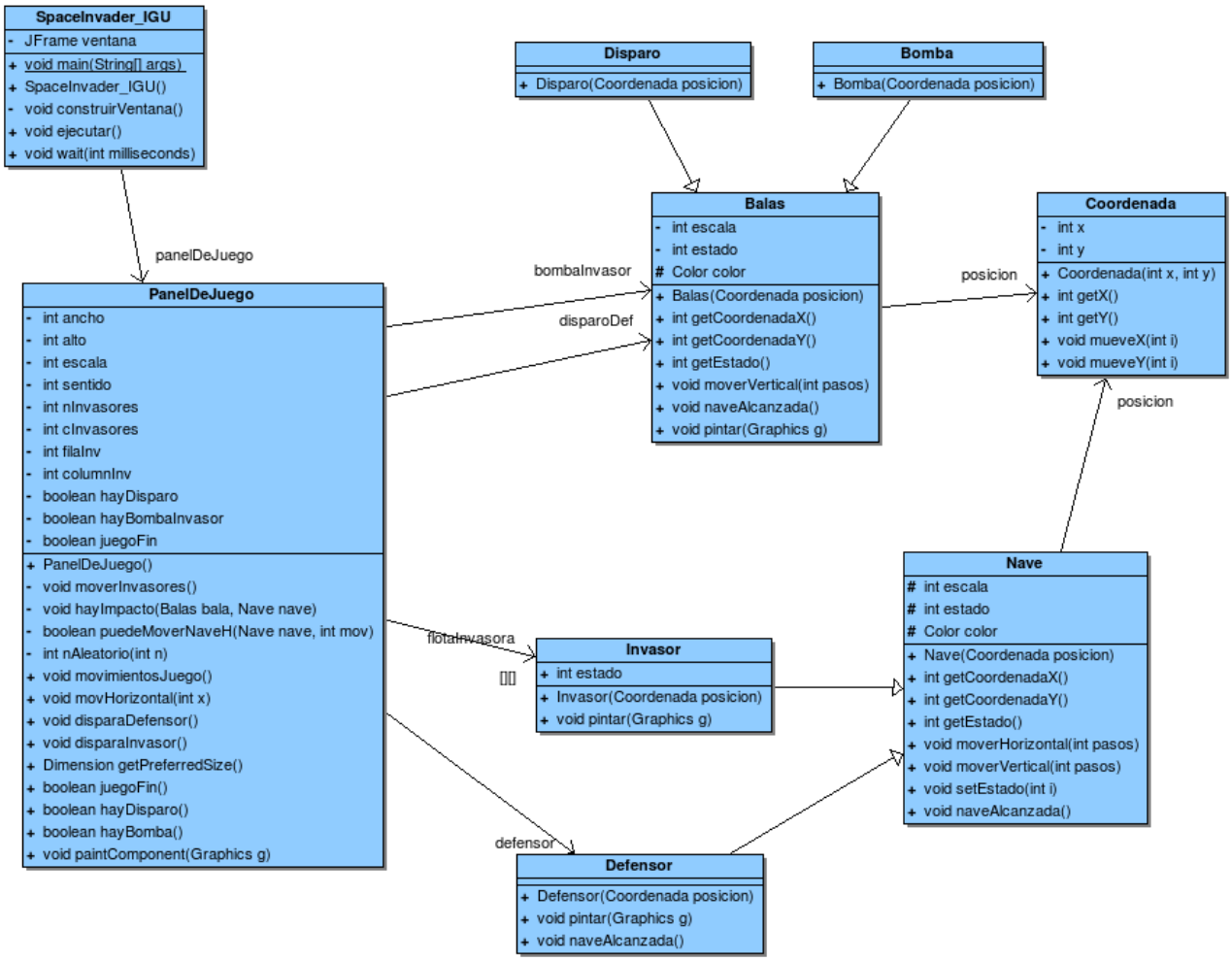
Práctica de Programación Orientada a Objetos – Junio 2012

José M. Coca de la Torre.
jcoca2@alumno.uned.es
Telf: 636214099


SpaceInvader_IGU
- JFrame ventana
+ void main(String[] args)
+ SpaceInvader_IGU()
- void construirVentana()
+ void ejecutar()
+ void wait(int milliseconds)
Disparo
+ Disparo(Coordenada posicion)
Bomba
+ Bomba(Coordenada posicion)
Balas
- int escala
- int estado
Color color
+ Balas(Coordenada posicion)
+ int getCoordenadaX()
+ int getCoordenadaY()
+ int getEstado()
+ void moverVertical(int pasos)
+ void naveAlcanzada()
+ void pintar(Graphics g)
Coordenada
- int x
- int y
+ Coordenada(int x, int y)
+ int getX()
+ int getY()
+ void mueveX(int i)
+ void mueveY(int i)
panelDeJuego
PanelDeJuego
- int ancho
- int alto
- int escala
- int sentido
- int nInvasores
- int cInvasores
- int filaInv
- int columnInv
- boolean hayDisparo
- boolean hayBombaInvasor
- boolean juegoFin
+ PanelDeJuego()
- void moverInvasores()
- void hayImpacto(Balas bala, Nave nave)
- boolean puedeMoverNaveH(Nave nave, int mov)
- int nAleatorio(int n)
+ void movimientosJuego()
+ void movHorizontal(int x)
+ void disparaDefensor()
+ void disparaInvasor()
+ Dimension getPreferredSize()
+ boolean juegoFin()
+ boolean hayDisparo()
+ boolean hayBomba()
+ void paintComponent(Graphics g)
bombaInvasor
disparoDef
posicion
posicion
flotaInvasora
Invasor
+ int estado
+ Invasor(Coordenada posicion)
+ void pintar(Graphics g)
Nave
int escala
int estado
Color color
+ Nave(Coordenada posicion)
+ int getCoordenadaX()
+ int getCoordenadaY()
+ int getEstado()
+ void moverHorizontal(int pasos)
+ void moverVertical(int pasos)
+ void setEstado(int i)
+ void naveAlcanzada()
defensor
Defensor
+ Defensor(Coordenada posicion)
+ void pintar(Graphics g)
+ void naveAlcanzada()

La aplicación se presta a diversos diseños, sobre todo en cuanto clases relativas a las naves y sus disparos: Por ejemplo se puede establecer bloque que genere un cuadrado (gfill3DRect) y crear las naves y los disparos como matrices de bloques, otro posible diseño seria establecer una sola clase para cada pieza, pero esta opción duplicaría en exceso el código a medida que se crearan nuevos tipos de naves y diferentes tipos de disparo.
En cuanto a la parte gráfica y la parte mecánica del juego, están bien diferenciadas y cumplen según el enunciado.

Al final me decidí por cinco clases, a saber:

- **SpaceInvadersIGU**: Interfaz gráfica.
- **PanelDeJuego**: Tablero, que hereda de la clase Jpanel con la lógica/mecánica del juego
- **Coordenadas**:Para posicionar las naves y los disparos.
- **Nave**: superclase para las naves, tiene dos subclases una para el **defensor** y otra para los **invasores**
- **Clase Bala**. Padre de las subclases de los proyectiles de los defensores (clase **disparo**) y los invasores (clase **Bomba**)

En un principio había pensado por sencillez crear una sola clase Nave para defensores e invasores y lo mismo para la clase Bala, pero una vez visto el vídeo con las orientaciones sobre la práctica decidí hacerlo de este modo para facilitar desarrollos posteriores.

También podrían haberse sacado algunos métodos de la clase Panel de Juego y refactorizarlos en otra clase. Pero una vez comprobado que todo funciona según lo pedido he dejado esas mejoras para los posteriores desarrollos mencionados arriba.

Así con este código para la clase **Nave** y las subclases podríamos crear nuevos tipos de Naves invasoras fácilmente, de igual modo con los disparos de las naves podrían crearse nuevos tipos de disparos.

Una vez establecidas las clases Coordenada, Nave y Bala, las subclases Disparo y Bomba, la parte mas compleja de la aplicación es la clase Panel de Juego, donde se establece toda la mecánica de la aplicación tanto la automática (movimiento de los Invasores, disparos y bombas) como el movimiento de la nave defensora con sus respectivas restricciones según el enunciado de la práctica.

Clase Panel de Juego

Implementa la parte lógica del juego, el tablero, la restricción de movimientos, la ciclo de movimientos de las naves y los disparos.

Atributos

- Medidas del tablero: ancho,alto;
- Escala de las formas: escala
- Invasores: Sentido del movimiento, filas y columnas de la matriz de invasores.
- etc

panelDeJuego()

Constructor de la clase. Inicia el juego, medidas del tablero, llamada a la clase nave para pintarla, para crear la flota invasora uso un array de 4 filas y 9 columnas. Otra posibilidad en cuanto a la flota invasora pude ser crear una clase flotaInvasora e implementa fuera de esta clase todos los métodos de movimiento y posicionado en el tablero de la flota. Pero lo he dejado así porque me ha resultado mas fácil trabajar con el array de invasores.

Métodos privados.

moverInvasores()

Establece el bucle que hace mover a los invasores de lado a lado del tablero, haciendo el cambio de sentido y bajando verticalmente. Si alguna nave invasora llega a la ultima línea del tablero o colisiona con la nave defensora, habrá acabado el juego.

hayImpaco()

Controla si los disparos de los defensores o las bombas de los invasores hacen impacto en uno u otro respectivamente, en el primer caso si hay impacto con el defensor termina el juego, en el segundo caso va eliminando invasores,, si todos los invasores son eliminados termina el juego.

puedoMoverNave(Nave nave, int mov)

Restricción del movimiento horizontal de las naves, comprueba si ha llegado alguno de los laterales del tablero.

nAleatorio(int n)

Devuelve un número aleatorio entre 0 y n.

Métodos públicos

movimientosJuego()

Método que implementa los movimientos automáticos del juego, es decir los de las naves invasoras, las bombas arrojadas por estas y los disparos del defensor.

movHorizontal(int x)

Método para mover el defensor, que es controlado con el *keyListener* de la interfaz gráfica.

disparaDefensor()

Crea el disparo del defensor, hace una consulta para establecer la posición de inicio en el morro del defensor.

disparaInvasor()
Llamada para crear el disparo del Invasor, elige un defensor de forma aleatoria y el tiempo entre dos disparos también es aleatorio.

getPreferredSize()
Para que el contenedor de la ventana establezca las dimensiones

juegoFin()
Controla el bucle juego de la interfaz gráfica.

hayDisparo()
Para controlar que no haya dos disparos a la vez

hayBomba()
Para controlar que no haya dos bombas a la vez

paintComponent(Graphics g)
Pinta el tablero y la pieza en el componente Jpanel

Clase Balas y subclases Bomba y Disparo.

Podíamos haber usado una sola clase, pero en aras de manejar el concepto de herencia y pensando en futuros desarrollos he decidido esta diseño de clases.

Atributos
- Coordenada: Posición en el tablero de la nave
- Escala: para posibles cambios de tamaño.
- Estado: Una nave puede estar viva o muerto.
- Color: Para establecer los colores.

Balas(Coordenada posicion)
Constructor de la clase para establecer los valores iniciales.

Métodos de consulta

getCoordenadaX()

getCoordenadaY()

getEstado()
Devuelven, las coordenadas y es estado de la nave.

Métodos de movimiento

moverVertical(int pasos)
Mueve la nave los pasos indicados

naveAlcanzada()
si la nave alcanzada es el defensor el disparo cambia de color y se pone amarillo también cambia el estado del disparo a 0 (destruido de modo que la clase panelDeJuego no volverá a pintarlo)

pintar(Graphics g)

Método para pintar las balas en el tablero centrándolas en el morro de la nave que hace el disparo bien sea el defensor o un invasor.

Subclases disparo y bomba

Ambas subclases solo hacen una llamada a la superclase, y establecen el color de cada uno de ellos.
Es cierto que no parece necesario crear estas subclases pero como ya he mencionado antes, para futuros desarrollos siembre será mas fácil implementar nuevos tipos de disparo (disparo rápido, bombas destructoras, congeladores de movimiento, etc).

Clase Nave y subclases Defensor e invasor.

La superclase nave nos servirá para mover las naves y establecer su estado vivo o muerto, es superclase de la subclase defensor e invasor

Atributos

Estado: vivo o muerto
posicion: Posición inicial.
Color: Color de la nave
Estado: Estado inicial

Nave(Coordenada posicion)

Constructor de la clase.

Métodos de la consulta

Devuelven datos: coordenadas, estado.

getCoordenadaX()

getCoordenadaY() {

getEstado() /**

Métodos de movimiento y estado

Controlan el movimiento de las naves, y su estado, cuando un nave es alcanzada establece el estado a muerto, para que la clase panelDeJuego no vuelva a pintar las naves alcanzadas

moverHorizontal(int pasos)

moverVertical(int pasos)

setEstado(int i) {

naveAlcanzada()

Subclase Defensor

La clase defensor está formada en resumen por dos rectángulos uno grade que hace la base y un cuadrado que hace el morro de la nave defensora, la pintamos en color azul. Cuando es alcanzada se establece en color amarillo.

Defensor(Coordenada posicion)

Constructor de la la clase, llama al constructor de la superclase, establece el color de la nave defensor en azul

pintar(Graphics g)

Para pintar la nave en el panel de juego.

naveAlcanzada()

Pinta de amarillo el defensor cuando es alcanzado. También llama al método de la superclase.

Subclase invasor

La clase defensor esta formada por cuatro cuadrados verdes la implementación es diferente con la única intención de mostrar y probar otras posibilidades.
Por lo demás es similar a la otra subclase.

Coordenada

Clase para establecer la posicion de las naves, los disparos y la bombas.

Atributos

- fila x
- Columna y

Coordenada(int x, int y)

Constructor de la clase

Métodos de consulta

public int getX()

public int getY()

Métodos de movimiento de las piezas

mueveX(int i) // Movimiento horizontal
mueveY(int i) // Movimiento Vertical

Clase SpaceInvaders_IGU

Es la clase de la parte gráfica, controla el tiempo en el juego y llama a la clase panel de juego.
Crea la ventana donde se ejecuta la aplicación y llama a la ejecucion de la misma