Projecto Integrado

Hungry Python

Miguel Casco nº 28966

Ricardo Fusco nº 29263

David Rosmaninho nº 29896

Introdução

O projecto que pretendemos elaborar chama-se Hungry Python. Trata-se de um jogo feito ao estilo do jogo Snake em python recorrendo à biblioteca pygame, basta apenas fazer o download da instalação da mesma para poder correr o jogo.

O jogador começa por controlar uma pequena cobra que se move constantemente pelo ecrã. No entanto este não pode parar ou abrandar a cobra mas pode determinar em que direcção se move. Entretanto um rato aparece aleatoriamente no ecrã e o jogador deve controlar a cobra com o objectivo de comer o rato. Cada vez que a cobra comer um rato cresce em tamanho aparecendo mais um segmento no corpo da mesma enquanto que outro rato aparece outra vez aleatoriamente no ecrã. O jogo acaba quando a cobra toca nos limites da janela do jogo ou quando se come a si própria.

O jogo irá ter um ecrã inicial com alguns elementos gráficos que após pressionada qualquer tecla irá iniciar o loop do jogo. Quando o jogador perder o jogo ser-lhe-á apresentado um ecrã de "game over" que lhe dará a hipótese de recomeçar um jogo novo ou apenas de parar a execução do mesmo.

Desenvolvimento

À medida que se for jogando e se se observar as variáveis e a constituição do código do jogo pode-se observar que os segmentos da cobra e o rato cabem sempre numa grelha. Nós chamámos a cada quadrado da grelha uma "célula". Estas células têm o seu próprio sistema de coordenadas cartesianas como, por exemplo, sendo (0, 0) a célula do canto superior esquerdo.

```
############
# Imports #
#################################################################################

""" Aqui sao importados os modulos necessarios ao funcionamento do programa dos
quais se destaca o modulo pygame. O pygame.locals contem todas as constantes
necessarias como MOUSEMOTION, MOUSEBUTTONUP, QUIT, entre outras. Ora este import
constitui apenas uma maneira de facilitar a escrita, por exemplo, em vez de
escrever pygame.locals.QUIT podemos apenas escrever QUIT."""
import random, pygame, sys
from pygame.locals import *

##############
# Constantes #
#################################################################################

"""Estas variaveis constantes determinam valores fundamentais para o jogo.
Usando estas constantes em vez de usar os valores directamente sem os atribuir
a uma variavel, e mais facil fazer mudancas visto que so temos de mudar os
valores num sitio - onde sao atribuidos os mesmos a variaveis. Neste jogo
optamos por dividir a janela em celulas(quadrados) que sao areas onde os
segmentos da cobra e do rato podem existir."""

FPS = 14
compJanela = 640 # comprimento da janela em pixels
altJanela = 480 # altura da janela em pixels
tamCel = 20   # tamanho das celulas da janela em pixels

"""O comprimento da janela e a altura da janela teem que ser multiplos do tamanh
das celulas(quadrados) existentes na grelha """
assert compJanela % tamCel == 0
assert altJanela % tamCel == 0
compCelulas = int(compJanela / tamCel)
altCelulas = int(altJanela / tamCel)
```

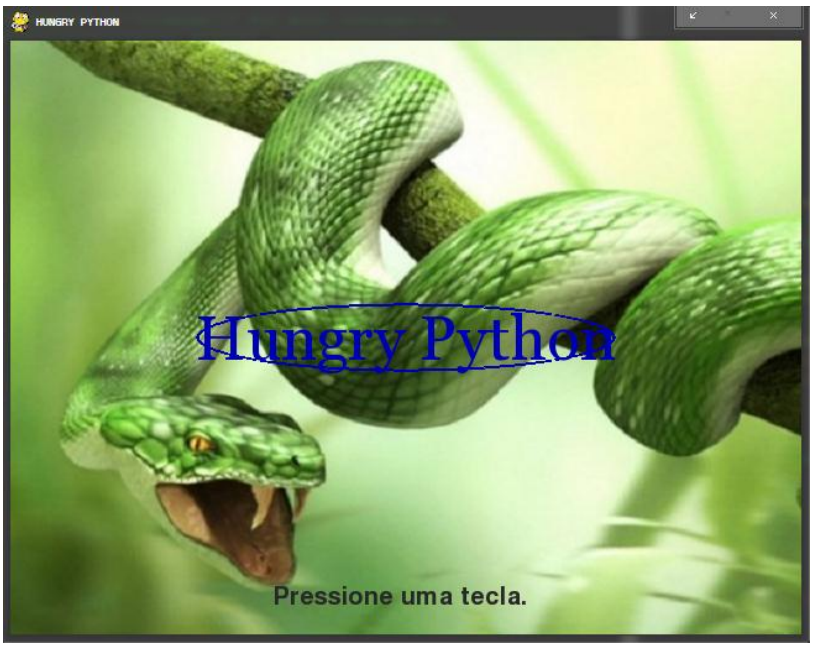
Este pedaço de código ao início do programa serve para definir algumas variáveis constantes importantes para o bom funcionamento do jogo. A altura e comprimento das células estão guardados na variável tamCel. Os asserts aqui presentes servem para assegurar que as células encaixam perfeitamente na janela, ou seja, servem para assegurar que apenas cabe na janela um número inteiro de células.

Após este pedaço de código estão várias variáveis com tuplos de 3 inteiros que servem para determinar a quantidade de cada cor primária para obter a cor final e temos a variável HEAD que, basicamente, não tem utilidade nenhuma, serve apenas para um enriquecimento da sintaxe, ou seja, para deixar o código mais perceptível (syntactic sugar).

A base do jogo foi colocada na função base(), que contem a inicialização do pygame, do clock, da superfície do jogo, do tipo de letra e tamanho e do titulo do programa (caption). Após ter inicializado tudo o jogador não começa logo imediatamente a jogar, primeiro é chamada uma função que mostra o ecrã inicial do jogo com uma imagem de fundo e uma animação com o título que serve mostrar ao jogador qual o programa que estão a correr e para lhe dar uma oportunidade de se preparar para o começo do jogo.

```
"""Funcao que mostra o ecra inicial"""
def ecrainicial():

    titleFont = pygame.font.SysFont('georgia', 50)
    titleSurf = titleFont.render('Hungry Python', True, AZUL)

    graus = 0

    while True:
        fundo = pygame.image.load('Awesome-snake-02-525x420.jpg')
        fundoRect = fundo.get_rect()
        DISPLAYSURF.blit(fundo, fundoRect)

        rotatedSurf = pygame.transform.rotate(titleSurf, graus)
        rotatedRect = rotatedSurf.get_rect()
        rotatedRect.center = (compJanela / 2, altJanela / 2)
        DISPLAYSURF.blit(rotatedSurf, rotatedRect)

        pygame.draw.ellipse(DISPLAYSURF, AZUL, rotatedRect, 2)

        teclamsg()

        if tecla():
            pygame.event.get() # verificar se ocorreu algum evento
            return
        pygame.display.update()
        FPSCLOCK.tick(FPS)
        graus += 5 # rodar 5 graus por cada frame
```

Para o ecrã inicial foram necessários três objectos para a superfície, um com o texto "Hungry Python" desenhado, outro com um aro à volta do texto e a imagem que pretendemos colocar no fundo. O render() serve para colocar o texto na superfície com a cor que acharmos apropriada. Depois é iniciado o loop para o ecrã inicial no qual vão estar dois objectos em rotação.



A função pygame.transform.rotate() não muda o objecto de superfície que tem como argumento, devolve um novo objecto de superfície com a imagem rodada pelo número de graus estabelecido. O .blit() é apenas uma função para "colar" o objecto de superfície na janela do jogo. Durante este loop é executada uma função chamada teclamsg() que mostra um texto que diz para pressionar uma tecla. Este loop vai continuar até que alguma tecla seja pressionada, é aqui que entra a função tecla().

Quando de facto se verificar que foi pressionada uma tecla e for terminado o ecrã inicial procede-se então ao início do loop do jogo onde é chamada a função onde se encontra a estrutura e a parte principal do jogo, correrjogo() que vai iniciar um jogo de Hungry Python. Esta função vai ter um return para proceder ao ecrã de game over quando a cobra do jogador colidir com alguma das paredes ou contra ela própria, sendo executada a função mostrargameover() que quando chegar a um return vai levar o loop ao inicio e executar um novo jogo de Hungry Python. Este while que contém estas duas funções(correrjogo() e mostrargameover()) vai continuar num loop infinito até que o jogador decida terminar o programa.

No início de um jogo, queremos que a cobra apareça numa localização aleatória, mas sem que apareça muito junto aos limites da janela, então para tal guardamos uma coordenada aleatória nas variáveis startx e starty (o compCelulas e a altCelulas são o número de colunas e o número de linhas respectivamente e não o comprimento e a altura em pixéis). O corpo da cobra vai ser guardado numa lista de dicionários, na qual há um dicionário por cada segmento da cobra (3 segmentos). Cada dicionário tem chaves 'x' e 'y' para as coordenadas XY do segmento do corpo. A cabeça da cobra fica na posição (startx, starty), o segundo segmento do corpo da cobra fica uma célula à esquerda da cabeça e o terceiro segmento fica duas células à esquerda da cabeça. A cabeça da cobra vai ser sempre a parte do corpo na posição 0 das coordenadas da cobra (cobraCoords[0] ou cobraCoords[HEAD]).

```
def correrjogo():
    global cobraCoords
    """ Estabelece um ponto de inicio da cobra """
    startx = random.randint(5, compCelulas - 6)
    starty = random.randint(5, altCelulas - 6)
    cobraCoords = [{'x': startx,     'y': starty},
                   {'x': startx - 1, 'y': starty},
                   {'x': startx - 2, 'y': starty}]
    direction = RIGHT
```

Após a inicialização do corpo da cobra é iniciado o loop while do jogo onde se começa o loop para a verificação de ocorrência de eventos.

Se o evento é do tipo QUIT é então chamada a função sair() para terminar o programa. Se o evento for do tipo KEYDOWN então temos que averiguar se a tecla que foi pressionada foi uma seta ou uma tecla WASD e assegurar que a cobra não se vire para ela própria, por exemplo se estiver mover-se em direcção à esquerda e se pressionássemos a seta para a direita ela iria imediatamente colidir com ela própria que não é o que queremos, é para isto que verificamos o estado da variável direction.

```
for event in pygame.event.get(): # loop para verificar os eventos
    if event.type == QUIT:
        sair()
    elif event.type == KEYDOWN:
        if (event.key == K_UP or event.key == K_w) and direction != DOWN:
            direction = UP
        elif (event.key == K_DOWN or event.key == K_s) and direction != UP:
            direction = DOWN
        elif (event.key == K_LEFT or event.key == K_a) and direction != RIGHT:
            direction = LEFT
        elif (event.key == K_RIGHT or event.key == K_d) and direction != LEFT:
            direction = RIGHT
        elif event.key == K_ESCAPE:
            sair()
```

De seguida temos o pedaço de código que faz a detecção de colisão. Podemos averiguar se a cabeça da cobra ultrapassou os limites da grelha verificando se a coordenada X da cabeça, que está guardada em cobraCoords[HEAD]['x'], é **-1**. Se for de facto **-1** significa que a cabeça está além do limite esquerdo da janela. Se a mesma em vez de **-1** for igual ao compCelulas significa que está além do limite direito da janela. O mesmo se verifica para a coordenada Y da cabeça, que está guardada em cobraCoords[HEAD]['y']. Se for **-1** significa que a cabeça está além do limite superior da janela. Se for igual à altCelulas significa que está além do limite inferior da janela. Se alguma destas condições se verificar apenas é necessário um return para sair da função correrjogo() e passar à execução da função mostrargameover() que mostra um ecrã de Game Over.

Depois da detecção das condições para game over temos a parte do código relativa à colisão da cobra com o rato. O princípio é o mesmo utilizado anteriormente para a colisão com as bordas da janela e com a própria cobra. Se a cabeça da cobra coincidir com as coordenadas XY do rato são determinadas novas coordenadas aleatórias para outro rato chamando a função locinicial(). Se a cabeça não colidir com o rato, é removido o último segmento do corpo da cobra na lista cobraCoords. Após a remoção da 'cauda' da cobra vai ser adicionado um novo segmento ao corpo da cobra (a cabeça) na direcção em que a mesma se está a mover, isto vai fazer com que a cobra cresça em tamanho por um segmento. Fica assim claro que para o caso em que a cabeça da cobra coincide com

```
""" verifica se a cobra comeu um rato """
if cobraCoords[HEAD]['x'] == ratoal['x'] and cobraCoords[HEAD]['y'] == ratoal['y']:
    """ quando a celula correspondente a cabeca esta na mesma posicao
     da celula do rato a celula que corresponde a cauda nao e removida"""
    ratoal = locinicial() # encontra uma nova localizacao aleatoria para o rato
else:
    del cobraCoords[-1] # retira o segmento correspondente a cauda
```

o rato para que ela cresça de tamanho basta não remover o segmento da cauda. Por isso é que a cobra mantem sempre o mesmo tamanho se não colidir com um rato pois é sempre removido o segmento da cauda e simultaneamente adicionado o segmento da cabeça.

De seguida temos a parte que diz respeito ao movimento e a direcção do mesmo. Para mover a cobra adicionamos um novo segmento ao início da lista cobraCoords. Como o segmento está a ser adicionado ao início da lista este irá tornar-se a nova cabeça, como já tínhamos visto anteriormente. A direcção é que vai determinar se é adicionado ou subtraído 1 da coordenada X ou Y.

É precisamente para adicionar o segmento da nova cabeça que serve o método insert(), ligeiramente diferente do append() pois podemos adicionar itens em qualquer posição da lista que queiramos.

O restante é o código necessário à criação do ecrã do jogo e os respectivos elementos gráficos como a cor de fundo, desenhar a cobra, o rato e a pontuação. É usado o pygame.display.update() para fazer um "refresh" da janela para que lá apareçam os elementos gráficos que a compõem, no fundo para os desenhar uma vez que já foram criados. Resta iniciar o clock e estabelecer os FPS's com que o jogo vai ser corrido com a função .tick().

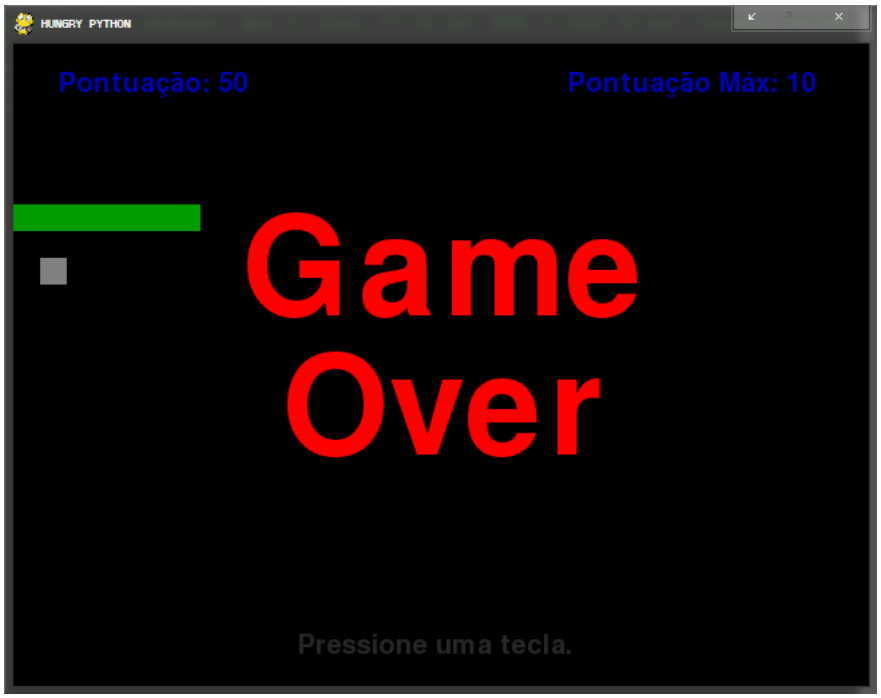
```
cobraCoords.insert(0, newHead)
DISPLAYSURF.fill(CORFUNDO)

cobra(cobraCoords)
rato(ratoal)
pontuacao((len(cobraCoords) - 3)*10)
pygame.display.update()
FPSCLOCK.tick(FPS)
```

A função que se segue, a teclamsg(), serve apenas para colocar na janela um texto que nos diz para pressionar uma tecla como já tinha sido referido anteriormente. É feito o render() da mensagem que pretendemos mostrar e a cor com que irá aparecer, o tamanho e o tipo de letra já foram inicialmente definidos na variável BASICFONT. De seguida é feito um rectângulo onde irá estar contida a mensagem, depois basta apenas indicar as coordenadas onde a mesma vai aparecer e por fim desenhar os elementos criados na janela para podermos visualizar (.blit()).

A função tecla(), já referida anteriormente, serve para, verificar se foi pressionada alguma tecla. Esta é utilizada no ecrã inicial e no ecrã de game over. A primeira coisa que esta função verifica é se há algum evento do tipo QUIT. Se houver um ou mais é executada a função sair() para terminar o programa. É exactamente para isto que serve o len(pygame.event.get(QUIT)), para verificar o comprimento da lista de eventos do tipo QUIT devolvida pela função pygame.event.get(). Se não for detectado nenhum evento do tipo QUIT é feito o get dos eventos do tipo KEYUP. Se o evento do tipo KEYUP for para a tecla Esc é chamada a função sair() e é também terminado o programa. Caso seja outro tipo

de tecla pressionada é devolvido o primeiro elemento da lista de eventos do tipo KEYUP. Caso não haja eventos do tipo KEYUP, faz-se um return para sair da função e continuar o loop do ecrã até que seja detectado algum evento.

Como já foi referido anteriormente, a função locinicial() é a função necessária para calcular uma nova localização para o rato. Esta função devolve um dicionário com as chaves 'x' e 'y', e os valores aleatórios para cada coordenada.



O ecrã de Game Over é bastante similar ao ecrã inicial apenas não tem uma animação definida, consiste apenas em mostrar dois objectos na janela, o texto de game over e o texto para pressionar uma tecla.

Depois é criado um delay de 500 milisegundos (meio segundo) antes de verificar se foi pressionada alguma tecla para que o novo jogo não comece logo de imediato se o jogador estiver a pressionar alguma tecla enquanto o jogo termina e o ecrã de game over é iniciado. É chamada a função tecla() antes de iniciar o loop while para ignorar qualquer evento que tenha havido desde que a função mostrargameover() foi chamada. Esta pausa e a chamada da função tecla() antes do loop while serve para prevenir a seguinte situação: Imagine-se que o jogador estava a tentar afastar-se dum limite da janela à última da hora, mas carregou na tecla tarde de mais e colidiu com o limite. Se isto acontece a tecla é pressionada após a função mostrargameover() ter sido chamada o que vai levar ao desparecimento quase instantâneo do ecrã de game over, fazendo com que o próximo jogo comece imediatamente surpreendendo o jogador. Esta 'pausa' serve apenas para tornar o jogo mais "user friendly".

A função seguinte (pontuacao(score)), tem apenas como objectivo renderizar a pontuação e a pontuação máxima até ao presente, criar um rectângulo para cada um, colocar as coordenadas onde queremos que apareça o texto e fazer o blit na superfície. No que diz respeito à pontuação temos o score que é o comprimento de cobraCoords menos 3 vezes 10 o que significa que por cada rato comido o jogador ganha 10 pontos. Foram também criadas duas variáveis a scoreFinal e o topScore. A variável scoreFinal adquire o valor total da pontuação no momento em que o jogador perde. A variável topScore é iniciada com 0 para comparar com a scoreFinal. Se o valor da variável topScore for menor que o valor da scoreFinal, a topScore assume o valor da scoreFinal, ou seja, quando a pontuação feita é maior ou igual à pontuação máxima, esta é actualizada e adquire o valor da pontuação feita.

```
""" Funcao que mostra durante o jogo a pontuacao """
def pontuacao(score):
    topscoreSurf = BASICFONT.render('Pontuação Máx: %s' % (topScore), True, AZUL)
    topscoreRect = topscoreSurf.get_rect()
    topscoreRect.topright = (600, 20)
    DISPLAYSURF.blit(topscoreSurf, topscoreRect)

    scoreSurf = BASICFONT.render('Pontuação: %s' % (score), True, AZUL)
    scoreRect = scoreSurf.get_rect()
    scoreRect.topright = (175, 20)
    DISPLAYSURF.blit(scoreSurf, scoreRect)
```

Com a base da cobra já definida e as suas coordenadas falta desenhá-la na superfície do jogo. É para isto que serve a função cobra(), que recebe a lista cobraCoords como argumento. Esta vai desenhar um quadrado verde por cada um dos segmentos constituintes da cobra.

```
""" Funcao que desenha a cobra """
def cobra(cobraCoords):
    for coord in cobraCoords:
        x = coord['x'] * tamCel
        y = coord['y'] * tamCel
        cobraSegmentRect = pygame.Rect(x, y, tamCel, tamCel)
        pygame.draw.rect(DISPLAYSURF, VERDEESCURO, cobraSegmentRect)
        if coord == cobraCoords[HEAD]:
            cobraCabecaRect = pygame.Rect(x+4, y+4, tamCel - 8, tamCel - 8)
            pygame.draw.rect(DISPLAYSURF, VERDE, cobraCabecaRect)
```

O loop for presente nesta função irá percorrer cada um dos dicionários da lista cobraCoords. Como as coordenadas da grelha usam toda a janela e começam na posição 0, 0 com 0 pixéis de comprimento e 0 pixéis de altura é relativamente fácil converter as coordenadas da grelha para coordenadas em pixéis. Primeiro são multiplicadas as coordenadas coord['x'] e coord['y'] pelo tamanho das células (tamCel). De seguida é criado um rectângulo para o segmento que está a ser analisado (x e y) e que irá ser passado como argumento na função pygame.draw.rect(). Tendo em conta que tamCel é o comprimento e a altura (o comprimento e a altura duma célula são ambos 20) vai ser este o tamanho do rectângulo do segmento. Depois é definida uma cor verde para todos os segmentos da cobra. Por fim é feito outro rectângulo mais pequeno por cima do rectângulo do segmento relativo a cabeça da cobra que vai ser colorido com um verde mais escuro para se distinguir a cabeça do resto do corpo.

A função rato() é bastante parecida à função cobra() exceptuando a cor do segmento relativo ao rato e é apenas um único segmento em vez de vários segmentos como é o caso da cobra.

Para concluir temos um if __name__ == '__main__' após definidas todas as funções, constantes e variáveis globais e locais, para podermos importar o programa na Shell do python e verificar se cada função está a devolver os valores correctos testando uma a uma individualmente.