        }  
    }  
    // Se hace la misma comprobación para la coordenada x  
}
```

Pinky:

Buscará colisionar con pacman. Para acercarse a pacman calculará la distancia medido en filas y columnas e intentará primero acercarse horizontalmente y luego verticalmente.

El metodo movimiento para pinky se expresa:

```

movimiento(Heroe pacman) {
    //primero intenta acercarse horizontalmente
    Si (coordenada x fantasma < coordenada x pacman ){
        Si (si no hay pared en direccion = 2) {
            direccion = 2 //se acerca horizontalmente por la derecha
        }
    }
    Si (coordenada x fantasma > coordenada x pacman) {
        Si (si no hay pared en direccion = 1) {
            direccion = 1 // se acerca horizontalmente por la izquierda
        }
    }
    // Se hace la misma comprobación para la coordenada y
}

```

Clyde:

No persigue a pacman, si no que deambula sin una ruta específica.

El método movimiento lo que hace es modificar la direccion de movimiento de clyde obtenida aleatoriamente por el metodo obtenerDireccion().

El método obtenerDireccion() genera una dirección aleatoria cada cierto tiempo

Controlador.

El controlador se podría haber implementado en una sola clase, pero se ha decidido dividir en 3 clases: una para las acciones propias del juego, otras para las acciones de pacman y otra para los fantasmas. Se decidió hacerlo de esta forma para dividir las clases con tareas específicas.

Controlador Juego:

Contiene el bucle del juego, este tiene todos los pasos que debe dar el juego por cada iteración del bucle.

Controlador Pacman:

Contiene las acciones que tiene que realizar pacman por cada iteración del bucle del juego así como de establecer los valores iniciales de pacman.

Básicamente en esta clase se comprueba si pacman come puntos del juego (se detecta colisión entre pacman y un punto), si pacman colisiona con algún fantasma, si pacman gana la partida (porque

comió todos los puntos) o si pacman pierde (colisiono con algún fantasma no azul.), establecer cuando los fantasmas deben convertirse en azules y el tiempo de duración de estos. También actualiza la posición de pacman y comprueba que no choque con ninguna pared

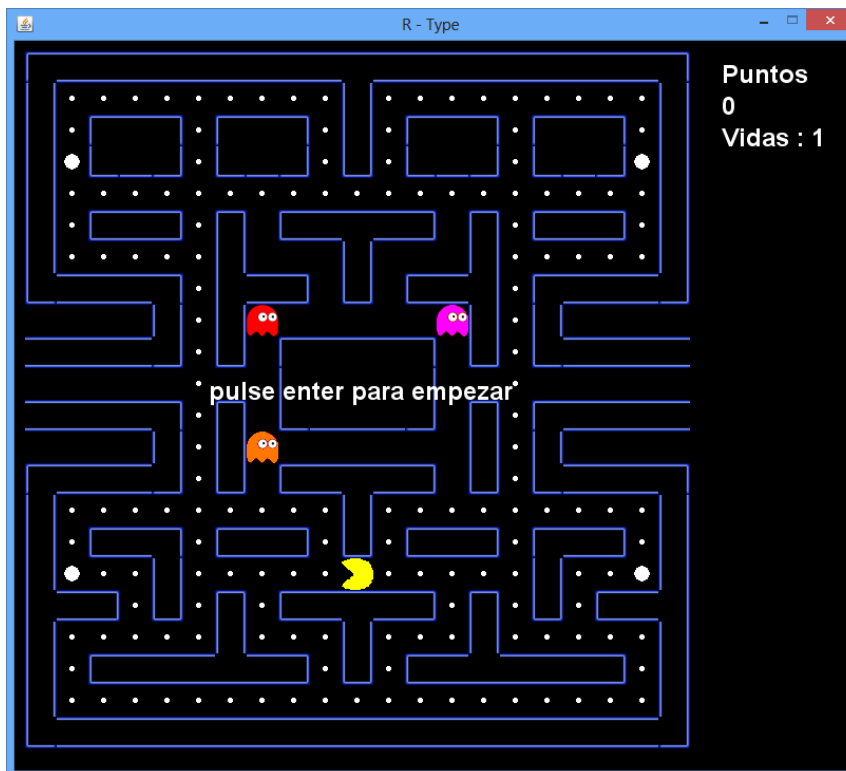
Controlador Fantasma:

Contiene las acciones que tienen que realizar los fantasmas por cada iteración del bucle del juego así como de establecer los valores iniciales de cada fantasma.

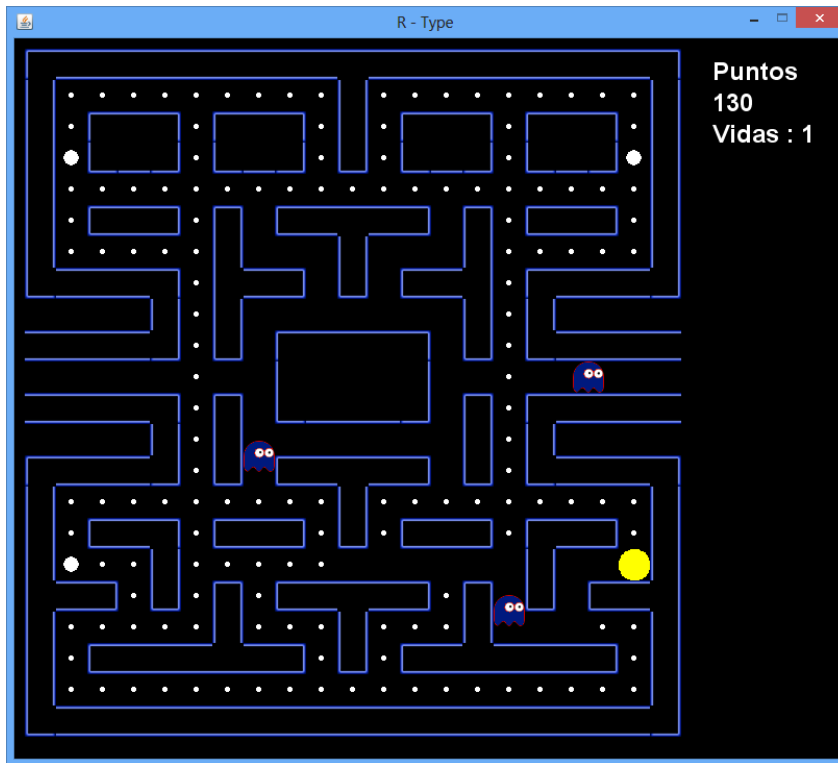
En esta clase se comprueba la colisión entre fantasmas. También actualiza la posición de cada uno de los fantasmas y establece el tiempo que dura el cambio de dirección cuando colisionan 2 fantasmas.

Imágenes del juego en funcionamiento

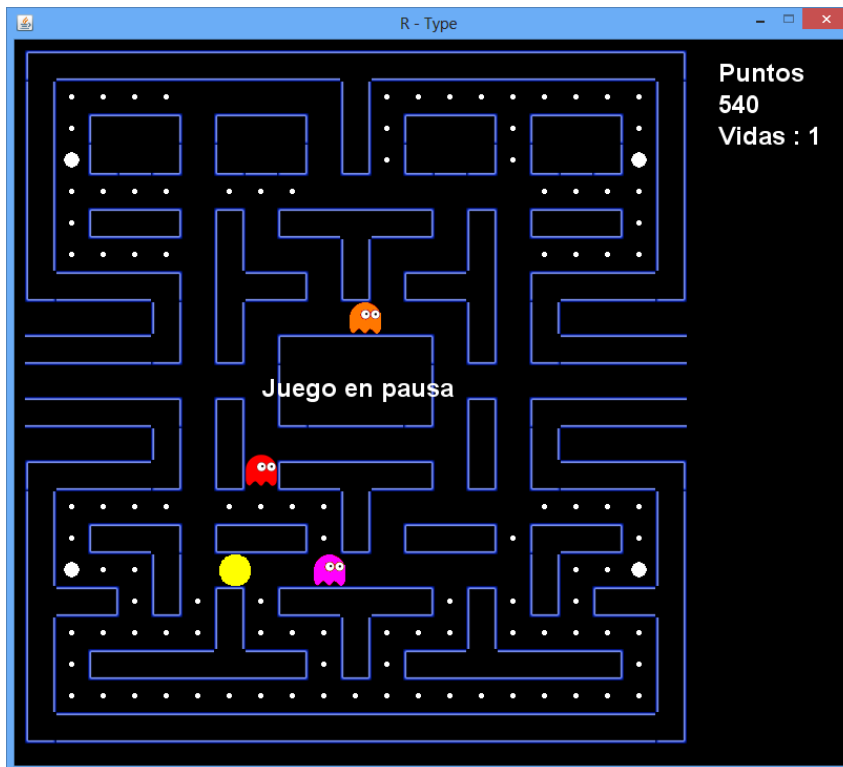
1. Pantalla del juego al iniciar



2. Fantasmas en modo azul



3. Juego en pausa



4. Juego terminado por colisionar con un fantasma



5. Juego ganado



Material a entregar

Dentro de la carpeta Parctica_POO se encuentra:

- El proyecto de la practica **Practica_Pacman_V1_BlueJ** con todos los ficheros *.java y *.class
- Dentro del proyecto en la carpeta **doc** se encuentra toda la documentación generada con javadoc
- Aplicación empaquetada en un archivo .jar listo para ejecutar : **pacman.jar**
- Archivo **Memoria.pdf**

La aplicación se desarrolló utilizando el entorno de desarrollo BlueJ y la versión JDK 7