

Estrutura de Dados em Python: objetos como estratégia

Bernardo Alves Furtado

IDP – Inteligência Artificial – Aula 3 – 13 de novembro, 2020

Overview

1 Estrutura

2 **class:** Objetos

- Deque de cartas

3 Interação

- De guerreiros à exércitos e estratégia

4 Aplicações

- Carros Ecológicos
- Gerador de Redes Adversárias: talento e sorte

5 Inteligência Artificial

Introdução + materiais

- Bernardo Alves Furtado
- PhD University Utrecht – Doutor Economia Cedeplar/UFMG
- Pesquisador no Ipea (2009) e CNPq (2014)
- PDF + Código + Links:
github.com/bafurtado/estrutura_dados_idp

Objetivo

Ilustrar + Motivar uso estrutura **objetos**

em #Python  [1]

Atributos – variáveis

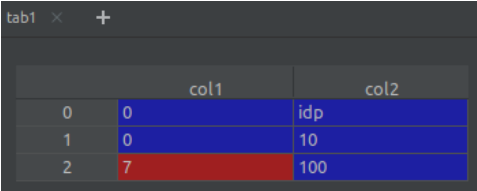
- `int: x = 7`
- `float: y = 10.0`
- `str: text = "data science"`
- 🔍: use `type(x)` para descobrir o tipo da variável em `#python`

Atributos – variáveis

- `int: x = 7`
- `float: y = 10.0`
- `str: text = "data science"`
- 🔍 use `type(x)` para descobrir o tipo da variável em #python
- `list: x = [0, 0, 7, "idp", 100]`
- `tuple: x = 0, 0, 7, "idp", 100`

Atributos II – tabelas

```
import pandas as pd  
tab1 = pd.DataFrame("col1": [0, 0, 7],  
                     "col2": ["idp", "10", "100"])
```



The screenshot shows a Jupyter Notebook interface with a cell titled 'tab1'. The cell contains a pandas DataFrame with two columns, 'col1' and 'col2', and three rows. The first row has values 0 and 'idp'. The second row has values 0 and '10'. The third row has values 7 and '100'. The cell '7' in the third row is highlighted in red.

| | col1 | col2 |
|---|------|------|
| 0 | 0 | idp |
| 1 | 0 | 10 |
| 2 | 7 | 100 |

Atributos III – matrizes n-dimensões

```
mat1 = [[ [0, 0, 7], [0, 7, 0], [0, 6, 9], [6, 0, 9]],  
         [ [0, 1, 0], [7, 0, 0], [0, 7, 0], [0, 0, 7]],  
         [ [1, 0, 0], [0, 0, 99], [0, 0, 6], [0, 0, 9]],  
         [ [0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 0, 1]]]
```

| | 0 | 1 | 2 | 3 |
|---|-----------|------------|-----------|-----------|
| 0 | [0, 0, 7] | [0, 7, 0] | [0, 6, 9] | [6, 0, 9] |
| 1 | [0, 1, 0] | [7, 0, 0] | [0, 7, 0] | [0, 0, 7] |
| 2 | [1, 0, 0] | [0, 0, 99] | [0, 0, 6] | [0, 0, 9] |
| 3 | [0, 0, 1] | [0, 1, 0] | [1, 0, 0] | [0, 0, 1] |

Atributos – Dicionários

```
dic1 = {"chave1": "valor1",  
        "chave2": 100,  
        "Professor": "Furtado"  
        "lista1": [0, 0, 7]  
        "total": 100  
        "Promoção": "idp"}
```



Métodos – funções

```
def soma(x, y):  
    return x + y
```

```
soma(90, 9)
```

Definição

class: estrutura permite criar **objetos** contém ao mesmo tempo "atributos" e "métodos"

Definição

class: estrutura permite criar **objetos** contém ao mesmo tempo "atributos" e "métodos"

```
class Dog:  
    pass
```

Definição

class: estrutura permite criar **objetos** contém ao mesmo tempo "atributos" e "métodos"

```
class Dog:  
    pass
```

```
jon_snow = Dog()  
<__main__.Dog at 0x7fb9a8c008e0>
```

Lâmpada Multicolorida

```
class Lampada:
    def __init__(self):
        self.status = 0
        self.cores = [Verde, Vermelho, Azul,
                      Amarelo]
```

Lâmpada Multicolorida

```
class Lampada:
    def __init__(self):
        self.status = 0
        self.cores = [Verde, Vermelho, Azul,
                      Amarelo]
    def acende(self):
        resposta = self.cores[self.status % 4]
        self.status += 1
```

PIX



```
class Conta:
    def __init__(self, id):
        self.saldo = 0
```


PIX



```
class Conta:
    def __init__(self, id):
        self.saldo = 0
    def movimenta(self, quantia):
        self.saldo += quantia
```

Deque de cartas

Carta

```
class Carta:
    naipes = [Paus, Ouros, Copas, Espadas]
    ranks = [As, 2, 3, 4, 5, 6, 7,
              8, 9, 10, Valete, Rainha, Rei]
    def __init__(self, naipe, rank):
        self.naipe = naipe
        self.rank = rank
```

Carta

```
class Carta:
    naipes = [Paus, Ouros, Copas, Espadas]
    ranks = [As, 2, 3, 4, 5, 6, 7,
              8, 9, 10, Valete, Rainha, Rei]
    def __init__(self, naipe, rank):
        self.naipe = naipe
        self.rank = rank
    def __eq__(self, outra):
        return self.naipe == outra.naipe and
               self.rank == outra.rank
```

Deque de cartas

Baralho

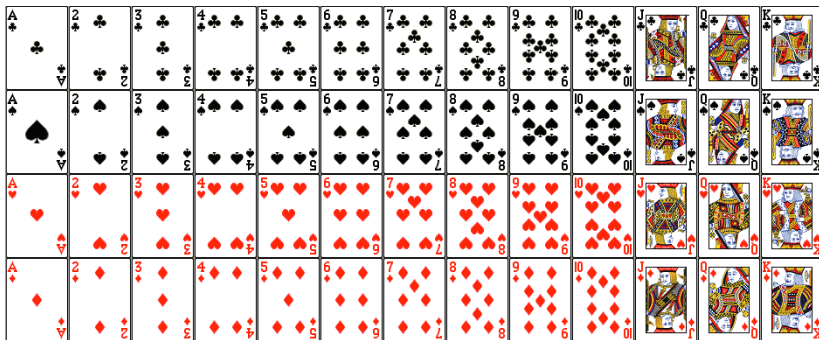
```
class Baralho:
    def __init__(self):
        self.cartas = list()
        for naipe in range(4):
            for rank in range(1, 14):
                carta = Carta(naipe, rank)
                self.cartas.append(carta)
```

Baralho – métodos

```
class Baralho:
    def adiciona_carta(self, carta):
        ...
    def remove_carta(self, carta):
        ...
    def embaralha(self):
        ...
```

Deque de cartas

Cartas



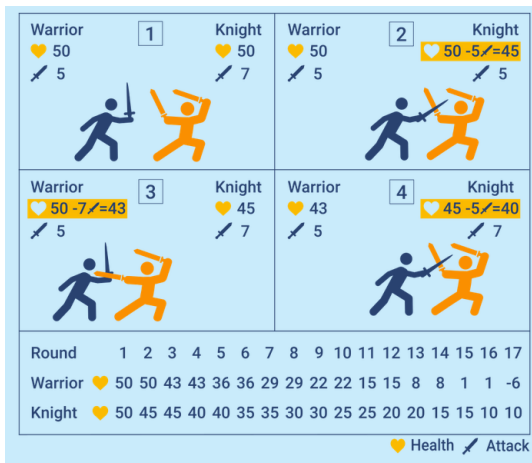
Deque de cartas

BlackJack – 21

BlackJack

De guerreiros à exércitos e estratégia

py.checkio.org [2]



De guerreiros à exércitos e estratégia

Warriors

```
class Warrior:
    def __init__(self):
        self.health = 50
        self.attack = 5
        self.vivo = True
    def ataca(self, outro):
        outro.health -= self.attack
        outro.vivo()
```

De guerreiros à exércitos e estratégia

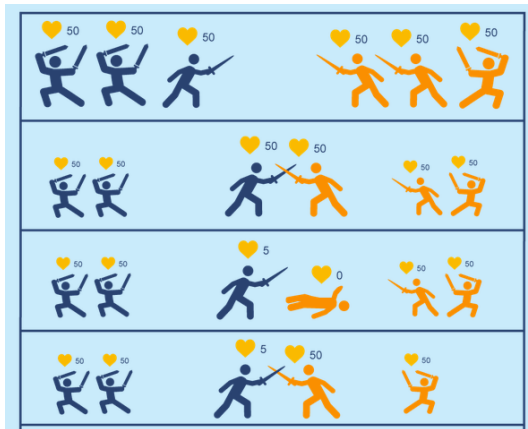
Warriors

```
class Warrior:
    def __init__(self):
        self.health = 50
        self.attack = 5
        self.vivo = True
    def ataca(self, outro):
        outro.health -= self.attack
        outro.vivo()

class Knight(Warrior):
    def __init__(self):
        super().__init__()
        self.attack = 7
```

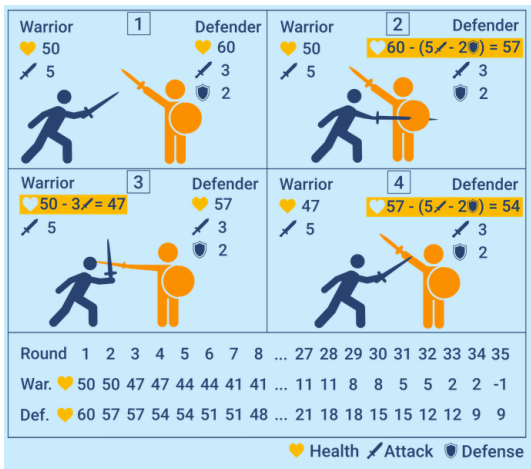
De guerreiros à exércitos e estratégia

Army



De guerreiros à exércitos e estratégia

Defenders



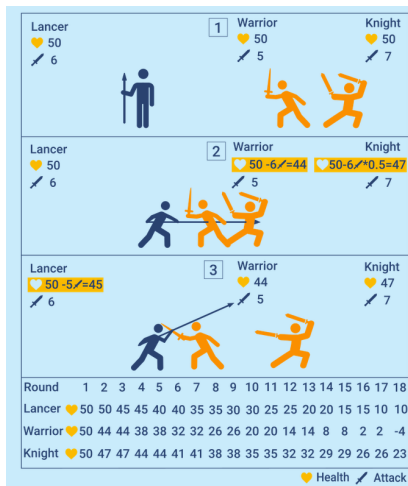
De guerreiros à exércitos e estratégia

Defenders – class

```
class Defender(Warrior):  
    def __init__(self):  
        super().__init__(health=60, attack=3)  
        self.defense = 2  
    def danos(self, ataque):  
        return max(0, ataque - self.defense)
```

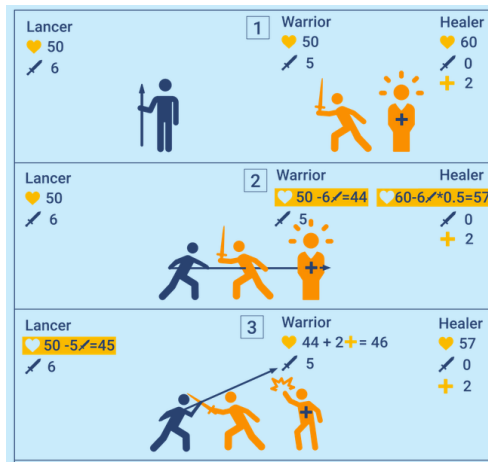
De guerreiros à exércitos e estratégia

Lancers



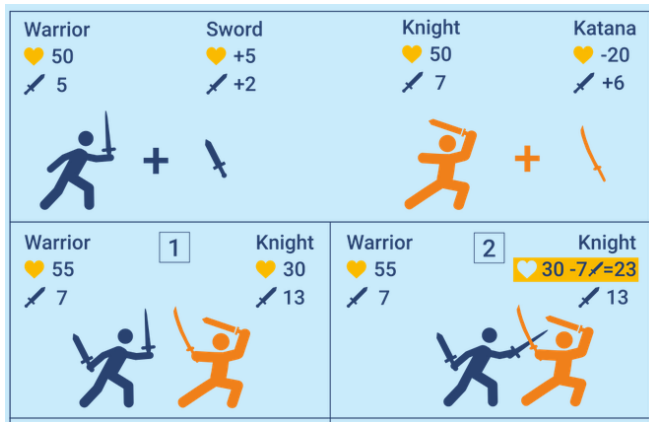
De guerreiros à exércitos e estratégia

Healers



De guerreiros à exércitos e estratégia

Armas



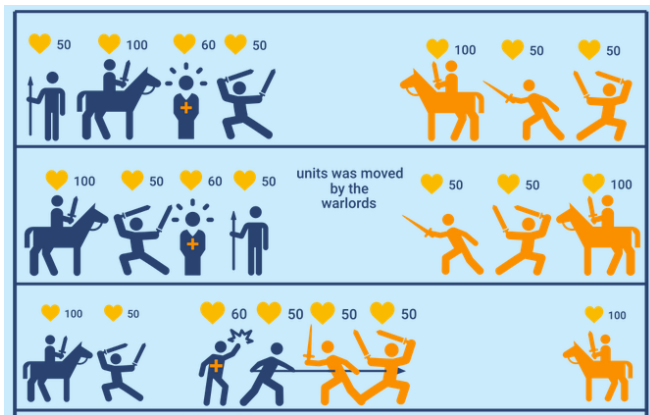
De guerreiros à exércitos e estratégia

Armas – class

```
class Sword(Weapon):  
    def __init__(self):  
        super().__init__(health=5, attack=2)  
class Warrior:  
    def equip_weapon(self, weapon):  
        self.health += weapon.health  
        self.attack += weapon.attack
```

De guerreiros à exércitos e estratégia

Warlord



Mercado Carros Sustentáveis [3]



- Modelo testa **políticas públicas** baseadas em:
 - 1 Emissões CO2 totais
 - 2 Custo financiamento
 - 3 Várias alternativas de políticas (impostos, descontos, multas, restrições)
- **Objetos**: consumidores, firmas, carros (gas., elétricos)
- Inovação: firmas investem \$ melhoria itens dos carros
- Consumidores tem preferências próprias e escolhem **características do carro** (e do mercado)

Escolha



Escolhe 2 critérios

```
class Consumidor:
```

```
    def comprar(self):
```

```
        criterios = [custo, $ energia, mercado,
                     eficiência, emissões, qual.,
                     emoção]
```

Escolha



Escolhe 2 critérios

```
class Consumidor:
```

```
    def comprar(self):
```

```
        criterios = [custo, $ energia, mercado,
                      eficiência, emissões, qual.,
                      emoção]
```

```
class Carro:
```

```
    def selecao(self, emoção, critério1, critério2):
```

```
        # processa seleção 2 critérios consumidor +
        # características próprias do objeto carro
        # (modificadas inovação firma)
```

Exemplo modelo carros sustentáveis

```
INFO:main:Production cost reduced by -441.20
INFO:main:Advertise material. We, at firm 7, have made an investment on gas of 50,000.00
INFO:main:Quality increased by 0.0668
INFO:main:Parameter e -- sold cars emission average -- is 0.2655
INFO:main:Government has paid/received total at this t 3 a net total of $ 2,014,820.57
INFO:main:Emissions at t 3 was 14,915,254.16. Emissions index: 1.0000
INFO:main:Time: 4 -- deliberate pausing for 1 seconds
INFO:main:Advertise material. We, at firm 1, have made an investment on gas of 50,000.00
INFO:main:Energy economy increased by 5.9398
INFO:main:Advertise material. We, at firm 7, have made an investment on gas of 50,000.00
INFO:main:Energy economy increased by 9.7125
INFO:main:Advertise material. We, at firm 5, have made an investment on gas of 195,684.78
INFO:main:Production cost reduced by -247.56
INFO:main:Advertise material. We, at firm 3, have made an investment on gas of 51,338.61
INFO:main:Energy economy increased by 7.8675
INFO:main:Parameter e -- sold cars emission average -- is 0.3774
INFO:main:Government has paid/received total at this t 4 a net total of $ 1,981,454.31
INFO:main:Emissions at t 4 was 15,573,117.07. Emissions index: 1.0441
```

Talento x Sorte [4]



- Sucesso = talento + habilidades (?) ...
- Estudos sugerem que **sorte** é + relevante ...
- Aplicação ADM: remuneração fundos financeiros, por exemplo
- Teste: "jogo **War**" – **estratégia** ou sorte?

Classes – Jogador – Mundo



```
class Jogador:  
    def adiciona_territorio(self):  
    def remove_territorio(self):  
    def define_prioridades(self):  
    def aloca_exercitos(self):  
    def rearranja(self):
```

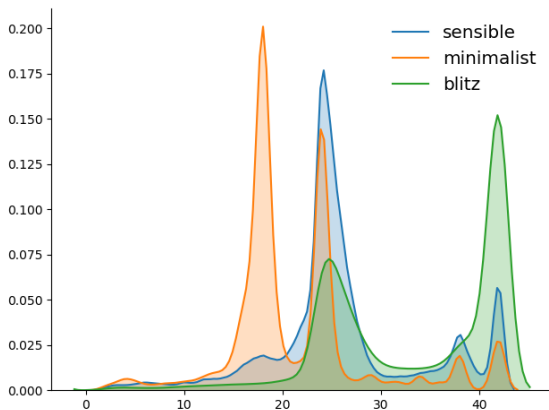

Classes – Jogador – Mundo



```
class Jogador:
    def adiciona_territorio(self):
    def remove_territorio(self):
    def define_prioridades(self):
    def aloca_exercitos(self):
    def rearranja(self):
class Mundo:
    def gera_mapa(self):
    def rodada(self):
```

Gerador de Redes Adversárias: talento e sorte

```
class Jogador: self.strategy
```



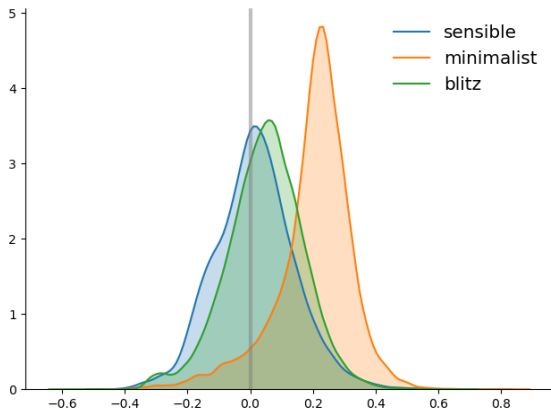
Resultados

| Estratégia | Vitórias | Perc. pts diff. |
|------------|----------|-----------------|
| Blitz | 52.563 | 19 p.p. |
| Sensible | 25.346 | -8 p.p. |
| Minimalist | 22.091 | -11 p.p. |

Table 1: Número vitórias estratégia 100.000 simulações e distância pontos percentuais da média esperada de 1/3.

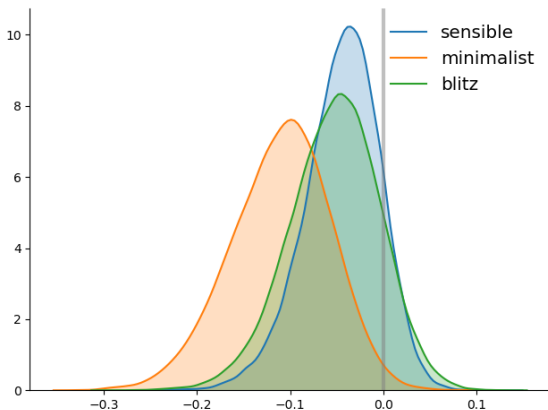
Gerador de Redes Adversárias: talento e sorte

Sorte do vencedor



Gerador de Redes Adversárias: talento e sorte

Sorte dos oponentes



Network

Network Animation

Sistemas Multi-Agentes

Autonomous Agents and Multiagent Systems

Turing Institute – UK

- 1 Teoria dos jogos
- 2 Algoritmos distribuídos
- 3 Ética e redes sociais
- 4 Robótica (enxames)
- 5 Aprendizado de máquinas
- 6 Sistemas multi-agentes
- 7 **Modelos baseados em agentes**
- 8 Aprendizagem por reforço

Sugestões livros

- 1 **Pense Python** – download gratis – Allen Downey –
tradução Luciano Ramalho
 - Para comprar (físico)
- 2 **Think Complexity** by Allen Downey

References I

- [1] “Imagens dessa apresentação são livres – Pixabay – ou do autor: <https://pixabay.com/vectors/snake-python-serpent-green-reptile-312561>.”
- [2] “Ilha Incinerator na plataforma py.checkio.org <https://py.checkio.org/station/incinerator/> .”
- [3] A. van der Vooren and E. Brouillat, “Evaluating CO2 reduction policy mixes in the automotive sector,” *Environmental Innovation and Societal Transitions*, vol. 14, pp. 60–83, Mar. 2015.
- [4] B. A. Furtado, “Contributions of Talent, Perspective, Context and Luck to Success,” *arXiv:2001.00034 [physics]*, Feb. 2020. arXiv: 2001.00034.

Obrigado! Perguntas? Comentários

- Twitter: [@furtadobb](#) [@sejaIDP](#)
- [GitHub/BAFurtado](#)
- <https://py.checkio.org/user/furtadobb/>
- <https://sites.google.com/view/bernardo-alves-furtado/home>
- researchgate.net/profile/Bernardo_Furtado
- furtadobb@gmail.com
- 