**Logística**

* Memoria (PDF/Word)
  + Requisitos funcionales
  + Diagrama de clases
  + Pruebas: justificación de que se han cumplido los requisitos funcionales.
* Fuentes (BlueJ)
* Jar (ejecutable)

Entregar en ZIP comprimido CON CONTRASEÑA: practicaPOO (para evitar el filtro antispam)

Enviar a (AMBAS COSAS)

* [juanci@lsi.uned.es](mailto:juanci@lsi.uned.es), asunto: [PRACTICA POO 2013] Nombre
* Plataforma ALF (entrega de trabajos)

Fecha de entrega:

* Junio: 7 de Junio (viernes)
* Septiembre: 1 de Septiembre

Guardias: Jueves de 16 a 20 – 913987620 (hilo en el foro)

**Realización**

* Vista principal
* Nave “guardiana”:
  + puede moverse en las cuatro direcciones y disparar (disparo continuo)
  + 1 vida
* “Marcianos”:
  + Se generan todas al comienzo
  + Cuando desaparecen por un lado de la pantalla (izquierda habitualmente; pero también arriba o abajo en las que se mueven verticalmente), aparecen por el otro.
  + Tipos
    - Movimiento exclusivamente horizontal
    - Movimiento horizontal y aleatorio arriba-abajo
* Menú
  + Nivel de dificultad: fácil, medio, difícil, extremo, determina:
    - el número de naves que se van a generar inicialmente (no se regeneran)
    - Velocidad a la que se desplazan
* Posibles mejoras:
  + Varias vidas
  + Puntuación

Uso de eclipse / BlueJ

Aspectos técnicos

* Modelo Vista Controlador (**IMPORTANTE**!!)
  + Las entidades del juego deben modelarse de manera independiente a lo que sería su visualización (**Modelo**). Ej.:
    - Nave: posición x, y, imagen, comportamiento (reacción a estímulos externos (implementado mediante métodos. Ej.: mover, disparar, )), alienígenas, etc.
    - Alienígena: posición x, y, imagen, comportamiento (dependerá del tipo de alienígenas).
  + **Vista** (alienígenas pintados en un JPanel)
  + **Controlador** (es el “mediador” que organiza la lógica del juego comunicando la vista y el modelo: detecta cambios en la vista y se los transmite al modelo para que se actualice y detecta cambios en el modelo y los propaga hasta la vista.
    - Síncrona: actualización del modelo cada cierto intervalo de tiempo y actualización de la vista. Esto lo hace a través de un **timer** (objeto de Swing) que llama a una función del controlador de manera periódica.
    - Asíncrona: manejador de las teclas (una pulsación es un evento asíncrono que avisará a nuestro controlador cada vez que pulsemos una tecla para que reaccione de una manera específica. Ej.: informar a la nave (la nave en consecuencia actualizará su modelo para que en el siguiente evento del timer aparezca en la vista).
  + En principio, tanto vista como controlador van a estar dentro de una clase que extienda a JPanel.
* JPanel

NOTAS:

* Repaint invalida el JPanel y hace que (aunque no de manera inmediata) se vuelva a llamar a Paint para actualizar la vista.
* Dibujar “a mano alzada”
  + **No es lo más recomendable. Es preferible para la práctica usar pintado de bitmaps.**
  + RenderingHints (~estados de render?)
* Clase que extiende JFrame y crea el JPanel
* Coordenadas: (0, 0) superior, izquierda, crece abajo y derecha.
* Es posible dibujar “fuera” del JPanel (hace clipping)
* ImagIcon: accede a un recurso (Ej.: URL, fichero, etc.) INTERESANTE: desde subclase de JPanel:
  + Ej.: ImageIcon ii = new ImageIcon(**this.getClass().getResource(“../uned.jpg”)**)
* Hay varias clases timer en java: nosotros vamos a trabajar con los timers de swing.
* Clase toolkit: “pegamento” entre las clases independientes de plataforma de java y la funcionalidad dependiente de plataforma.
  + Toolkit.getDefaultToolkit().sync();
* JPanel.setDoubleBuffered
* Pregunta: ¿para qué invocar al método dispose al final de paint?
* En el evento asíncrono del manejador de teclas no actualizamos el estado real del modelo. En realidad, no tenemos toda la información de entrada (sólo un evento aislado). Tenemos que agrupar toda la información de entrada y procesarla en la actualización del modelo (NOTA (mía): además, no es una buena idea hacer ese cómputo – que puede ser complejo – en respuesta a un evento asíncrono en una aplicación en tiempo real).
* En el ejemplo de la clase KeyAdapter es una clase interna al “controlador” para poder acceder al modelo. Se podría pasar en un constructor, etc.