



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

AD1: Touros e Vacas (Bulls and Cows)

1 Objetivo

Escreva um programa Python usando OO, que implemente o jogo conhecido como “Touros e Vacas” ¹.

Nesse trabalho será utilizada a classe Enum, disponível a partir da versão 3.4 do Python. Para versões anteriores, é necessário instalar um pacote adicional ²: “pip install enum34”.

Além disso, como em português várias palavras são acentuadas, é fundamental trabalhar com strings Unicode ^{3 4}, o que é muito mais fácil programando em Python 3, porque o tipo **str** já é unicode, nesse caso.

Em Python 2, no entanto, um arquivo no formato UTF-8 (*Unicode Transformation Format with 8 bits*) deve ser aberto, da seguinte forma:

```
1 import codecs
2 f = codecs.open(filename, 'r', 'utf-8')
```

Assim, em Python 2, cada string lida de um arquivo no formato UTF-8 ou via teclado, deve ser decodificada pelo método **decode** da classe **str**:

s_unicode = "uma string qualquer".decode("utf-8").

1.1 Descrição do Jogo

Este é um jogo de adivinhação de palavras.

¹<http://www.microsoft.com/pt-br/store/p/bulls-and-cows-free/9nblggh09zkl>

²<https://pypi.org/project/enum34/>

³<https://learn.rmotr.com/python/understanding-unicode-in-python/strings-and-unicode/encodings-and-utf-8>

⁴<https://medium.com/@agustinb/introduction-to-unicode-and-utf-8-in-python-9e7a844edddd>

- Há dois jogadores: Jogador₁ e Jogador₂.
- No início, é gerada uma palavra secreta e o Jogador₁ deve tentar adivinhá-la, fornecendo uma série de palavras. Quando o Jogador₁ fornecer uma palavra que não possua ao menos uma nova letra na posição correta, ele passa a vez ao Jogador₂. 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 Jogador₁ vê apenas o número de letras da palavra secreta, neste exemplo, **prato**.

- O Jogador₁ deve continuar escolhendo palavras de cinco letras, por exemplo, **garfo**, que deverá aparecer na tela como:

garfo

- Quando o Jogador₁ 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 Jogador₁ escolhendo outra palavra de cinco letras, por exemplo, **bravo**:

Touros: *ra*o
 bravo

Vacas:

Gansos: gfbv

1.2 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`:

```

1 def __init__(self, filename):
2     """constrói uma WordList, a partir de um arquivo de palavras."""
3
4 def check(self, word):
5     """retorna True se a palavra estiver no dicionário, e False caso contrário.
6     Todas as palavras no dicionário estão em letras minúsculas."""
7
8 def generate(self, size):
9     """retorna a próxima palavra do tamanho especificado, selecionada
10    aleatoriamente pelo objeto WordList.
11    Retorna None se o dicionário não contiver palavras do tamanho especificado."""
12
13 def binarySearch(self, alist, word):
14     """retorna o índice de uma string "word", na lista de palavras "alist",
15     ou -1 se "word" não estiver na lista.
16     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* ⁵, chamado `Status`. Este tipo é implementado como uma classe para você. Não deve ser modificado.

```

1 from enum import Enum
2 ##
3 # Enumerated type representing possible status after
4 # guessing a word in the BullsAndCows game.
5 #
6 class Status(Enum):
7     INVALID_WORD = 1
8     LOSE_TURN    = 2
9     KEEP_TURN    = 3
10    WIN           = 4
11    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:

```

1 ##
2 # Character to use as a placeholder for the hidden
3 # characters when displaying the "bulls".
4 #
5 PLACEHOLDER = '*'

```

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

⁵<https://docs.python.org/3/library/enum.html>

```

1 def __init__(self, wlist, arg):
2     """Constrói um novo jogo, usando o objeto WordList fornecido,
3         como um dicionário."""
4
5     1) Se arg for do tipo inteiro, ele especifica o tamanho das palavras
6         a serem utilizados no jogo.
7     O jogo começará com a primeira palavra do tamanho especificado,
8         que é gerada a partir da lista de palavras.
9
10    2) Se arg for do tipo string, ele especifica a palavra secreta. O jogo
11        utilizará o tamanho desta string como o comprimento das palavras válidas.
12    A palavra secreta fornecida deve fazer parte do dicionário.
13    O propósito deste construtor é simplificar os testes.
14
15 def startNewRound(self):
16     """Inicia uma nova rodada do jogo, usando a próxima palavra
17         gerada pelo objeto WordList deste jogo."""
18
19 def isOver(self):
20     """Retorna True se a rodada atual terminou
21         (ou seja, não há mais letras ocultas)."""
22
23 def getSecretWord(self):
24     """Retorna a palavra secreta para esta rodada.
25         A string retornada deve utilizar letras minúsculas."""
26
27 def getAllGuessedLetters(self):
28     """Retorna uma string contendo todas as letras de todas as palavras
29         chutadas na rodada atual, sem duplicatas, na ordem em que ocorreram
30         pela primeira vez nas tentativas dos jogadores.
31         A string retornada deve estar em letras minúscula."""
32
33 def getBulls(self):
34     """Retorna a sequência com os touros revelados em suas posições
35         e outras letras substituídas por PLACEHOLDER.
36         Todas as letras na string retornada são minúsculas."""
37
38 def getGeese(self):
39     """Retorna uma string contendo todas as letras utilizadas na rodada atual,
40         e que não ocorrem na palavra secreta.
41         Todas as letras na string retornada são minúsculas, sem duplicatas,
42         e a ordem das letras é a mesma que em getAllGuessedLetters()."""
43
44 def getCows(self):
45     """Retorna uma string contendo todas as letras utilizadas na rodada atual,
46         e que ocorrem na palavra secreta.
47         Todas as letras na string retornada são minúsculas, sem duplicatas,
48         e a ordem das letras é a mesma que em getAllGuessedLetters()."""
49
50 def guess(self, word):
51     """Processa a tentativa de um jogador e retorna o estado apropriado.
52         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 ⁶. Para executar o método *main*, é necessário que haja um arquivo de dicionário, **palavras.txt** ⁷, 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

```
1  #!/usr/bin/env python
2  # coding: UTF-8
3  #
4  ## @package BullsAndCowsGame
5  #
6  # Bulls and Cows (also known as Cows and Bulls or Pigs and Bulls or
7  # Bulls and Cleots)
8  # is an old code-breaking paper and pencil game for two players,
9  # predating the similar commercially marketed board game Mastermind.
10 #
11 # @author Paulo Roma
12 # @since 12/07/2016
13 # @see http://en.wikipedia.org/wiki/Bulls_and_cows
14 # @see http://www.python-course.eu/sets_frozensets.php
15 # @see https://www.dicio.com.br/palavras-com-cinco-letras/
16 #
17
18 import sys
19 import codecs
20 from enum import Enum
21 from random import randint, random, choice, shuffle
22     .....
23     .....
24     .....
25
26 def main(argv=None):
27     if argv is None:
28         argv = sys.argv
29     .....
30     .....
31     .....
32     game = BullsAndCowsGame(wlist, size)
33     ui = TextUI(game)
34     ui.runUI()
35
36 if __name__=="__main__":
37     sys.exit(main())
```

⁶<http://orion.lcg.ufrj.br/python/ADs/TextUI.py>

⁷<http://orion.lcg.ufrj.br/python/ADs/palavras.txt>

```

1 def __init__(self, givenGame):
2     """construtor. Salva o objeto jogo passado como argumento."""
3
4 def runUI(self):
5     """Interface principal. Executa o laço que testa a palavra chutada por
6         um jogador, e decide quem joga depois.
7         Só termina ao final do jogo, ou se for interrompido pelo teclado."""
8
9 def display(self):
10    """Imprime o estado do jogo (Touros, Vacas e Gansos)."""
11
12 def playAgain(self):
13    """pergunta ao jogador se quer começar um novo turno do jogo
14        (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 sequência de operações.
- Há uma mensagem curta indicando o resultado esperado.
- Existem assertivas do módulo unittest, da forma:

```

assertEqual(expected_value, actual_value, msg),
assertNotEqual(expected_value, actual_value, msg),
assertTrue(expr, msg),
assertFalse(expr, msg),
assertIs(first, second, msg),
assertIsNot(first, second, msg),
assertIsNone(expr, msg),
assertIsNotNone(expr, msg),
assertIn(first, second, msg),
assertNotIn(first, second, msg),
assertIsInstance(obj, cls, msg),
assertIsNotInstance(obj, cls, msg),
assertRaises(exc, fun, *args, **kwds),
assertRaisesRegex(exception, regex, callable, *args, **kwds).

```

```

1  #!/usr/bin/env python
2  # coding: UTF-8
3  #
4  ## @package BullsAndCowsTest
5  #
6  # Class for testing the Bulls and Cows game.
7  #
8  # @author Paulo Roma
9  # @since 18/01/2017
10 # @see https://docs.python.org/2/library/unittest.html
11 #
12
13 from BullsAndCowsGame import BullsAndCowsGame
14 from BullsAndCowsGame import WordList
15 from BullsAndCowsGame import Status
16 import sys
17 import unittest
18
19 ##
20 # Class for testing for certain aspects of the behavior of
21 # BullsAndCowsGame.
22 #
23 #
24 class BullsAndCowsTest(unittest.TestCase):
25     ##
26     # Do-nothing word list to use for testing.
27     #
28     wordList = WordList("palavras.txt")
29
30     def test_Bulls(self):
31         msg = "If 'capaz' has been guessed and secret word is 'caixa',\
32             getBulls should return 'ca***'"
33         game = BullsAndCowsGame(self.wordList, "caixa")
34         game.guess("capaz")
35         self.assertEqual(game.getBulls(), "ca***", msg)
36
37     ....
38     ....
39     ....
40
41 if __name__=="__main__":
42     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`⁸ e `palavras5.txt`⁹, copiando e colando, a partir do site no rodapé, e executar o seguinte script bash, para gerar

⁸<https://www.dicio.com.br/palavras-com-quatro-letras/>

⁹<https://www.dicio.com.br/palavras-com-cinco-letras/>

o seu arquivo ordenado de palavras:

```
1 #!/bin/sh
2
3 # utf-8 sort
4 export LC_ALL=C
5
6 cat palavras4.txt > aaa
7 cat palavras5.txt >> aaa
8
9 sort aaa > palavras.txt
10 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 sequência 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 indentação 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 3 (recomendável que rode em python 2 também). 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 binária¹⁰ 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...

¹⁰<https://www.lcg.ufrj.br/cwdc/11-python/showCodePython.php?f=sort>