VJ1203 – Programación I
Práctica 5: Pygame y Juegos*
Curso 2014/15

Introducción a Pygame

En esta práctica vamos a utilizar Pygame, que es un conjunto de módulos Python con los que se pueden crear videojuegos 2D de una manera rápida y sencilla. También puede utilizarse para crear programas multimedia o interfaces gráficas de usuario. Pygame está orientado al manejo de *sprites* –que son imágenes de mapa de bits, a menudo pequeñas y parcialmente transparentes– lo que permite trabajar con formas diferentes del rectángulo.

Para trabajar en Pygame iremos configurando una pantalla (la ventana principal) superponiendo objetos de diferentes tamaños y formas (que pueden ser formas geométricas, imágenes, etc). En la dirección `http://www.pygame.org/docs/` podéis encontrar información detallada y tutoriales sobre Pygame. En concreto, en la dirección `http://www.pygame.org/docs/ref/index.html` podéis consultar todos los objetos y métodos que forman parte de Pygame.

En el programa a continuación se puede ver el modo de trabajo en Pygame: inicialización seguida de un bucle con la actualización de la superficie de dibujo (fondo, imágenes, etc), el tratamiento de eventos (teclado, ratón, etc) y la visualización en pantalla de la superficie de dibujo:

```
import pygame
from pygame.locals import *
import time

# Constantes para la inicialización de la superficie de dibujo
PANTANCHO = 640
PANTALTO = 480
WHITE = (255, 255, 255)

def main():
    # Inicialización de Pygame
    pygame.init()
    # Inicialización de la velocidad de refresco
    FPS = 60
    fpsClock = pygame.time.Clock()
    # Inicialización de la superficie de dibujo (display surface)
    screen = pygame.display.set_mode((PANTANCHO, PANTALTO))
    pygame.display.set_caption('Gato␣con␣animación␣y␣sonido')
```

*La práctica 5 tiene una duración de tres sesiones.

```
    # Inicialización del sonido
    beep = pygame.mixer.Sound('beeps.wav')
    # Inicialización de la imagen del gato y de su tamaño
    cat = pygame.image.load('cat.png')
    (cat_tamx, cat_tamy) = cat.get_size()
    # Inicialización de las coordenadas del gato
    catx = 10
    caty = 10

    # Bucle principal del juego
    fin = False
    direccion = 'E' # Este
    screen.fill(WHITE)
    while not fin:
        direccion_anterior = direccion
        if direccion == 'E': # Este
            catx += 5
            if catx+cat_tamx >= PANTANCHO:
                catx -= 5
                direccion = 'S'
        elif direccion == 'S': # Sur
            caty += 5
            if caty+cat_tamy >= PANTALTO:
                caty -= 5
                direccion = 'O'
        elif direccion == 'O': # Oeste
            catx -= 5
            if catx <= 0:
                catx += 5
                direccion = 'N'
        elif direccion == 'N': # Norte
            caty -= 5
            if caty <= 0:
                caty += 5
                direccion = 'E'
        if direccion_anterior != direccion:
            beep.play()
            time.sleep(1)
            beep.stop()
        # Copiar la imagen del gato a la superficie de dibujo
        screen.blit(cat, (catx, caty))
        # Tratar los eventos (teclado, ratón, etc)
        for event in pygame.event.get():
            if event.type == QUIT:
                fin = True
        # Visualizar en pantalla la superficie de dibujo
        pygame.display.update()
        fpsClock.tick(FPS)

    pygame.quit()

if __name__ == '__main__':
    main()
```

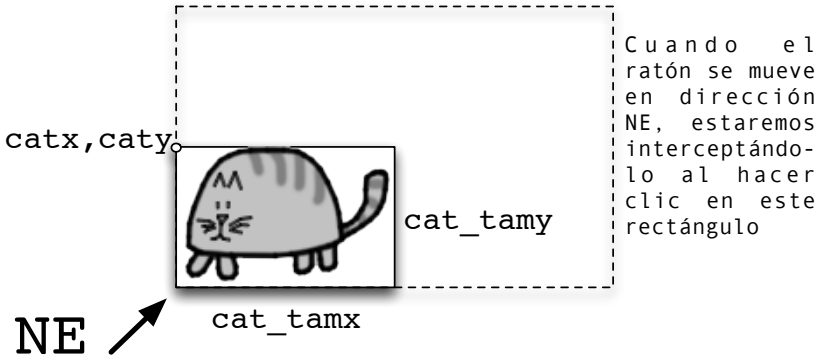
Ejercicios de imágenes con Pygame

Ej. 1 — Desplazar al gato por el borde de la pantalla sin que deje rastro Estudia y trata de enterder el programa anterior y después ejecútalo. Piensa qué cambios habría que hacer para que el gato no deje rastro por donde va pasando.

Aplica los cambios a la función `main()` y comprueba que funciona correctamente. Para terminar, modifica el valor de la variable `FPS` (que es una constante) y trata de entender qué sucede.

Ej. 2 — Desplazar al gato por la pantalla en diagonal, a partir de una posición y de una dirección iniciales En este programa debes leer por teclado por un lado las coordenadas correspondientes a la posición inicial del gato, y por otro una dirección inicial en ángulo de 45° (o sea, NE, SE, SO o NO). El programa deberá hacer que el gato se mueva en diagonal a partir de la posición indicada y empezando con esa dirección inicial. Cuando el gato llegue a algún lateral de la pantalla deberá "rebotar" saliendo en otra dirección, también en ángulo de 45°.

Ej. 3 — Desplazar al gato por la pantalla e interceptarlo con el ratón Tomando como punto de partida el programa del **Ej. 1** o del **Ej. 2**, escribe un programa que además de hacer que se mueva el gato lo cambie de dirección si hacemos clic con el ratón en una posición que se encuentre en su trayectoria, avisando con un *beep* sonoro. Para simplificar, tendremos en cuenta los clics sobre un rectángulo definido en función de la posición y las dimensiones de la figura del gato, por ejemplo:



Guía:

- Para determinar si un punto está dentro un rectángulo como el anterior definir la función `punto_dentro(posx, posy, dimx, dimy, x, y)`, que dados el extremo superior izquierdo del rectángulo (`posx` y `posy`), su tamaño (`dimx` y `dimy`) y las coordenadas de un punto (`x` e `y`), devuelva un booleano que indique si el punto está dentro del rectángulo que definen los parámetros `posx`, `posy`, `dimx` y `dimy`.
- Para identificar el evento pulsación del botón izquierdo del ratón usar la expresión `event.type == pygame.MOUSEBUTTONDOWN and event.button == 1`, y para obtener la posición donde se ha pulsado utilizar el método `pygame.mouse.get_pos()`.

El "buscaminas"

El "buscaminas" normalmente se juega sobre un tablero cuadrado de $n \times n$ casillas donde se han colocado un número determinado de bombas. Todas las casillas del

tablero estarán ocultas inicialmente. Cuando el jugador seleccione una casilla para destaparla, si hay una bomba explotará, terminando el juego. Si no hay bomba la casilla se destapará y se mostrará el número de bombas que hay en las casillas adyacentes (incluyendo las diagonales). Si el jugador deduce que en una casilla determinada hay una bomba, deberá marcarla con una bandera (*flag*). El jugador continuará seleccionando casillas para destaparlas o marcándolas con un "flag", si cree que esconden una bomba. El juego terminará bien cuando explote una bomba o bien cuando el jugador haya marcado todas las bombas con un "flag", en cuyo caso se comprobará si lo ha hecho correctamente. Ver `http://es.wikipedia.org/wiki/Buscaminas` para más detalles.

Ejercicios de tableros para el "buscaminas"

Ej. 4 — Construir los tableros para jugar al "buscaminas" En este ejercicio has de construir los tableros para jugar al "buscaminas", así como una serie de funciones/procedimientos para trabajar con ellos. La estructura de datos que usaremos para los tableros será una matriz (lista de listas) cuya dimensión dependerá del nivel de dificultad que el usuario haya elegido. Si el nivel es `'E'` (*easy*) la dimensión será 8×8 y habrá 10 bombas, si es `'M'` (*medium*) la dimensión será 16×16 y habrá 40 bombas, y si es `'H'` (*hard*) la dimensión será 16×30 y habrá 99 bombas.

Utilizaremos **dos matrices**: una matriz para el tablero que se oculta (**tablero oculto**), cuyas casillas pueden contener bien una bomba o bien un número del 0 al 8 que indica las bombas que hay en las casillas adyacentes; y otra matriz para el tablero que se muestra al usuario (**tablero visible**), cuyas casillas pueden contener un "flag" o una marca de casilla oculta, además de lo anterior.

Para terminar, los valores que se pueden almacenar en las casillas de las matrices se deben definir con constantes, que utilizaremos en todas las funciones sin necesidad de pasarlas como parámetro. Por ejemplo:

```
C0='0'
C1='1'
...
C8='8'
CBOMBA='B'
CFLAG='F'
COCULTA='-'
```

Guía:

- Definir la función `crear_tablero_oculto(filas, columnas)`, que debe crear y devolver una matriz con las filas/columnas indicadas, para el tablero oculto. Inicialmente todas las casillas deberán contener `'0'` (constante `C0`).
- Definir la función `crear_tablero_visible(filas, columnas)`, que debe crear y devolver una matriz con las filas/columnas indicadas, para el tablero visible. Inicialmente todas las casillas deberán contener `'-'` (constante `COCULTA`).
- Definir el procedimiento `poner_bombas_tablero_oculto(toculto, bombas)`, que dado un tablero oculto y el número de bombas que corresponde al nivel modifique el tablero colocando ese número de bombas en posiciones

aleatorias. Tener en cuenta que las casillas con bomba deberán contener `'B'` (constante `CBOMBA`).

- Definir el procedimiento `poner_info_tablero_oculto(toculto)`, que dado un tablero oculto con bombas lo modifique añadiendo en cada casilla la información sobre el número de bombas que hay en las casillas adyacentes. Al finalizar las casillas deberán contener una bomba `'B'` (constante `CBOMBA`) o bien uno de los valores del `'0'` al `'8'` (constantes `C0` a `C8`).
- Definir el procedimiento `imprimir_tablero(tablero)`, que dado un tablero (oculto o visible) lo imprima por pantalla en modo texto.
- Definir la función `tablero_visible_destapar(tvisible, toculto, fila, columna)`, que dados un tablero visible y uno oculto, y la fila/columna de una casilla a destapar, modifique el tablero visible poniendo la información del tablero oculto en la fila/columna indicados, si el movimiento se puede realizar. Además la función deberá devolver un booleano, si el movimiento se ha realizado (`True` si se ha destapado una bomba, con lo que el juego terminará con una explosión; `False` si no), y `None` en caso contrario. El movimiento no se realizará cuando la fila/columna estén fuera de rango, o cuando la casilla ya esté destapada.
- Definir la función `tablero_visible_marcar(tvisible, fila, columna, onoff)`, que dados un tablero visible, la fila/columna de una casilla a marcar, y un booleano indicando si se quiere marcar o desmarcar, modifique el tablero visible poniendo/quitando un "flag" en la fila/columna indicados, si el movimiento se puede realizar. Además la función deberá devolver un valor numérico si el movimiento se ha realizado (`+1` si se ha marcado una bomba, y `-1` si se ha desmarcado), y `None` en caso contrario. El movimiento no se realizará cuando la fila/columna estén fuera de rango, cuando la casilla que se quiere marcar ya esté marcada, o cuando la casilla que se quiere desmarcar no esté marcada.
- Definir la función `comprobar_tablero_visible(tvisible, toculto, bombas)`, que dados un tablero visible y uno oculto, y el número de bombas que corresponde al nivel, devuelva un booleano que indique si el jugador ha identificado todas las bombas correctamente, o sea si todas las casillas con "flag" (constante `CFLAG`) en el tablero visible se corresponden con bombas del tablero oculto (constante `CBOMBA`), siempre y cuando se hayan marcado todas las bombas que esconde el tablero.

Ejercicios para el "buscaminas" (en modo texto)

Ej. 5 — Menú del juego del "buscaminas" En este ejercicio has de definir una función `menu_buscaminas()` que: (1) muestre por pantalla las opciones posibles, (2) lea de teclado una opción dentro de un bucle, hasta que la opción introducida sea válida, y (3) devuelva la opción leída. Las opciones disponibles serán las siguientes:

1. Destapar casilla
2. Marcar casilla
3. Desmarcar casilla
4. Bombas por detectar
5. Salir

Ej. 6 — Bucle del juego del "buscaminas" En este ejercicio has de definir una función `main()` con el bucle para el juego del "buscaminas". Para ello debes utilizar las funciones/procedimientos que has definido en el **Ej. 4** y en el **Ej. 5**.
Guía:

- La función ha de leer de teclado el nivel de dificultad (`'E'`, `'M'` o `'H'`), inicializar los tableros, inicializar las variables necesarias, y a continuación entrar en un bucle hasta que el juego termine.
- El bucle debe: (1) visualizar el tablero visible; (2) mostrar el menú de opciones y en función de la opción escogida realizarla, leyendo de teclado los parámetros necesarios; y (3) determinar si el juego ha terminado (porque se ha destapado una bomba o porque ya se han marcado las posiciones de todas las bombas que esconde el tablero).
- Al terminar el bucle se deberá imprimir un mensaje indicando si el jugador ha ganado (o si ha "volado").

Ejercicios para el "buscaminas" (en modo gráfico)

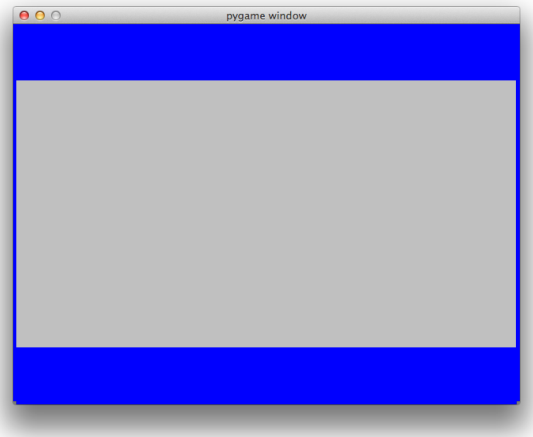
Ej. 7 — Visualizar el tablero del "buscaminas" en modo gráfico (OPCIONAL) En este ejercicio debes definir las funciones/procedimientos para visualizar en modo gráfico el tablero visible del juego del "buscaminas".
También en este caso, debes definir y usar constantes. Por ejemplo, además de las constantes definidas en el **Ej. 4**:

```
NUMFIL=16        # Número de filas del tablero (fijo)
NUMCOL=30        # Número de columnas del tablero (fijo)
MINAS=99         # Número de minas del tablero (fijo)

C0image = pygame.image.load('empty.png')   # Imágenes de las casillas
C1image = pygame.image.load('1.png')
...
C8image = pygame.image.load('8.png')
CBOMBAimage = pygame.image.load('bomb.png')
CFLAGimage = pygame.image.load('flag.png')
COCULTAimage = pygame.image.load('normal.png')

PANTANCHO = 640   # Ancho de la pantalla
PANTALTO = 480    # Alto de la pantalla
(C_TAMX, C_TAMY) = C0image.get_size()      # Tamaño de las casillas
TAMSEP = 1        # Tamaño de la separación entre casillas

TABANCHO = ...    # Ancho del tablero (definir a partir de las anteriores)
TABALTO = ...     # Alto del tablero (definir a partir de las anteriores)
TABORIGX = ...    # Origen X del tablero (definir a partir de las anteriores)
TABORIGY = ...    # Origen Y del tablero (definir a partir de las anteriores)

BLUE = (0, 0, 255)
GREY = (192, 192, 192)
```

Guía:

- Definir la función `crear_imagen_tablero()`, que debe crear y devolver una superficie de Pygame (*surface*) con un rectángulo de color gris, que será el

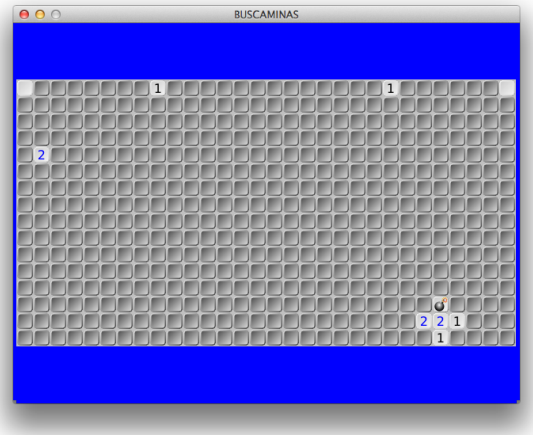
fondo sobre el que colocaremos las imágenes que representan los distintos tipos de casillas. Por ejemplo:



Para crear el fondo del tablero puedes utilizar las siguientes sentencias:

```
imagen = pygame.Surface((TABANCHO, TABALTO))
imagen.fill(GREY)
```

- Definir la función `colocar_casillas_imagen_tablero(pantalla, x, y, imagen_tablero, tvisible)`, que dados una pantalla, una posición en la pantalla (`x` e `y`), una imagen del tablero (superficie de Pygame creada con la función `crear_imagen_tablero`) y un tablero visible, dibuje sobre la pantalla en la posición indicada tanto la imagen del tablero como las casillas correspondientes. Por ejemplo:



Ej. 8 — Leer la jugada del "buscaminas" en modo gráfico (OPCIONAL) En este ejercicio has de definir la función `leer_movimiento_buscaminas_raton()`, que debe leer una jugada del juego del "buscaminas" en modo gráfico. El objetivo es que el usuario indique la casilla de su jugada haciendo clic directamente en la ventana gráfica. La función debe contener un bucle que terminará cuando el usuario

pulse dentro del tablero, devolviendo la fila y columna con la que se corresponde la posición donde ha pulsado.
Guía:

- Recuerda que para identificar la pulsación del botón izquierdo del ratón y para obtener la posición donde se ha pulsado puedes usar `event.type == pygame.MOUSEBUTTONDOWN and event.button == 1` y `pygame.mouse.get_pos()`, respectivamente.

Ej. 9 — Bucle del juego del "buscaminas" en modo gráfico (OPCIONAL)
En este ejercicio has de definir una función `main()` con el bucle para jugar al "buscaminas" en modo gráfico. Para ello debes utilizar las funciones/procedimientos que has definido en los ejercicios anteriores. También puedes añadir los elementos necesarios para que todos los mensajes al usuario se impriman en la ventana gráfica en vez de la consola.