Fundação CECIERJ - Vice Presidência de Educação Superior a Distância
Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação com Interfaces Gráficas
AD1 1^o semestre de 2017.
Professores: Mario Benevides e Paulo Roma

---

1 Touros e Vacas (Bulls and Cows)

1.1 Objetivo

Escreva um programa Python usando OO, que implemente o jogo conhecido como "Touros e Vacas":
http://www.microsoft.com/pt-br/store/p/bulls-and-cows-free/9nblggh09zkl

1.2 Descrição do Jogo

Este é um jogo de adivinhação de palavras.

- Há dois jogadores: Jogador_1 e Jogador_2.
- No início, é gerada uma palavra secreta e o Jogador_1 deve tentar adivinhá-la, fornecendo uma série de palavras. Quando o Jogador_1 fornecer uma palavra que não possua ao menos uma nova letra na posição correta, ele passa a vez ao Jogador_2. O vencedor é aquele que conseguir adivinhar a palavra secreta.
- As palavras devem ter quatro ou cinco letras. O número de letras não pode mudar, uma vez que o jogo tenha começado.
- Inicialmente, o Jogador_1 vê apenas o número de letras da palavra secreta, neste exemplo, **prato**.

  *****

- O Jogador$_1$ deve continuar escolhendo palavras de cinco letras, por exemplo, **garfo**, que deverá aparecer na tela como:

  *****
  garfo

- Quando o Jogador$_1$ pressionar *enter*, deve ser impresso na tela:

  Touros ****o
  garfo
  Vacas: ar
  Gansos: gf

- Isto significa que a letra **o** está no lugar certo, as letras **ar** ocorrem na palavra mas estão fora do lugar, e as letras **gf** estão erradas, isto é, não ocorrem na palavra.

- O jogo continua com o Jogador$_1$ escolhendo outra palavra de cinco letras, por exemplo, **bravo**:

  Touros: *ra*o
  bravo
  Vacas:
  Gansos: gfbv

1.3 Definições

- Um touro é uma posição na qual uma letra, em uma das palavras chutadas, casa com uma letra da palavra secreta. Uma letra em qualquer outra posição é dita oculta (mesmo que essa letra também seja um touro).

- Uma vaca é uma letra que exista na palavra chutada, mas que ainda apareça oculta em alguma posição da palavra secreta.

- Um ganso é uma letra de alguma palavra chutada, mas que não exista na palavra secreta.

- Se um chute de um jogador contiver um novo touro, e não existirem mais letras ocultas, este jogador é o vencedor (estado WIN).

- Se o chute de um jogador incluir um touro, mas ainda houver letras ocultas, o turno deste jogador prossegue (estado KEEP_TURN); senão, a vez é passada ao outro jogador (estado LOSE_TURN).

- Se o chute de um jogador for uma palavra inválida (não existente no dicionário ou com uma quantidade diferente de letras), o jogador passa a vez (estado INVALID_WORD) e o chute é ignorado (ou seja, nenhuma letra de uma palavra inválida deve ser adicionada às vacas, touros ou gansos.)

Note que uma dada letra pode aparecer tanto como touro ou vaca, ao mesmo tempo. Por exemplo, se a palavra secreta for "posse" e o jogador chutar "poste", então os touros são "pos*e" mas a letra "s" também é uma vaca, porque ainda há uma letra "s" oculta. Note também que um jogador não precisa realmente adivinhar a palavra secreta para vencer. No exemplo acima, o jogador poderia ter vencido chutando a palavra "prova" para obter o touro final, "p".

2 A Classe WordList

A implementação do jogo depende de uma classe chamada WordList, que desempenha o papel de um dicionário e também fornece um método para escolher palavras secretas, aleatoriamente. Esta classe deve ser implementada completamente apenas na AD2.

Estão definidas as seguintes operações (métodos) na classe WordList:

- **def __init__(self, filename)**: constrói uma WordList, a partir de um arquivo de palavras.

- **def check(self, word)**: retorna *true* se a palavra estiver no dicionário, e $false$ caso contrário. Todas as palavras no dicionário estão em letras minúsculas.

- **def generate(self, size)**: retorna a próxima palavra do tamanho especificado, selecionada aleatoriamente pelo objeto WordList. Retorna *None* se o dicionário não contiver palavras do tamanho especificado.

- **def binarySearch(self, alist, word)**: retorna o índice de uma string "word", na lista de palavras "alist", ou -1 se "word"não estiver na

lista. Deve ser executada uma busca binária, haja vista que "alist"está ordenada.

A classe WordList contém uma lista de palavras, inicializada a partir de um arquivo, e um selecionador de elementos aleatórios de uma lista. Geralmente, um objeto WordList deve ser construído no método **main**, e passado à classe principal do jogo (BullsAndCowsGame).

3 O Tipo enumerável Status

Quando um jogador tenta uma palavra, existem vários resultados possíveis que são representados pelas seguintes constantes:

- Status.INVALID_WORD – a tentativa do jogador tem o tamanho errado ou não está no dicionário;
- Status.LOSE_TURN – a tentativa do jogador não revela nenhuma nova letra (touro);
- Status.KEEP_TURN – a tentativa do jogador revela pelo menos uma nova letra (touro);
- Status.WIN – a tentativa do jogador revela uma nova letra e não há mais letras ocultas;
- Status.OVER – indica um erro quando uma tentativa é feita após o término do turno do jogo.

Estas constantes são definidas em um *tipo enumerável*, ou *enum*, chamado Status. Este tipo é implementado como uma classe para você. Não deve ser modificado.

```
##
 # Enumerated type representing possible status after
 # guessing a word in the BullsAndCows game.
 #
class Status(Enum):
  INVALID_WORD = 1
  LOSE_TURN    = 2
```

KEEP_TURN = 3
WIN = 4
OVER = 5

Status deve ser usado da mesma maneira que outras constantes (por exemplo, CrapsGame.READY, CrapsGame.COME_OUT e assim por diante). Eles funcionam como valores primitivos. Variáveis do tipo Status podem ser criadas e comparadas, usando o operador ==. No entanto, como os valores Status não são números inteiros, só é possível compará-los uns aos outros, e não a inteiros arbitrários.

4 Especificação da Classe BullsAndCowsGame

Sua classe deve incluir a seguinte definição de constante:

```
##
 #  Character to use as a placeholder for the hidden
 #  characters when displaying the "bulls".
 #
PLACEHOLDER = '*'
```

Existem dois construtores e oito métodos públicos. Todos os métodos que retornam strings devem retornar apenas letras minúsculas.

- **def __init__ (self, wlist, arg)**: Constrói um novo jogo, usando o objeto WordList fornecido, como um dicionário.

  1) Se **arg** for do tipo inteiro, ele especifica o tamanho das palavras a serem utilizados no jogo. O jogo começará com a primeira palavra do tamanho especificado, que é gerada a partir da lista de palavras.

  2) Se **arg** for do tipo string, ele especifica a palavra secreta. O jogo utilizará o tamanho desta string como o comprimento das palavras válidas. A palavra secreta fornecida deve fazer parte do dicionário. O propósito deste construtor é simplificar os testes.

- **def startNewRound(self)**: Inicia uma nova rodada do jogo, usando a próxima palavra gerada pelo objeto WordList deste jogo.

- **def isOver(self)**: Retorna *true* se a rodada atual terminou (ou seja, não há mais letras ocultas).

- **def getSecretWord(self)**: Retorna a palavra secreta para esta rodada. A string retornada deve utilizar letras minúsculas.
- **def getAllGuessedLetters(self)**: Retorna uma string contendo todas as letras de todas as palavras chutadas na rodada atual, sem duplicatas, na ordem em que ocorreram pela primeira vez nas tentativas dos jogadores. A string retornada deve estar em letras minúscula.
- **def getBulls(self)**: Retorna a seqüência com os touros revelados em suas posições e outras letras substituídas por PLACEHOLDER. Todas as letras na string retornada são minúsculas.
- **def getGeese(self)**: Retorna uma string contendo todas as letras utilizadas na rodada atual, e que não ocorrem na palavra secreta. Todas as letras na string retornada são minúsculas, sem duplicatas, e a ordem das letras é a mesma que em getAllGuessedLetters().
- **def getCows(self)**: Retorna uma string contendo todas as letras utilizadas na rodada atual, e que ocorrem na palavra secreta. Todas as letras na string retornada são minúsculas, sem duplicatas, e a ordem das letras é a mesma que em getAllGuessedLetters().
- **def guess(self, word)**: Processa a tentativa de um jogador e retorna o estado apropriado. Este método não é sensível a letras maiúsculas e minúsculas.

5 A Classe TextUI

Depois de ter o jogo funcionando em alguma extensão, experimente-o criando uma interface de usuário TextUI. Este código pode ser modificado à vontade, já que os outros componentes não dependem dele.

A sua implementação deve possuir um método *main*, que irá instanciar um objeto TextUI. Para executar o método *main*, é necessaŕio que haja um arquivo de dicionário, **palavras.txt**, no diretório do projeto. Este arquivo está ordenado e possui palavras de quatro e cinco letras. O nome deste arquivo deve ser passado como argumento na chamada do jogo:

```
[cascavel:~/cederj/AD/2017-1] BullsAndCowsGame.py palavras.txt
```

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package BullsAndCowsGame
#
#  Bulls and Cows (also known as Cows and Bulls or Pigs and Bulls or
#                  Bulls and Cleots)
#  is an old code-breaking paper and pencil game for two players,
#  predating the similar commercially marketed board game Mastermind.
#
#  @author Paulo Roma
#  @since 12/07/2016
#  @see http://en.wikipedia.org/wiki/Bulls_and_cows
#  @see http://www.python-course.eu/sets_frozensets.php
#  @see https://www.dicio.com.br/palavras-com-cinco-letras/
#

import sys
import codecs
from enum import Enum
from random import randint, random, choice, shuffle
        ......
        ......
        ......

def main(argv=None):
        if argv is None:
                argv = sys.argv
        ......
        ......
        ......
        game = BullsAndCowsGame(wlist, size)
        ui = TextUI(game)
        ui.runUI()

if __name__=="__main__":
        sys.exit(main())
```

- **def __init__ (self, givenGame)**: construtor. Salva o objeto jogo passado como argumento.
- **def runUI(self)**: Interface principal. Executa o laço que testa a palavra chutada por um jogador, e decide quem joga depois. Só termina ao final do jogo, ou se for interrompido pelo teclado.
- **def display(self)**: Imprime o estado do jogo (Touros, Vacas e Gansos).
- **def playAgain(self)**: pergunta ao jogador se quer começar um novo turno do jogo (nova rodada ou partida).

6 A Classe BullsAndCowsTest

Implemente um conjunto de testes, na forma de uma classe, para validar a sua implementação. Seus casos de teste devem executar todos os métodos implementados.

- Cada caso de teste se concentra em apenas uma operação ou seqüência de operações.
- Há uma mensagem curta indicando o resultado esperado.
- Existem assertivas da forma:

  assertEqual(expected_value, actual_value, msg), do módulo unittest.

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package BullsAndCowsTest
#
#  Class for testing the Bulls and Cows game.
#
#  @author Paulo Roma
#  @since 18/01/2017
#  @see https://docs.python.org/2/library/unittest.html
#
```

```
from BullsAndCowsGame import BullsAndCowsGame
from BullsAndCowsGame import WordList
from BullsAndCowsGame import Status
import sys
import unittest

##
 # Class for testing for certain aspects of the behavior of
 # BullsAndCowsGame.
 #
 #
class BullsAndCowsTest(unittest.TestCase):
  ##
   # Do-nothing word list to use for testing.
   #
  wordList = WordList("palavras.txt")

  def test_Bulls(self):
    msg = "If 'capaz' has been guessed and secret word is 'caixa',\
           getBulls should return 'ca***'"
    game = BullsAndCowsGame(self.wordList, "caixa")
    game.guess("capaz")
    self.assertEqual(game.getBulls(), "ca***", msg)

  ....
  ....
  ....

if __name__=="__main__":
   unittest.main()
```

7 Recomendações

O dicionário de palavras é garantido estar com letras minúsculas, mas o chute do jogador (e a palavra para o segundo construtor) podem não estar. A coisa esperta a fazer é provavelmente converter internamente todas as strings para minúsculas.

Você deve criar dois arquivos (com 500 palavras cada um), palavras4.txt e palavras5.txt, copiando e colando, a partir do site abaixo, e executar o seguinte script bash, para gerar o seu arquivo ordenado de palavras:

```
#!/bin/sh

# https://www.dicio.com.br/palavras-com-cinco-letras/
# https://www.dicio.com.br/palavras-com-quatro-letras/

# utf-8 sort
export LC_ALL=C

cat palavras4.txt > aaa
cat palavras5.txt >> aaa

sort aaa > palavras.txt
rm aaa
```

Note que algumas palavras são acentuadas, e a ordenação precisa funcionar independentemente disso. O script acima assume strings codificadas em UTF-8.

Sua classe precisará armazenar a palavra secreta, e provavelmente manter uma segunda string com os "touros" mostrados corretamente, como nas palavras exibidas, tais como "*ra*o", no exemplo da introdução.

É provavelmente mais simples NÃO tentar armazenar as vacas e os gansos como variáveis de instância. Mantendo apenas uma string contendo todas as letras utilizadas até o momento (por exemplo, a string a ser retornada pelo método getAllGuessedLetters ()), pode-se criar uma seqüência de vacas ou gansos sempre que necessário.

8 Sugestões para Desenvolver o Projeto

1. Como de costume, pode-se criar *stubs* para os métodos e construtores (apenas declarações, sem código).

2. Comece com o segundo construtor (aquele que especifica a primeira palavra) e ignore o objeto WordList inicialmente (isto é, permita palavras inválidas nas tentativas).

3. Implemente getSecretWord().

4. Comece com uma implementação simples de guess() que apenas registra as tentativas, e implemente getAllGuessedLetters(). Escreva casos de teste para garantir que tudo está funcionando corretamente até agora. (Não se preocupe com o valor de retorno de guess(), basta retornar Status.INVALID_WORD).

5. Modifique guess() para que, se uma palavra chutada for do tamanho errado, retorne Status.INVALID_WORD, sem processar a tentativa.

6. Implemente getGeese(). Escreva casos de teste.

7. Modifique guess() para tratar os touros e implementar getBulls(). Teste-o.

8. Implemente getCows().

9. Modifique guess() e implemente isOver() para que, se não houver mais letras ocultas, IsOver() retorne *true*. Se guess() é chamado quando isOver() já é *true*, ele deve retornar Status.OVER, sem processar a tentativa.

10. Modifique guess() para retornar apropriadamente Status.KEEP_TURN or Status.LOSE_TURN.

11. Use um objeto WordList para rejeitar tentativas que não estão no dicionário. Implemente startNewRound().

12. Use exceções para condições de erro, tais como: arquivo inexistente, término do programa pelo teclado, palavra inválida, tipo não previsto.

13. Todos os métodos e classes devem estar devidamente comentados, no estilo Doxygen. Use um estilo consistente para identação e formatação.

14. Defina como atributo de classe ou objeto (instância) apenas o que for estritamente necessário.

9 Correção

A classe BullsAndCowsTest que está sendo entregue deve testar alguns aspectos mais simples do jogo. Porém, você deve testar todo o resto. Perceba que rodar a interface do usuário não é provavelmente a melhor maneira de testar o programa. Em particular, quando nós testarmos o seu código, não rodaremos a apenas UI, mas sim, focaremos nas especificações fornecidas.

10 O que Entregar?

Deve ser enviado o código fonte em mídia digital e o programa deverá rodar em ambiente Linux com python 2 e python 3. Foi mais fácil explicar o jogo como um todo, mas a classe WordList só precisará ser implementada completamente na AD2. Por enquanto, basta escolher uma palavra secreta arbitrariamente e aceitar qualquer chute com o mesmo número de letras.

A AD2 substituirá a classe TextUI por uma interface gráfica, e implementará a busca binária para verificar se uma dada palavra faz parte do dicionário.

Perceba que seu jogo deve ser testado pelo tutor. Logo, a palavra secreta deve ser impressa na tela, para que ele possa testá-lo mais facilmente.

A implementação completamente comentada possui 500 linhas de código. No entanto, não fornecemos código nesse curso. Entenda que não existe gabarito de programa. Se você não fizer a sua própria implementação, ninguém vai fazer por você...