# O jogo dos "queques"

O jogo dos queques é uma simplificação do jogo da Batalha Naval. Agora, em vez de navios, temos tabuleiros compostos por um conjunto de 4 pratos de 1 queque, 3 pratos de 2 queques, 2 pratos de 3 queques e 1 prato de 4 queques. Estes apenas se dispõem na horizontal.

Observe o seguinte exemplo.

A picture containing food, game

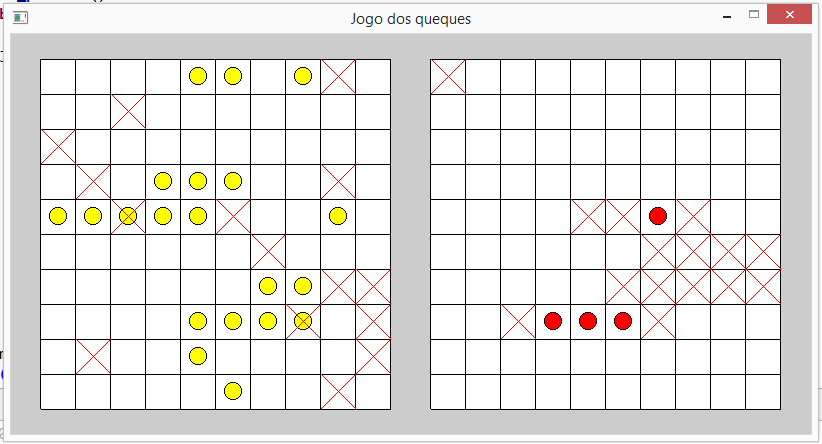
Description automatically generated

O tabuleiro tem uma dimensão 10 x 10.

# Estruturação do jogo

Este jogo já apresenta alguma complexidade, pelo que será útil utilizar a decomposição para estruturar melhor o jogo em subproblemas:

1. Criar o tabuleiro: O tabuleiro é criado como uma matriz de 10x10 valores (não mutável). Os valores são gerados de forma aleatória, representando os pratos de queques.
   * 0, quadrícula vazia;
   * 1 a 4 - queque pertencente a um prato com 1, 2, 3 ou 4 queques.
2. Desenho da UI: Neste subproblema é desenhada a interface com o utilizador UI (User Interface). Esta é composta por dois tabuleiros:
   * O tabuleiro do jogador (o do lado esquerdo na figura que se segue), onde são visualizados os seus queques e os "tiros" do oponente;
   * O tabuleiro do oponente, neste caso será o computador, (o do lado direito na figura que se segue), onde são visualizados os queques à medida que vão sendo descobertos e os "tiros" falhados pelo jogador.



E claro, a lógica do módulo p5:

1. setup(): Este subproblema irá focar-se na criação dos dois tabuleiros e em decidir o jogador que inicia o jogo.
2. draw(): O ciclo de jogo. Verifica quem joga, executa a jogada através da lógica de jogo e valida se o jogo terminou e quem é o vencedor.
3. mouse\_pressed(): O jogador irá interagir através do rato, premindo sobre a quadrícula correspondente.

# Definições globais

Para manter o jogo configurável, definiram-se as seguintes constantes, logo após a importação dos módulos p5 e random:

from p5 import \*

import random

# definicoes

LARGURA = 800 # largura da janela grafica

ALTURA = 400 # altura da janela grafica

LINHAS = 10 # numero de linhas de cada um dos tabuleiros

COLUNAS = 10 # numero de colunas de cada um dos tabuleiros

FORNADA = ((1,4), (2, 3), (3,2), (4, 1)) # 1 de 4, 2 de 3, ...

MAX\_TENTATIVAS = 50 # numero máximo de tentativas para colocar, aleatoriamente, os queques em cada um dos tabuleiros

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Eye** | Observe a forma como é criada a literal FORNADA.  Trata-se de um tuplo com 4 elementos. E cada um destes elementos é também um tuplo com dois valores.  Na realidade podemos visualizar esta estrutura da seguinte forma:   |  |  | | --- | --- | | 1 | 4 | | 2 | 3 | | 3 | 2 | | 4 | 1 |   Trata-se de uma matriz constante.  Como aceder a um determinado elemento diretamente?  FORNADA [2][1]  Qual o valor retornado por esta expressão? |

Estas definições permitem flexibilizar a dimensão da janela e do tabuleiro de jogo, assim como o número e tipo de pratos de queques no tabuleiro.

Em p5, é geralmente necessário também criar algumas variáveis globais...

# variáveis globais

agora\_joga = 0

O jogador que joga em cada jogada (0 - computador; 1 - humano).

E a seguir o tabuleiro, a lista de tiros e o número de queques ainda não comidos/encontrados, quer do jogador humano e quer do computador:

# tabuleiro, lista de tiros e queques não comidos - do jogador humano

tab\_humano = None

lista\_humano\_tiros = []

quequesH\_nao\_comidos = 0

# tabuleiro, lista de tiros e queques não comidos - do computador

tab\_computador = None

lista\_computador\_tiros = []

quequesC\_nao\_comidos = 0

As variáveis quequesC\_nao\_comidos e quequesH\_nao\_comidos determinam o número de queques que faltam para terminar o jogo.

|  |  |
| --- | --- |
| **Eye** | Porque razão é que em p5 é necessário criar todas estas variáveis globais? |

# Criar o tabuleiro

O tabuleiro é constituído por uma matriz, que tipicamente é formada por um vetor de vetores. Em Pyhon é implementada através de tuplo com 10 linhas, cada linha sendo um tuplo de 10 valores. Sendo os tuplos imutáveis, definem um tabuleiro de valores constantes. Por esta razão, os tiros não poderão alterar diretamente o tabuleiro, sendo armazenados numa lista para posterior visualização.

A função cria\_tabuleiro(lin, col, fornada) recebe três parâmetros:

* lin, o número de linhas do tabuleiro (inteiro);
* colunas, o número de colunas do tabuleiro (inteiro);
* fornada, uma matriz com a configuração dos pratos de queques a colocar no tabuleiro.

O algoritmo é composto por 2 passos:

1. Criar tabuleiro vazio;
2. Colocar no tabuleiro, de forma aleatória, os pratos com os queques.

## Criar tabuleiro vazio

Neste passo do algoritmo é criada uma lista de 10 tuplos, cada um com 10 zeros.

# construtor - cria tabuleiro com dimensão lin x col

def cria\_tabuleiro(lin, col, fornada):

# cria um tabuleiro inicial como uma lista de linhas de 0s (vazio)

tab = []

for i in range(lin):

tab.append((0,)\*col) # coloca linha de 0s

|  |  |
| --- | --- |
| **Eye** | Ainda na secção anterior referimos que o nosso tabuleiro é um tuplo... Mas embora o objetivo é termos um tabuleiro final que é um tuplo, para a sua construção é conveniente utilizar antes uma lista, pelo facto de esta ser mutável.  O que faz a seguinte instrução?  (0,)\*col  Cria um tuplo com tantos 0 quantos a variável col. Ou seja, se col for 10, cria o tuplo:  (0,0,0,0,0,0,0,0,0,0) |

|  |  |
| --- | --- |
| **Head with gears** | Algo para pensar...  Um tuplo com apenas um elemento é representado com vírgula:  (0,)  Pois se utilizar a seguinte expressão obtém, não um tuplo, mas um inteiro (o número zero).  (0)  Na realidade, nem é necessário colocar os parêntesis. Basta definir a sequência de valores do tuplo separados apenas por vírgulas:  tuplo = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 |

## Colocar no tabuleiro os pratos de queques;

O algoritmo percorre o tuplo que define os tipos de pratos a colocar (a FORNADA) e, para cada prato desse tipo, tenta colocá-lo no tabuleiro numa posição aleatória.

Neste passo do algoritmo, que é aleatório, pode não ser possível colocar todas os pratos no tabuleiro. Depende um pouco da “arrumação” destes. Para isso definiu-se um máximo de 50 tentativas. Caso não seja possível colocar algum dos pratos, a função retorna False.

# para cada tipo de prato

for n, comp in fornada:

# para cada numero de pratos

for i in range(n): # lado esquerdo do par e' a quantidade

# coloca os queques no tabuleiro

for i in range(MAX\_TENTATIVAS):

# determina coordenadas aleatorias

l = random.randint(0, lin-1)

c = random.randint(0, col-comp-1)

linha = tab[l]

# verifica se há espaço livre na linha

if linha[c:c+comp].count(0) == comp:

tab[l] = linha[:c] + (comp, )\*comp + linha[c+comp:]

break

else:

# nao e' possivel criar o prato no tabuleiro

return False

return tuple(tab)

|  |  |
| --- | --- |
| **Eye** | A função de conversão tuple faz a conversão do tabuleiro, de uma lista para um tuple.  A partir desta conversão, o tabuleiro fica imutável.  Que vantagens poderia ter criar o tabuleiro apenas com listas?  Algo para pensar ao longo da leitura do próximo capítulo. |

# Interface com o utilizador

A interface com utilizador é gráfica, representando dois tabuleiros, lateralmente.

A picture containing drawing

Description automatically generated

O tabuleiro da esquerda é o do jogador humano, onde este visualiza o posicionamento dos seus queques e os tiros dados pelo oponente.

O tabuleiro do lado direito apresenta os queques adivinhados pelo jogador humano e os tiros em falso.

A função desenhaUI() desenha na janela gráfica p5, seguindo o seguinte algoritmo:

def desenhaUI():

global tab\_humano, lista\_humano\_tiros

1. Definição das dimensões da quadrícula e origens dos tabuleiros na janela: (ox1, oy) e (ox2, oy)

quadricula = min( (LARGURA-40) // COLUNAS // 2, (ALTURA -50) // LINHAS )

ox1 = (LARGURA - 2 \* COLUNAS \* quadricula) // 2 -20

ox2 = LARGURA // 2 + 20

oy = (ALTURA - LINHAS \* quadricula) // 2

1. Desenhar quadriculas dos dois tabuleiros;

fill("white")

rect((ox1, oy), quadricula \* COLUNAS, quadricula\* LINHAS)

rect((ox2, oy), quadricula \* COLUNAS, quadricula\* LINHAS)

for i in range(1,LINHAS):

line((ox1+i\*quadricula, oy), \

(ox1+i\*quadricula, oy + LINHAS \* quadricula))

line((ox2+i\*quadricula, oy), \

(ox2+i\*quadricula, oy + LINHAS \* quadricula))

for i in range(1,COLUNAS):

line((ox1, oy + i \* quadricula), \

(ox1 + COLUNAS \* quadricula, oy + i \* quadricula))

line((ox2, oy + i \* quadricula), \

(ox2 + COLUNAS \* quadricula, oy + i \* quadricula))

1. Desenhar os queques do jogador como círculos amarelos no tabuleiro esquerdo;

# desenha os queques do jogador

ponto = (ox1 + quadricula//2, oy + quadricula//2)

for lin in tab\_humano:

for queque in lin:

if queque > 0:

fill("yellow")

circle(ponto, quadricula//2)

ponto = move\_ponto(ponto, quadricula, 0)

ponto = move\_ponto(ponto, -COLUNAS\*quadricula, quadricula)

1. Desenhar no tabuleiro direito os tiros do jogador como pontos vermelhos, caso acerte, ou cruzes vermelhas caso falhe;

# desenha os tiros do jogador

for lin, col, tiro in lista\_humano\_tiros:

if tiro > 0:

ponto = (ox2 + quadricula//2 + col\*quadricula, \

oy + quadricula//2 + lin\*quadricula)

fill("red")

circle(ponto, quadricula//2)

else:

ponto = (ox2 + col\*quadricula, oy + lin\*quadricula)

stroke("red")

line(ponto, move\_ponto(ponto,quadricula, quadricula))

line(move\_ponto(ponto, 0, quadricula), \

move\_ponto(ponto,quadricula, 0))

stroke("black")

1. Desenhar os tiros do computador como cruzes vermelhas no tabuleiro esquerdo.

# desenha os tiros do computador

for lin, col, tiro in lista\_computador\_tiros:

ponto = (ox1 + lin\*quadricula, oy + col\*quadricula)

stroke("red")

line(ponto, move\_ponto(ponto,quadricula, quadricula))

line(move\_ponto(ponto, 0, quadricula), move\_ponto(ponto,quadricula, 0))

stroke("black")

Foi utilizada um função auxiliar para calculo das deslocações de pontos:

# desloca um ponto

def move\_ponto(ponto, x, y):

return (ponto[0]+x, ponto[1]+y)

# Inicialização do jogo

A função setup() cria a janela gráfica com os dois tabuleiros e atualiza o número de queques presentes nesses tabuleiros, utilizando a função auxiliar soma\_queques().

No final determina, aleatoriamente, a quem cabe iniciar o jogo, através da variável global agora\_joga.

# inicialização do jogo

def setup():

global tab\_computador, tab\_humano, \

quequesC\_nao\_comidos, quequesH\_nao\_comidos, jogador

# dimensão da janela

size(LARGURA, ALTURA)

title("Jogo dos queques")

# cria os tabuleiros com os queques

tab\_humano = cria\_tabuleiro(LINHAS, COLUNAS, TABULEIRO)

tab\_computador = cria\_tabuleiro(LINHAS, COLUNAS, TABULEIRO)

if not tab\_computador or not tab\_humano:

print("Não foi possível criar o tabuleiro")

exit()

else:

quequesH\_nao\_comidos = quequesC\_nao\_comidos = soma\_queques(FORNADA)

# determina quem joga primeiro (computador - 0 / humano - 1)

jogador = random.randint(0,1)

A função auxiliar soma\_queques() retorna o valor do somatório dos produtos entre o número de pratos (n) e o número de queques (q) em cada um desses pratos. O valor deste somatório é, afinal, o número total de queques contidos em FORNADA

def soma\_queques(fornada):

soma = 0

for n, q in fornada:

soma += n\*q

return soma

# Iterando através das jogadas – o ciclo de jogo

O ciclo de jogo é implementado em p5 através da função draw() seguindo o seguinte algoritmo:

1. Desenha a UI do jogador;
2. Caso seja o computador a jogar, determina jogada aleatória (a jogada do jogador humano é realizada no evento do rato);
   * Se o o computador falhar, muda para o jogador humano;
3. Verifica se algum jogador venceu

# ciclo de jogo

def draw():

global tab\_computador, tab\_humano, quequesC, quequesH, jogador, \

lista\_humano\_tiros, lista\_computador\_tiros

# desenha o interface do jogador

desenhaUI()

# jogada do computador

if jogador == 0:

# tiro aleatório

l = random.randint(0, LINHAS-1)

c = random.randint(0, COLUNAS-1)

tiro = tab\_humano[l][c]

lista\_computador\_tiros.append((l, c, tiro))

#print('Jogada do computador: '+ str(l) + " : " + str(c))

# se falhou muda de jogador

if tiro == 0:

jogador = 1 # passa para o jogador humano

else:

quequesH -= 1

# verifica se alguem ganhou

if quequesC == 0:

print("Parabens! É o vencedor.")

exit()

elif quequesH == 0:

print("Perdeu... O computador venceu.")

exit()

# Interação através do rato

Neste jogo, optou-se por utilizar o rato para selecionar diretamente a quadrícula de forma a tornar a interação mais intuitiva. Em p5 a função que recebe os eventos de premir o rato é mouse\_pressed().

O algoritmo é simples:

1. Se for o jogador humano a jogar, determina o valor da quadrícula através das coordenadas do rato (mouse\_x, mouse\_y)
2. Realiza o tiro do jogador e verifica se acertou em algum queque...
3. Se não acertou, então troca para o outro jogador;
4. Se acertou, então decrementa o número de queques que faltam adivinhar.

# evento do rato - jogada do jogador humano

def mouse\_pressed():

global tab\_computador, lista\_humano\_tiros, jogador, quequesC

# jogada do humano

if jogador == 1:

tiro = get\_rato(mouse\_x, mouse\_y)

if tiro:

l = tiro[0]

c = tiro[1]

alvo = tab\_computador[l][c]

lista\_humano\_tiros.append((l,c, alvo))

# se falhou muda de jogador

if alvo == 0:

jogador = 0 # passa para o jogador computador

else:

quequesC -= 1

Utilizou-se uma função auxiliar para determinar a quadrícula a partir das coordenadas do rato.

def get\_rato(x, y):

quadricula = min( (LARGURA-40) // COLUNAS // 2, (ALTURA -50) // LINHAS )

ox2 = LARGURA // 2 + 20

oy = (ALTURA - LINHAS \* quadricula) // 2

c = (x-ox2) // quadricula

l = (y-oy) // quadricula

if l>=0 and l<LINHAS and c>=0 and c<COLUNAS:

return (l, c)

else:

return False

A função retorna um par de coordenadas (linha,coluna) caso o rato tenha sido premido dentro da quadrícula direita, ou False caso contrário.

Não se esqueça de no final adicionar a instrução que executa o p5.

if \_\_name\_\_ == '\_\_main\_\_':

run()

# Desafio final

Nesta solução, fez-se a restrição de os tabuleiros estarem apenas na horizontal. Acrescente a possibilidade de o tabuleiro ter tabuleiros também na vertical. E a impossibilidade de os tabuleiros estarem juntos, obrigando a, pelo menos, uma casa a separar cada dois tabuleiros.

|  |  |
| --- | --- |
| **Share with person** | Partilhe o **seu** jogo! |