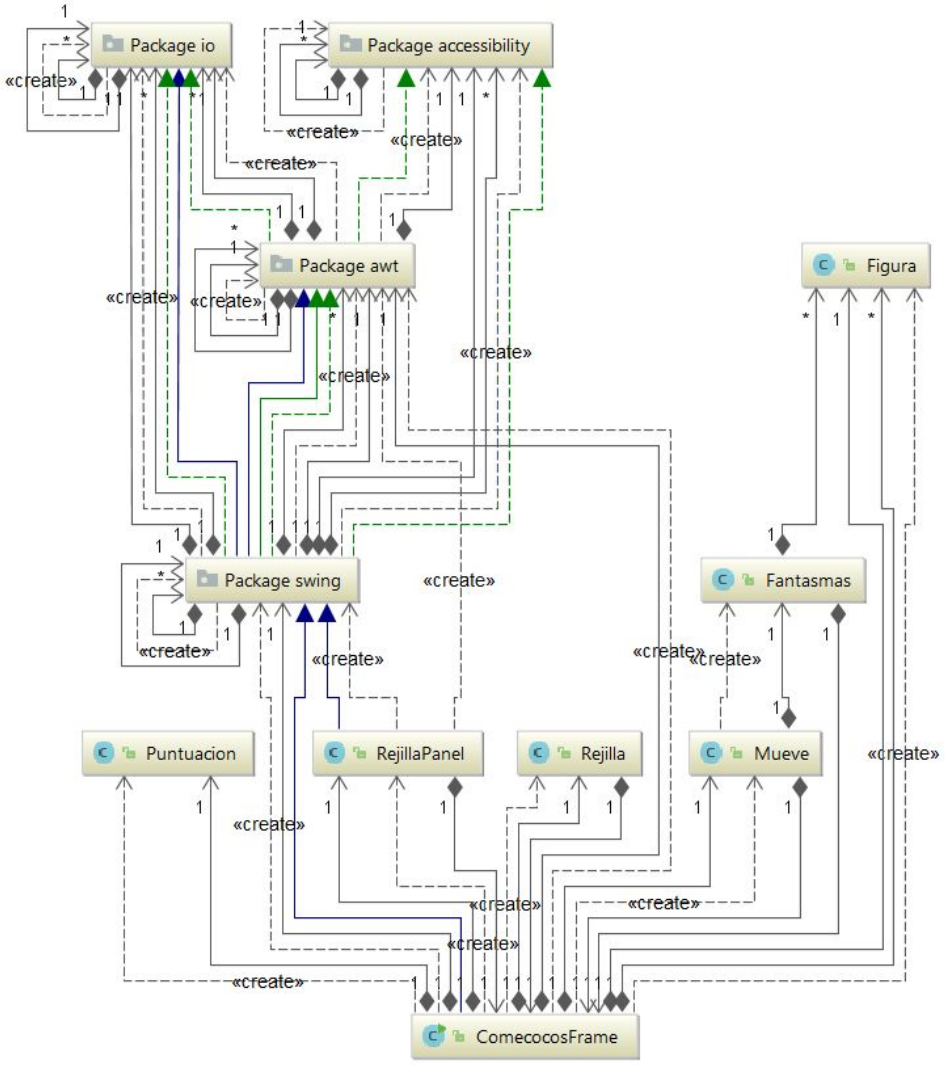
Complemento de Programación 2017/2018 José Antonio Hurtado Morón.
76065499Z

Práctica 8

Comecocos

Índice

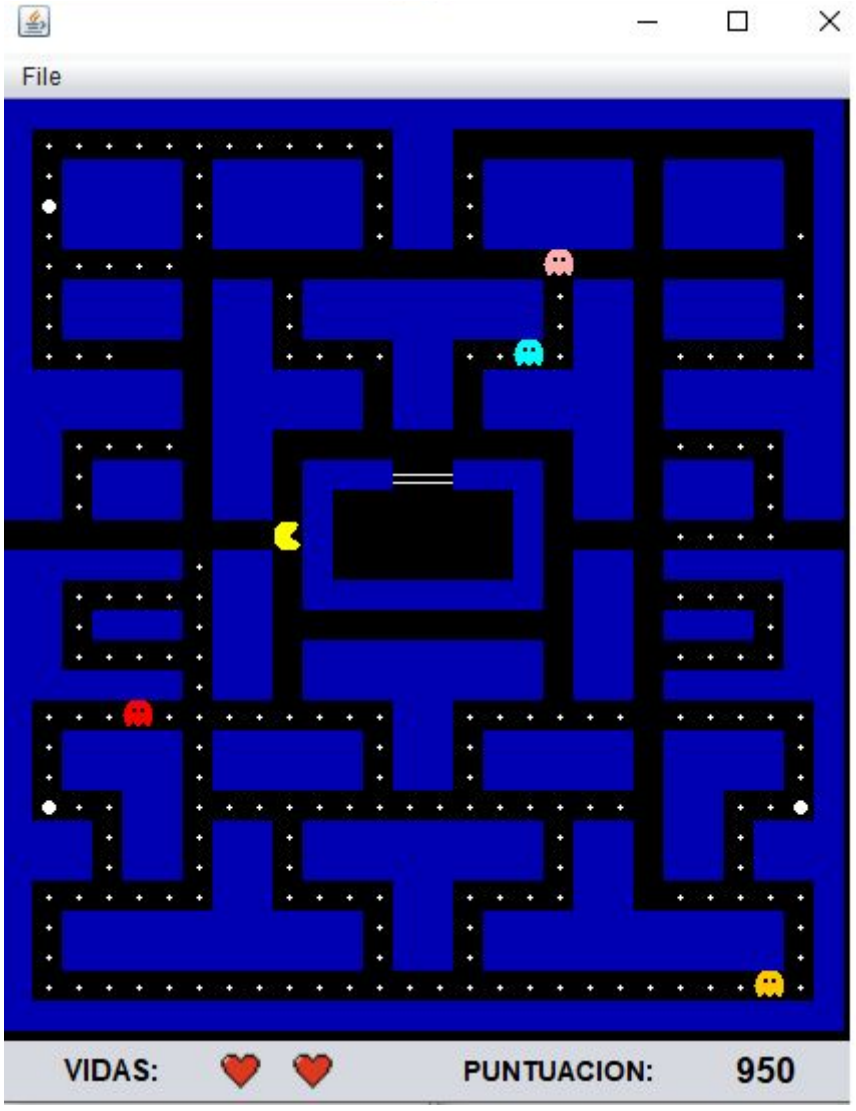
Diagrama UML



COMECOCOS

Esta es mi implementación de el juego del Comecocos para las prácticas de la asignatura de Complementos de programación. El juego esta hecho en Java, utilizando Swing para los componentes gráficos y una hebra para darle movimiento.

Esta es una captura de como ha quedado el resultado:



La clase principal del programa será ComecocosFrame, donde se inicializará el juego y se crearán referencias a el resto de clases. El esquema general del programa podría verse como una clase RejillaPanel que dibuja gráficamente cada uno de los componentes, y una clase principal, que ejecuta una Hebra infinita que implementará el movimiento, y que se encarga de actualizar tanto la lógica de los elementos del juego como los gráficos.

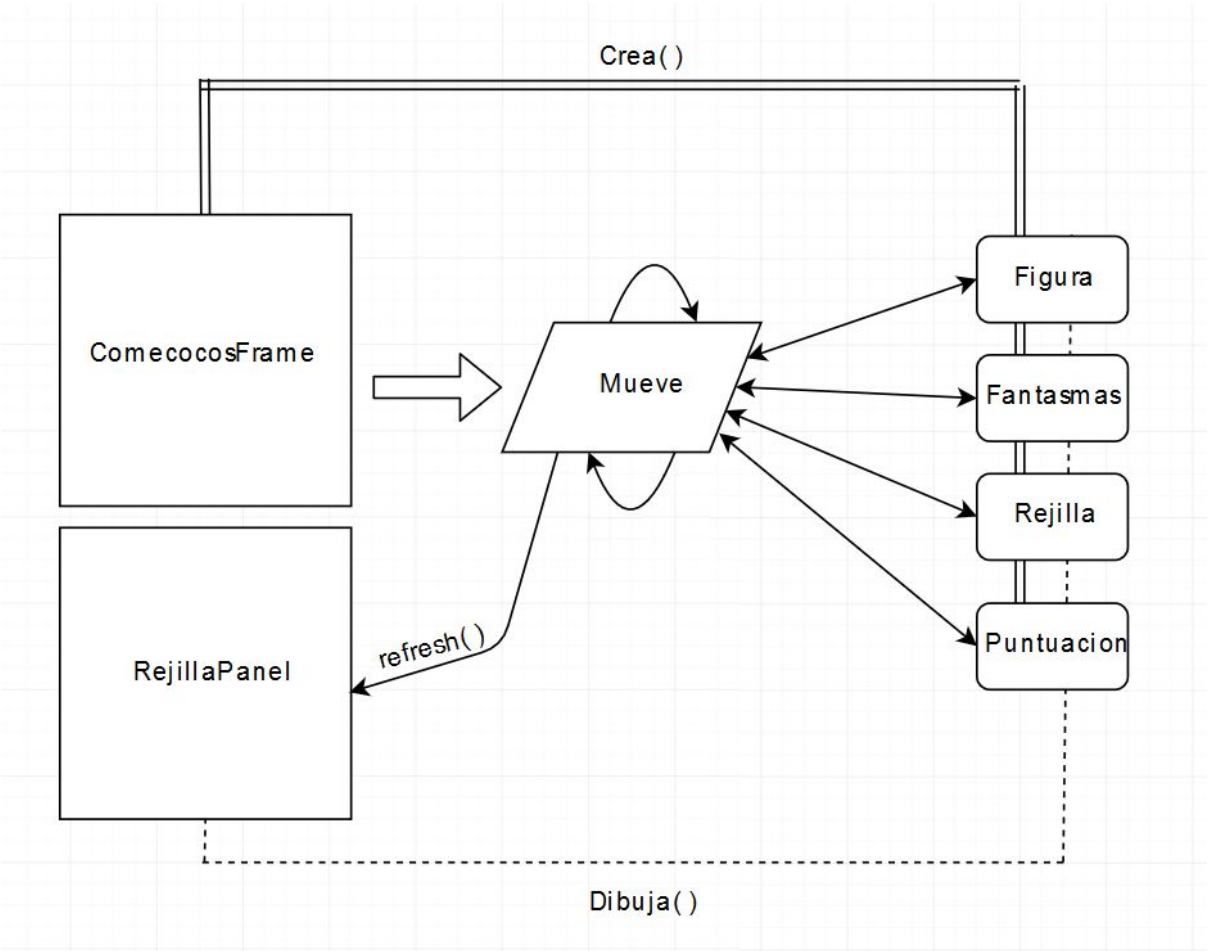


Diagrama aproximado de funcionamiento.

Clases

Comecocos Frame

Programa e interfaz principal del comecocos.Contiene todos los objetos y variables del programa.Sobre él se disponen el resto de componentes y gráficos.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	addPuntos(int p)
boolean	azules()
void	fantasmaMuerto(int i) Este metodo devuelve el fantasma a su posiciÃ³n inicial en el centro del mapa y da 200 puntos al usuario.
void	gestionaVidas()
Figura	getComecocos() GET figura comecocos.
Figura[]	getFantasmas() GET array de fantasmas.
RejillaPanel	getPanel() GET objeto rejillaPanel (graphics).
Rejilla	getRejilla() GET objeto rejilla.
void	init() En este metodo complementario al constructor tenemos las variables y objetos que hay que reiniciar cuando empecemos una nueva partida cuando subimos de nivel.
static void	main(java.lang.String[] args)
void	muerto() Metodo que reinicia la posiciÃ³n del comecocos cuando un fantasma se lo come.TambiÃ©n disminuye en uno el numero de vidas.
void	nextLevel()
void	playSound(int delay, java.lang.String name)
void	setAzules() Metodo que enciende el boolean azules durante 10 segundos, durante este tiempo todo el modo de juego cambiarÃi.
void	setDireccionFantasma(int i, int direccion)

Rejilla Panel

Dibuja todos los componentes del laberinto utilizando objetos de las clases Swing/AWT. Se encarga de dibujar gráficamente el mapa, el comecocos y los fantasmas, y además de el mapeo de las teclas.

Constructor Summary

Constructors

Constructor and Description
`RejillaPanel()`
`RejillaPanel(ComecocosFrame fr)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`dibujaComecocos(java.awt.Graphics g, Figura comecocos)` Dibuja el comecocos en 2 fases que se van alternando (boca abierta y boca cerrada)
void	`dibujaFantasma(java.awt.Graphics g, Figura[] fantasmas)` Dibuja todos los fantasmas tomando las mismas fases que el comecocos.
void	`dibujaRejilla(java.awt.Graphics g)` Dibuja el tablero de juego en funcion de la rejilla inicial definida en la clase Rejilla.
void	`drawIt(java.awt.Graphics g, int posicionX, int posicionY, int i)`
boolean	`getParpadeoComecocos()`
protected void	`paintComponent(java.awt.Graphics g)`
void	`setParpadeoComecocos()`
void	`setParpadeoFantasma(int i)`

Figura

Esta clase define una unidad elemental de nuestro juego, que serán las figuras. Cada figura contará con dos enteros i e j que definirán la posición de la figura, y un entero que definirá la dirección, donde usaremos el mapeo:

0 - arriba
1 - derecha
2 - abajo
3 - izquierda

Constructor Summary

Constructors

Constructor and Description
`Figura(int i, int j, int direccion)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`int`	`getDireccion()`
`int`	`getI()`
`int`	`getJ()`
`void`	`mueve(int direccion)`
`void`	`setDireccion(int direccion)`
`void`	`setI(int i)`
`void`	`setJ(int j)`

Rejilla

Clase que contiene el mapa. (Rejilla inicial en forma de array y la matriz de caracteres). También implementa ciertas clases que realizan comprobaciones o modificaciones en la matriz de la rejilla.

Constructor Summary

Constructors

Constructor and Description
`Rejilla(ComecocosFrame fr)` Constructor Principal

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`colision(Figura f, int direccion)` Clase que comprueba si una figura colisiona en su siguiente movimiento.
`void`	`comePunto(Figura f, int direccion)` Clase que comprueba si el comecocos se comerÃ¡ un punto en su siguiente movimiento
`int`	`getAltura()` Devuelve la cantidad de celdas de alto que posee el laberinto.
`int`	`getAnchura()` Devuelve la cantidad de celdas de ancho que posee el laberinto.
`char`	`getCelda(int i, int j)` Devuelve el tipo de la celda demarcada por i,j.
`boolean`	`nivelCompleto()` Clase que comprueba si ya se ha comido todos los puntos (nivel completado)
`void`	`reiniciar()`
`void`	`setCelda(int i, int j, char type)` Modifica el tipo de la celda i,j con el valor C.

Mueve

Clase que implementa una tarea Runnable que se ejecutará de forma continua. Se encarga de actualizar continuamente los gráficos y ejecutar el resto de métodos necesarios para la ejecución sincronizada del juego (cambiar la dirección de movimiento de las figuras, detectar colisiones tanto como con las paredes como con otras figuras, etc.)

Constructor Summary

Constructors

Constructor and Description
`Mueve(ComecocosFrame fr)` Constructor de la clase, que inicializa la referencia utilizadas por la hebra al ComecocosMidlet, establece el retardo en milisegundos entre movimiento y movimiento de la Figura actual, y comienza a ejecutar la hebra.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`getParado()` Nos dice si la hebra estÃ¡ o no parada.
`boolean`	`getTerminado()` Devuelve si la hebra ha terminado su ejecuciÃ³n
`void`	`nextLevel()`
`void`	`reanudar()` Reanuda el movimiento de la hebra.
`void`	`run()` CÃ³digo que constituye las sentencias de la tarea.
`void`	`suspender()` Detiene momentaneamente la ejecuciÃ³n de la hebra, haciendo que la Figura actual quede parada.
`void`	`terminar()` Termina la ejecuciÃ³n de la hebra.

Fantasmas

En esta clase vamos a implementar diferentes métodos relacionados con el movimiento de los fantasmas. Nos serÃ¡n de utilidad tanto para programar el algoritmo que decidirá el movimiento de los fantasmas, como para definir que ocurre cuando entramos en el modo "fantasmas comestibles".

Constructor Summary

Constructors

Constructor and Description
`Fantasmas(ComecocosFrame fr, Figura[] fantasmas)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`colisionComecocos(Figura comecocos)` Metodo que comprueba si algun fantasma se come al comecocos.
`int`	`getFantasmaPosicion(int i, int j)`
`int`	`movimientoFantasmaAleatorio(int index)` movimientoFantasmaAleatorio ********** Algoritmo que de forma aleatoria, utilizando la clase Random, devuelve la direcciÃ³n de movimiento de el fantasma {index}.
`int`	`movimientoFantasmaEuclideo(int index)` movimientoFantasmaAleatorio ********** Algoritmo que calcula la distancia euclidea desde cada una de las siguientes celdas a las que se puede mover el fantasma {index} hasta el comecocos, y elige aquella que: - Sea mas cercana al comecocos (modo normal) - Sea mas lejana al comecocos (caso fantasmas azules)

Puntuación

En esta clase vamos a implementar algunos métodos para llevar la cuenta de la puntuación de la partida.

Constructor Summary

Constructors
Constructor and Description
`Puntuacion()`

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	**Method and Description**	
`void`	`addPuntos(int puntuacion)`	
`int`	`getPuntuacion()`	

Sonido

En esta clase vamos a implementar una hebra para reproducir sonidos cuando se disparen mediante ciertos eventos. Esta clase solo realiza las funciones de reproducción, ya que el 'trigger' se hará en el codigo usando Timers.

Constructor Summary

Constructors
Constructor and Description
`Sonido(int tiempo, java.lang.String nombreSonido)` Este constructor permite operar con un sonido previamente "cargado" llamandolo mediante su nombre

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	**Method and Description**	
`void`	`run()`	