Kwarwp Documentation

Release 20.09.a1

Carlo E. T. Oliveira

Oct 05, 2020

Contents:

Jogo para aprendizado de programação.

Author Carlo E. T. Oliveira

Affiliation Universidade Federal do Rio de Janeiro

Homepage Projeto Kwarwp

CHAPTER 1

Descrição

Um jogo de aventura que se joga aprendendo e criando programas na linguagem Python.

Este ambiente facilita a aprendizagem da linguagem Python.

O jogo é dirigido principalmente ao ensino de programação de computadores para jovens e crianças do ensino médio e fundamental.

CHAPTER 2

Documentação

2.1 Tutorial Kwarwp

Tutorial para construir o Kwarwp usando o Vitollino.

Author Carlo E. T. Oliveira

Affiliation Universidade Federal do Rio de Janeiro

Homepage Projeto Kwarwp

2.1.1 Introdução

Este tutorial ensina passo a passo a criação de uma versão do jogo Kwarwp usando a biblioteca Vitollino.

2.1.2 Tutorial

Tutorial Github

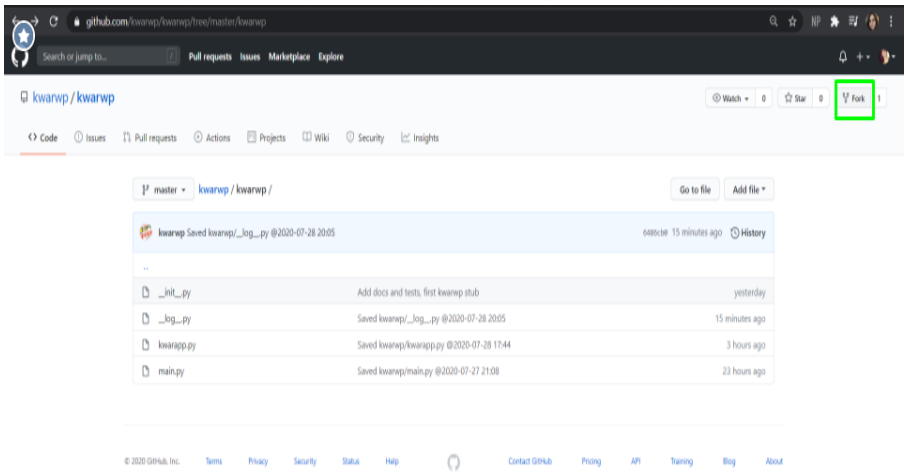
Introdução

No **Github** quando você clica em **fork** no repositório de alguém, ele pega o repositório completo dessa pessoa e copia para sua conta **git**, lá você poderá editar esses arquivos e depois devolver a essa pessoa com as suas edições, se a pessoa aceitar, suas alterações também entram no repositório dele (STACKOVERFLOW, 2016).

Tutorial - Como fazer um fork no github

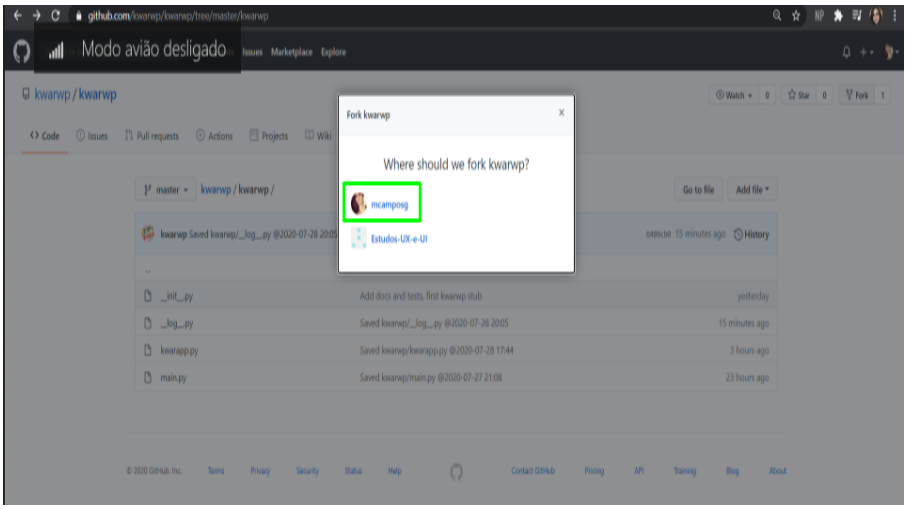
Step 1

- Após logar em sua conta do github acesse o repositorio do kwarwp Link: https://github.com/kwarwp/kwarwp e clique no botão fork localizado a direita (sinalizado em verde na imagem abaixo).



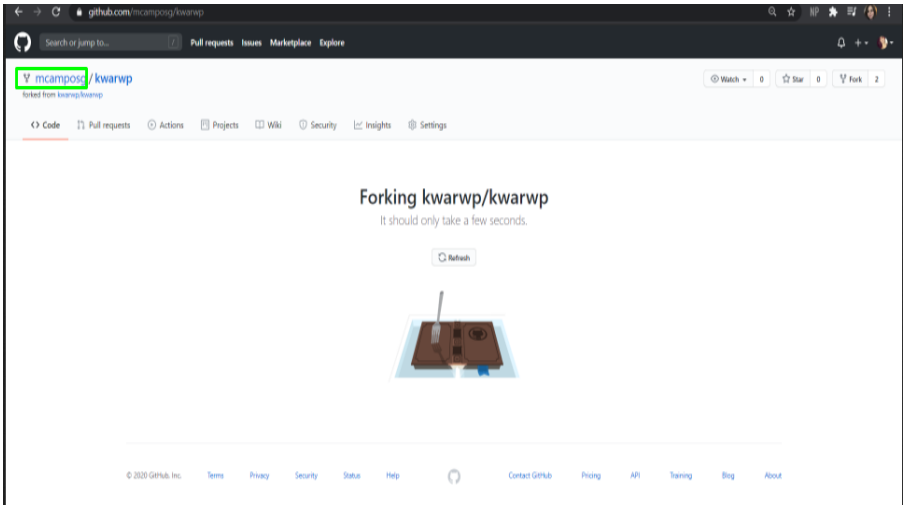
Step 2

- Escolha a conta para a qual deseja enviar a “cópia” do projeto (forkar)



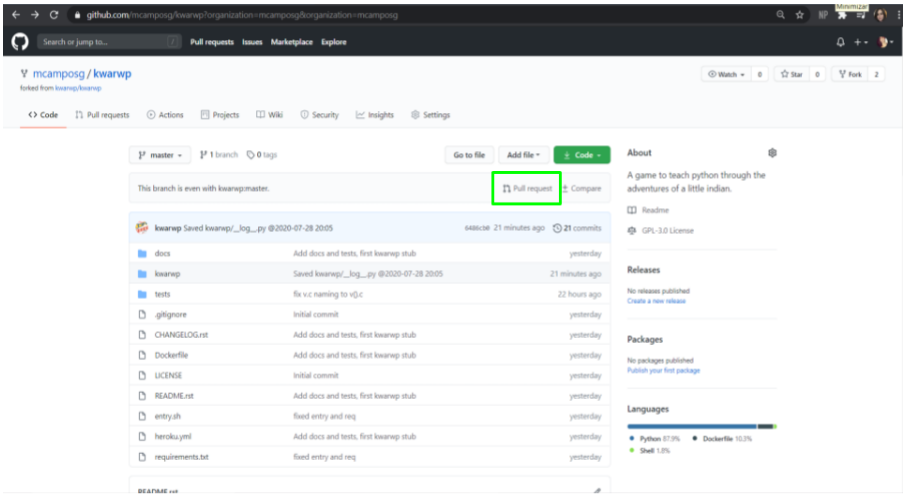
Step 3

- Depois de alguns segundos a cópia será enviada para a sua conta, você irá notar que o nome do seu usuário vai aparecer antes do nome do projeto (como mostra a imagem).



Step 4

- Agora você pode realizar mudanças sem alterar o projeto principal, para que suas mudanças sejam aplicadas ao projeto original você deve fazer um Pull request, assim o usuário original pode verificar se aceita ou não suas mudanças.



Referências

branch e pull-request

- Como Criar um Pull Request no GitHub | DigitalOcean
- Git e Github - Do clone ao pull request | Da2k Blog
- Como Criar um Pull Request no Github

tutoriais

- O que essas palavras significam no Git/GitHub: fork, clone, track? - Stack Overflow em Português
- GitHub: criando um Fork e enviando um Pull Request – Tiago Pariz
- git - Pull new updates from original GitHub repository into forked GitHub repository - Stack Overflow
- https://www.youtube.com/watch?v=IOEtQEm_MAY

Tela Inicial

Nesta tela vamos montar um simulacro do primeiro desafio kwarwp.

Note: Nesta versão estamos usando um componente especial do vitollino o **Jogo**. O Jogo é uma fábrica de componentes **Vitollino**, retorna um componente construído com os mesmos parâmetros de chamada de um compnente original. Veja alguns métodos usados no exemplo abaixo

---

Chamadas do Jogo

Jogo.c: Equivale a chamar uma Cena importada do Vitollino

Jogo.a: Equivale a chamar um Elemento importado do Vitollino

Código Fonte

Este tutorial ensina passo a passo a criação de um Ambiente de desenvolvimento na WEB.

```
class Kwarwp():
    """ Jogo para ensino de programação.

        :param vitollino: Empacota o engenho de jogo Vitollino.
    """
    OCA = "https://i.imgur.com/dZQ8liT.jpg"
    INDIO = "https://imgur.com/8jMuupz.png"
    SOLO = "https://i.imgur.com/sGoKfvs.jpg"
    TORA = "https://imgur.com/ldI7IbK.png"
    PICHE = "https://imgur.com/tLLVjfN.png"
    CEU = "https://i.imgur.com/UAETaiP.gif"
    SOL = "https://i.imgur.com/PfodQmT.gif"

    def __init__(self, vitollino=None, cenario="default"):
        self.v = vitollino()
        self.cena = self.cria(cenario=cenario) if vitollino else None

    def cria(self, cenario="default"):
        """ Cria o ambiente de programação Kwarwp."""
        cena = self.v.c(self.SOLO)
        indio = self.v.a(self.INDIO, w=100, h=100, x=300, y=400, cena=cena)
        oca = self.v.a(self.OCA, w=100, h=100, x=500, y=100, cena=cena)
        tora = self.v.a(self.TORA, w=100, h=100, x=100, y=400, cena=cena)
        piche = self.v.a(self.PICHE, w=100, h=100, x=100, y=100, cena=cena)
        piche = self.v.a(self.CEU, w=600, h=100, x=0, y=0, cena=cena)
        sol = self.v.a(self.SOL, w=60, h=60, x=0, y=40, cena=cena)
        cena.vai()
        return cena
```

Tela Gerada

Usando um Mapa

Nesta tela vamos montar um simulacro do primeiro desafio kwarwp. O desafio será montado a partir de um mapa com símbolos.

Note: Nesta versão estamos usando um componente especial do vitollino o **Jogo**. O Jogo é uma fábrica de componentes **Vitollino**, retorna um componente construído com os mesmos parâmetros de chamada de um compnente original. Veja alguns métodos usados no exemplo abaixo

---

Chamadas do Jogo

Jogo.c: Equivale a chamar uma Cena importada do Vitollino

Jogo.a: Equivale a chamar um Elemento importado do Vitollino

Código Fonte

Aqui especificamos um mapa que orienta a construção da arena. Cada símbolo representa um elemento, definido a seguir num dicionário de imagens dos elementos.

```
MAPA_INICIO = """
@....&
......
......
.#.^..
"""
```

Arena onde os desafios ocorrem.

Esta versão recebe como parâmetro um mapa que define a montagem da arena.

param vitollino Empacota o engenho de jogo Vitollino.

param mapa Um texto representando o mapa do desafio.

```
class Kwarwp():
  ...
```

Esta versão usa um dicionário para guardar as imagens dos elementos.

```
GLIFOS = {
"&": "https://i.imgur.com/dZQ8liT.jpg",  # OCA
"^": "https://imgur.com/8jMuupz.png",  # INDIO
".": "https://i.imgur.com/npb9Oej.png",  # VAZIO
"_": "https://i.imgur.com/sGoKfvs.jpg",  # SOLO
"#": "https://imgur.com/ldI7IbK.png",  # TORA
"@": "https://imgur.com/tLLVjfN.png",  # PICHE
"~": "https://i.imgur.com/UAETaiP.gif",  # CEU
"*": "https://i.imgur.com/PfodQmT.gif"  # SOL
}
```

A arena é construída a partir da matriz textual **mapa** dada. O atributo **lado** define a largura e altura .

```
def __init__(self, vitollino=None, mapa=MAPA_INICIO, medidas={}):
    self.v = vitollino()
    """Cria um matriz com os elementos descritos em cada linha de texto"""
    mapa = mapa.split()
    """Largura da casa da arena dos desafios, número de colunas no mapa"""
    self.lado, self.col = 100, len(mapa[0])
    self.cena = self.cria(mapa=mapa) if vitollino else None
```

A arena é construída a partir da matriz usando uma **list comprehension** .

```
def cria(self, mapa="  "):
```

Cria o ambiente de programação Kwarwp.

param mapa Um texto representando o mapa do desafio.

```
"""Cria um cenário com imagem de terra de chão batido, céu e sol"""
lado = self.lado
cena = self.v.c(self.GLIFOS["_"])
ceu = self.v.a(self.GLIFOS["~"], w=lado*self.col, h=lado, x=0, y=0, cena=cena)
sol = self.v.a(self.GLIFOS["*"], w=60, h=60, x=0, y=40, cena=cena)
```

A construção entre chaves **[]** é chamada **list comprehension**. Neste caso usamos intenamente duas iterações, uma para as linhas e outras para as colunas. Também estamos usando a função embutida **enumerate()**. Esta função pega uma lista e retorna outra lista, mas contendo tuplas onde o primeiro elemento é o índice do elemento original e o outro é o elemento original. . Posiciona os elementos segundo suas posições i, j na matriz mapa

```
[self.cria_elemento(imagem, x=i*lado, y=j*lado+lado, cena=cena)
    for j, linha in enumerate(mapa) for i, imagem in enumerate(linha)]
cena.vai()
return cena
```

Cria um elemento na arena do Kwarwp na posição definida.

param x coluna em que o elemento será posicionado.

param y linha em que o elemento será posicionado.

param cena cena em que o elemento será posicionado.

```
def cria_elemento(self,imagem, x, y, cena):
    lado = self.lado
    return self.v.a(self.GLIFOS[imagem], w=lado, h=lado, x=i*lado, y=j*lado+lado,␣
↪cena=cena)
```

Tela Gerada

Adicionando o Indio

Para melhorar o nosso jogo kwarwp, teremos que fazer o índio executar coisas guiadas pela programação. Precisamos agora ter uma classe Indio, para que ela possa executar os comandos. Neste exercício só iremos separar a construção do índio, usando uma classe para isso. No momento o índio não apresenta nenhum comportamento especial, foi somente uma refatoração. Este código é uma modificação do código descrito em *Usando um Mapa*

Note: Refatoração (do inglês Refactoring) é o processo de modificar um sistema de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.

O uso desta técnica aprimora a concepção (design) de um software e evita a deterioração tão comum durante o ciclo de vida de um código. Esta deterioração é geralmente causada por mudanças com objetivos de curto prazo ou por alterações realizadas sem a clara compreensão da concepção do sistema.

Outra consequência é a melhora no entendimento do código, o que facilita a manutenção e evita a inclusão de defeitos. Esta melhora no entendimento vem da constante alteração do código com objetivo de facilitar a comunicação de motivações, intenções e objetivos por parte do programador.

A refatoração é comumente feita quando se vai criar novas funcionalidades no código. O código é preparado para que as novidades sejam incorporadas da melhor maneira, sem perturbar ou incluir códigos confusos.

Wikipedia

Classe Indio

Com esta classe vamos separar o índio dos outros elementos da tela. Com isso poderemos colocar funcionalidades nela que os outros não tem.

Cria o personagem principal na arena do Kwarwp na posição definida.

param imagem A figura representando o índio na posição indicada.

param x Coluna em que o elemento será posicionado.

param y Linha em que o elemento será posicionado.

param cena Cena em que o elemento será posicionado.

```
class Indio():

   def __init__(self, imagem, x, y, cena):
      self.lado = lado = Kwarwp.LADO
      self.indio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
```

Classe Kwarwp

Vamos começar melhorando a classe Kwarwp, aplicando nela o conceito de fábrica. A fábrica constrói um componente segundo a especicificação dada. No nosso caso temos um símbolo que identifica o componenete no mapa. Este símbolo nos diz que tipo de objeto tem ali e também qual é a imagem que deve representar este objeto.

Jogo para ensino de programação.

param vitollino Empacota o engenho de jogo Vitollino.

param mapa Um texto representando o mapa do desafio.

param medidas Um dicionário usado para redimensionar a tela.

```
class Kwarwp():
   VITOLLINO = None
   """Referência estática para obter o engenho de jogo."""
   LADO = None
   """Referência estática para definir o lado do piso da casa."""

   def __init__(self, vitollino=None, mapa=MAPA_INICIO, medidas={}):
      Kwarwp.VITOLLINO = self.v = vitollino()
      """Cria um matriz com os elementos descritos em cada linha de texto"""
      self.mapa = mapa.split()
      """Largura da casa da arena dos desafios, número de colunas no mapa"""
      self.lado, self.col, self.lin = 100, len(self.mapa[0]), len(self.mapa)+1
      Kwarwp.LADO = self.lado
      w, h = self.col *self.lado, self.lin *self.lado
      self.taba = {}
      """Dicionário que a partir de coordenada (i,J) localiza um piso da taba"""
      medidas.update(width=w, height=f"{h}px")
      self.cena = self.cria(mapa=self.mapa) if vitollino else None
```

Método Cria

Este método define uma fábrica de componentes.

param mapa Um texto representando o mapa do desafio.

```
def cria(self, mapa=""):
```

Note: O Python suporta um tipo de contêiner como dicionários chamado "namedtuples ()" presente no módulo, "coleções". Como dicionários, eles contêm chaves com hash para um valor específico. Mas, pelo contrário, suporta o acesso a partir do valor-chave e da iteração, a funcionalidade que falta nos dicionários. Uma tupla nomeada assume o formato **nome_tupla = namedtuple("nome_tupla", "nome dos campos separados por branco")**

Operações em namedtuple (): Operações de acesso

1. Acesso por índice: os valores de atributo de namedtuple () são ordenados e podem ser acessados usando o número do índice, diferentemente dos dicionários que não são acessíveis pelo índice.
2. Acesso por nome da chave: O acesso por nome da chave também é permitido como nos dicionários.
3. usando getattr (): - Essa é outra maneira de acessar o valor, fornecendo o valor nomeado de parâmetro e chave como argumento.

GeeksForGeeks-Namedtuple

Esta tupla nomeada serve para definir o objeto construido e sua imagem.

nome Fab O nome da tupla que descreve a fábrica.

campo objeto O tipo de objeto que vai ser criado.

campo imagem A imagem que representa o objeto que vai ser criado.

```
from collections import namedtuple as nt
Fab = nt("Fab", "objeto imagem")
```

O atributo **fabrica** é um dicionário que relaciona o símbolo no mapa com a fábrica necessária para criar o componente.

Dicionário que define o tipo e a imagem do objeto para cada elemento.

```
fabrica = {
"&": Fab(self.coisa, f"{IMGUR}dZQ8liT.jpg"), # OCA
"^": Fab(self.indio, f"{IMGUR}8jMuupz.png"), # INDIO
".": Fab(self.vazio, f"{IMGUR}npb9Oej.png"), # VAZIO
"_": Fab(self.coisa, f"{IMGUR}sGoKfvs.jpg"), # SOLO
"#": Fab(self.coisa, f"{IMGUR}ldI7IbK.png"), # TORA
"@": Fab(self.coisa, f"{IMGUR}tLLVjfN.png"), # PICHE
"~": Fab(self.coisa, f"{IMGUR}UAETaiP.gif"), # CEU
"*": Fab(self.coisa, f"{IMGUR}PfodQmT.gif"), # SOL
"|": Fab(self.coisa, f"{IMGUR}uwYPNlz.png")  # CERCA
}
```

Cria um cenário com imagem de terra de chão batido, céu e sol. O mapa pode pode ser o definido no argumento ou atributo da instância do Kwarwp.

```
mapa = mapa if mapa != "" else self.mapa

mapa = self.mapa
lado = self.lado
cena = self.v.c(fabrica["_"].imagem)
ceu = self.v.a(fabrica["~"].imagem, w=lado*self.col, h=lado, x=0, y=0, cena=cena)
sol = self.v.a(fabrica["*"].imagem, w=60, h=60, x=0, y=40, cena=cena)
```

Cria um cenário com imagem de terra de chão batido, céu e sol. O mapa pode pode ser o definido no argumento ou atributo da instância do Kwarwp. Esta construção é uma compreensão de dicionário que posiciona os elementos segundo suas posições i, j na matriz mapa

---

Note: Como a compreensão de lista, o **Python** permite a compreensão de dicionário. Podemos criar dicionários usando expressões simples. Uma compreensão de dicionário assume o formato **{key: value for (key, value) em iterável}**

GeeksForGeeks_Dict_Comprenension

```
self.taba = {(i, j): fabrica[imagem].objeto(
      fabrica[imagem].imagem, x=i*lado, y=j*lado+lado, cena=cena)
      for j, linha in enumerate(mapa) for i, imagem in enumerate(linha)}

cena.vai()
return cena
```

Métodos Fabricantes - Coisa

Este método define uma fábrica para coisas que estão no cenário.

param imagem imagem que representa o elemento que será posicionado.

param x coluna em que o elemento será posicionado.

param y linha em que o elemento será posicionado.

param cena cena em que o elemento será posicionado.

Cria um elemento na arena do Kwarwp na posição definida.

```
def coisa(self, imagem, x, y, cena):
    lado = self.lado
    return self.v.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
```

Métodos Fabricantes - Indio

Este método define uma fábrica criando o índio o personagem principal.

param imagem imagem que representa o elemento que será posicionado.

param x coluna em que o elemento será posicionado.

param y linha em que o elemento será posicionado.

param cena cena em que o elemento será posicionado.

Cria o personagem principal na arena do Kwarwp na posição definida. Em vez de criar diretamente um elemento do Vitollino, cria uma classe para lidar com o componente e seu comportamento distinto.

```
def indio(self, imagem, x, y, cena):
    lado = self.lado
    return Indio(imagem, x=x, y=y, cena=cena)
```

Movendo o Indio

Para mover o índio, adicionamos o método **anda ()** na sua classe. No Kwarwp original, havia um método que executava o roteiros de comandos programado pelo usuário. Colocamos então um método **executa ()** que vai conter estes comandos. Também no jogo original bastava clicar no céu para executar os comandos. Programamos então um enlace do evento clique com método executa. No caso, o **ceu** pertence à classe Kwarwp então criamos um método **executa ()** na classe Kwarwp e chamamos o respectivo método do índio, **self.o_indio.executa ()**. No momento o índio só apresenta o comportamento de andar. Este código é uma modificação do código descrito em *Adicionando o Indio*

Classe Indio

Com esta classe vamos separar o índio dos outros elementos da tela. Com isso poderemos colocar funcionalidades nela que os outros não tem. No momento o índio tem o método *kwarwp.kwarapp.Indio.anda()* que movimenta o personagem na direção que está olhando. O método *kwarwp.kwarapp.Indio.executa()* contém o conjunto de comandos que dever ser executados para resolver o desafio.

Cria o personagem principal na arena do Kwarwp na posição definida.

param imagem A figura representando o índio na posição indicada.

param x Coluna em que o elemento será posicionado.

param y Linha em que o elemento será posicionado.

param cena Cena em que o elemento será posicionado.

```
class Indio():

    def __init__(self, imagem, x, y, cena):
        self.lado = lado = Kwarwp.LADO
        self.indio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
```

Método Anda

Este método define um comportamento que faz o personagem andar na direção para onde está olhando.

```
def anda(self):
    """ Faz o índio caminhar na direção em que está olhando.
    """
    self.posicao = (self.posicao[0], self.posicao[1]-1)
    """Assumimos que o índio está olhando para cima, decrementamos a posição **y**"""
    self.indio.y = self.posicao[1]*self.lado
    self.indio.x = self.posicao[0]*self.lado
```

Método Executa

Este método define um roteiro do comportamento que o personagem vai executar.

```
def executa(self):
    """ Roteiro do índio. Conjunto de comandos para ele executar.
    """
    self.anda()
```

Kwarwp - Enlace do Céu

A classe Kwarwp vai ter um enlace que liga o clique no céu com o chamado do roteiro de execuções do indio.

Jogo para ensino de programação.

param vitollino Empacota o engenho de jogo Vitollino.

param mapa Um texto representando o mapa do desafio.

param medidas Um dicionário usado para redimensionar a tela.

```
class Kwarwp():
    VITOLLINO = None
    ...
    self.o_indio = None
    """Instância do personagem principal, o índio, vai ser atribuído pela fábrica do␣
↪índio"""
    ...
```

Veja o código completo no tutorial *Método Cria*

Enlace no Método Cria

Este método define uma fábrica de componentes.

param mapa Um texto representando o mapa do desafio.

```
def cria(self, mapa=""):
...
ceu = self.v.a(fabrica["~"].imagem, w=lado*self.col, h=lado, x=0, y=0, cena=cena,␣
↪vai= self.executa)
```

(continues on next page)

(continued from previous page)

```
"""No argumento *vai*, associamos o clique no céu com o método **executa ()** desta␣
↪classe"""
...
```

Delegando a Execução

Este método recebe o evento .

param _ este argumento recebe a estrutura oriunda do evento, o _ indica que não será usado.

```
def executa(self, *_):
    """ Ordena a execução do roteiro do índio.
    """
    self.o_indio.executa()
```

Atribuindo o Indio na Fábrica

Este método define uma fábrica criando o índio o personagem principal.

param imagem imagem que representa o elemento que será posicionado.

param x coluna em que o elemento será posicionado.

param y linha em que o elemento será posicionado.

param cena cena em que o elemento será posicionado.

Cria o personagem principal na arena do Kwarwp na posição definida. Em vez de criar diretamente um elemento do Vitollino, cria uma classe para lidar com o componente e seu comportamento distinto. O atributo da instância **o_indio** passa a ser uma referência para uma instância da classe *kwarwp.kwarapp.Indio*

```
def indio(self, imagem, x, y, cena):
    self.o_indio = Indio(imagem, x=x, y=y, cena=cena)
    return self.o_indio
```

Organizando a Taba

O movimento do índio ainda está errático pois ele não sonda o ambiente, não respeita os limites da taba e atropela os objetos. Vamos povoar a taba com espaços vazios predeterminados. As coisas e o índio serão colocados nestes espaços e o índio só poderá ir para um vazio se ele estiver dentro dos limites da taba e estiver desocupado.

Para mover o índio, modificamos o método **anda ()** na sua classe. Na versão anterior, ele mudava a coordenada do elemento. Nesta ele muda o elemento para um **Vazio** adjacente.

See also:

Este código é uma modificação do código descrito em *Movendo o Indio*

Padrões de projeto

Padrões de projeto são a linguagem culta do programador. Eles são soluções típicas para problemas comuns em projeto de software. Cada padrão é como uma planta de construção que você pode customizar para resolver um problema de projeto particular em seu código.

See also:

Neste site Padrões de Projeto, veja também na Apresentação de Slides Padrões de Projeto

O protocolo duplo despacho

Neste protocolo é estabelecido um diálogo entre um objeto que quer entrar, o **Imigrante** e a vaga que quer ocupar, o **Destino**. O destino estando vago, pede para o imigrante ocupar imediatamente e recebe o pedido **ocupou** () do imigrante. Caso odestino esteja ocupado, ele delega a decisão ao **Ocupante** que decide ignorar ou acatar o pedido de acesso enviando o pedido de **ocupa** no últino caso. Neste exemplo, o protocolo está implementado em *Classe Indio - Duplo Despacho* e na *Classe Vazio*

See also:

Ver uma explicação externa em duplo despacho

O protocolo objeto de estado

Neste protocolo o objeto **Vazio** pode estar em um *Estado Ocupado* ou *Estado Vago*. Nesta implementação usaremos um jeito Pythônico para fazer uma modificação dinâmica de comportamento. Como métodos são objetos de primeira ordem no Python, em vez de criarmos classes para representar os estados, simplesmente trocamos a operação do método **acessa** () quando o estado é chaveado.

See also:

Ver uma explicação externa em estado de objeto

Classe Vazio

Com esta classe vamos separar locais onde coisas e o índio podem ser alocados. Vamos organizar a maneira como objetos se deslocam nesta taba usando *o protocolo duplo despacho*. Para entrar em um vazio o objeto pede **acessa** () e só entra se receber um convite **ocupa** (). Um outro protocolo que vamos usar é *o protocolo objeto de estado* Neste protocolo o objeto assume comportamentos diferentes caso esteja vago ou ocupado.

Cria um espaço vazio na taba, para alojar os elementos do desafio.

param imagem A figura representando o espaço vazio (normalmente transparente).

param x Coluna em que o elemento será posicionado.

param y Cinha em que o elemento será posicionado.

param cena Cena em que o elemento será posicionado.

```
class Vazio():
    """ Cria um espaço vazio na taba, para alojar os elementos do desafio.

        :param imagem: A figura representando o espaço vazio (normalmente␣
↪transparente).
        :param x: Coluna em que o elemento será posicionado.
        :param y: Cinha em que o elemento será posicionado.
        :param cena: Cena em que o elemento será posicionado.
    """

    def __init__(self, imagem, x, y, cena, ocupante=None):
        self.lado = lado = Kwarwp.LADO
```

(continues on next page)

(continued from previous page)

```
        self.posicao = (x//lado,y//lado-1)
        self.vazio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
        self._nada = Kwarwp.VITOLLINO.a()
        self.acessa = self._acessa
        self.ocupante = ocupante or self
        """O ocupante será definido pelo acessa, por default é o vazio"""
        self.acessa(ocupante)
```

Estado Ocupado

Consulta o ocupante atual se há permissão para substituí-lo pelo novo ocupante. Veja o *O protocolo objeto de estado*.

param ocupante O canditato a ocupar a posição corrente.

```
def _valida_acessa(self, ocupante):
    """ Consulta o ocupante atual se há permissão para substituí-lo pelo novo␣
↪ocupante.

        :param ocupante: O canditato a ocupar a posição corrente.
    """
    self.ocupante.acessa(ocupante)
```

Estado Vago

Atualmente a posição está vaga e pode ser acessada pelo novo ocupante.

A responsabilidade de ocupar definitivamente a vaga é do candidato a ocupante Caso ele esteja realmente apto a ocupar a vaga e deve cahamar de volta ao vazio com uma chamada ocupou.

param ocupante O canditato a ocupar a posição corrente.

See also:

Veja o *O protocolo objeto de estado*.

```
def _acessa(self, ocupante):
    """ Atualmente a posição está vaga e pode ser acessada pelo novo ocupante.

    A responsabilidade de ocupar definitivamente a vaga é do candidato a ocupante
    Caso ele esteja realmente apto a ocupar a vaga e deve cahamar de volta ao vazio
    com uma chamada ocupou.

        :param ocupante: O canditato a ocupar a posição corrente.
    """
    ocupante.ocupa(self)
```

Confirmando a Ocupação

O candidato à vaga decidiu ocupá-la e efetivamente entra neste espaço.

Este ocupante vai entrar no elemento do Vitollino e definitivamente se tornar o ocupante da vaga. Com isso ele troca o estado do método acessa para primeiro consultar a si mesmo, o ocupante corrente usando o protocolo definido em **_valida_acessa ()**

param ocupante O canditato a ocupar a posição corrente.

```
def ocupou(self, ocupante):
    """ O candidato à vaga decidiu ocupá-la e efetivamente entra neste espaço.

    :param ocupante: O canditato a ocupar a posição corrente.

    Este ocupante vai entrar no elemento do Vitollino e definitivamente se tornar
    o ocupante da vaga. Com isso ele troca o estado do método acessa para primeiro
    consultar a si mesmo, o ocupante corrente usando o protocolo definido em
    **_valida_acessa ()**

    """
    self.vazio.ocupa(ocupante)
    self.ocupante = ocupante
    self.acessa = self._valida_acessa
```

Pedido para Ocupar

Pedido por uma vaga para que ocupe a posição nela.

Neste caso, um objeto Vazio nunca vai ocupar nenhuma vaga. Este método está definido aqui para efeito de objeto nulo

param vaga A vaga a ser ocupada.

```
def ocupa(self, vaga):
    """ Pedido por uma vaga para que ocupe a posição nela.

    No caso do espaço vazio, não faz nada.
    """
    pass
```

Pedido para Sair

Pedido por um ocupante para que desocupe a posição nela.

Quando um ocupante deixa a vaga, ele envia este comando para desfazer a ocupação. Ver *O protocolo duplo despacho*

```
def sai(self):
    """ Pedido por um ocupante para que desocupe a posição nela.
    """
    self.ocupante = self
    self.acessa = self._acessa
```

Propriedade Elemento (elt).

A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento no outro . No caso do espaço vazio, vai retornar um elemento que não contém nada.

```
@property
def elt(self):
    """ A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento␣
↪no outro .
```

(continues on next page)

(continued from previous page)

```
    No caso do espaço vazio, vai retornar um elemento que não contém nada.
    """
    return self._nada.elt
```

Classe Indio - Duplo Despacho

Vamos modificar esta classe para ela suportar *O protocolo duplo despacho*

Cria o personagem principal na arena do Kwarwp na posição definida.

param imagem A figura representando o índio na posição indicada.

param x Coluna em que o elemento será posicionado.

param y Linha em que o elemento será posicionado.

param cena Cena em que o elemento será posicionado.

```
class Indio():

    def __init__(self, imagem, x, y, cena):
        self.lado = lado = Kwarwp.LADO
        self.indio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
        self.vaga = self
        self.posicao = (x//lado,y//lado)
        self.indio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
```

Método Anda - Acessa uma Vaga

Este método foi modificado para procurar na taba um vazio adjacente e realizar *O protocolo duplo despacho*.

```
def anda(self):
    """ Faz o índio caminhar na direção em que está olhando.
    """
    destino = (self.posicao[0], self.posicao[1]-1)
    """Assumimos que o índio está olhando para cima, decrementamos a posição **y**"""
    taba = self.taba.taba
    if destino in taba:
        vaga = taba[destino]
        """Recupera na taba a vaga para a qual o índio irá se transferir"""
        vaga.acessa(self)
        """Inicia o protocolo duplo despacho, pedindo para acessar a vaga"""

def executa(self):
    """ Roteiro do índio. Conjunto de comandos para ele executar.
    """
    self.anda()
```

Indio como Vaga Nula

O índio é usado como objeto nulo, representando uma vaga.

```
def sai(self):
    """ Rotina de saída falsa, o objeto Indio é usado como uma vaga nula.
    """
    pass
```

Indio no Despacho Duplo

O índio implementa *O protocolo duplo despacho*, no papel de ocupante de uma vaga. O índio tambe pode funcionar como um objeto intransponível, poi quando a vaga que ele ocupa delega a ele o pedido **acessa** (), ele não responde nada, negando acesso.

```
@property
def elt(self):
    """ A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento␣
↪no outro .

    No caso do índio, retorna o elt do elemento do atributo **self.indio**.
    """
    return self.indio.elt

def ocupa(self, vaga):
    """ Pedido por uma vaga para que ocupe a posição nela.

    :param vaga: A vaga que será ocupada pelo componente.

    No caso do índio, requisita que a vaga seja ocupada por ele.
    """
    self.vaga.sai()
    self.posicao = vaga.posicao
    vaga.ocupou(self)
    self.vaga = vaga

def acessa(self, ocupante):
    """ Pedido de acesso a essa posição, delegada ao ocupante pela vaga.

    :param ocupante: O componente candidato a ocupar a vaga já ocupada pelo índio.

    No caso do índio, ele age como um obstáculo e não prossegue com o protocolo.
    """
    pass
```

Kwarwp - Fabricando Vagas

A classe Kwarwp vai ser modificada para que na fábrica seja sempre criado um **Vazio**. Neste vazio, o objeto a ser posicionado é alocado nesta vaga do local vazio.

Jogo para ensino de programação.

param vitollino Empacota o engenho de jogo Vitollino.

param mapa Um texto representando o mapa do desafio.

param medidas Um dicionário usado para redimensionar a tela.

```
class Kwarwp():
    VITOLLINO = None
    ...
    self.o_indio = None
    """Instância do personagem principal, o índio, vai ser atribuído pela fábrica do
↪índio"""
    ...
```

See also:

Veja o código anterior da classe no tutorial *Movendo o Indio*

Vagas nas Fábricas de Componentes

Estes método definen fábricas de componentes.

param x coluna em que o elemento será posicionado.

param y linha em que o elemento será posicionado.

param cena cena em que o elemento será posicionado.

```
def coisa(self, imagem, x, y, cena):
    """ Cria um elemento na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.

    Cria uma vaga vazia e coloca o componente dentro dela.
    """
    coisa = Indio(imagem, x=0, y=0, cena=cena, taba=self)
    """o índio tem deslocamento zero, pois é relativo à vaga"""
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=coisa)
    """Aqui o índio está sendo usado para qualquer objeto, enquanto não tem o próprio"
↪""
    return vaga

def vazio(self, imagem, x, y, cena):
    """ Cria um espaço vazio na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.
    """
    vaga = Vazio(imagem, x=x, y=y, cena=cena, ocupante=self)
    """ O Kwarwp é aqui usado como um ocupante nulo, que não ocupa uma vaga vazia."""
    return vaga

def indio(self, imagem, x, y, cena):
    """ Cria o personagem principal na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.
    """
    # self.o_indio = Indio(imagem, x=x, y=y, cena=cena)
```

(continues on next page)

(continued from previous page)

```
    self.o_indio = Indio(imagem, x=0, y=0, cena=cena, taba=self)
    """o índio tem deslocamento zero, pois é relativo à vaga"""
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=self.o_indio)
    return vaga
```

Ocupante nulo

O Kwarwp é aqui usado como um ocupante objeto nulo, usado ao fabricar espaços vazios O pedido de ocupar é ignorado.

```
def ocupa(self, *_):
    """ O Kwarwp é aqui usado como um ocupante falso, o pedido de ocupar é ignorado.
    """
    pass
```

Melhorando o Índio

O movimento do índio pode melhorar, podendo dobrar à esquerda e direita.

Para direcionar o índio, modificamos o método **anda ()** para considerar a direção. Adicionamos *Os Métodos Direcionais e a Fala*

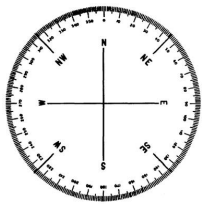
Neste tutorial também incuímos duas novas classes e seus comportamentos, a **Oca** e o **Piche**. Ele tem comportamentos que se assemelham entre si e também se parecem com o **Vazio**. Para tirar proveito disso usamos o mecanismo de heranca da linguagem Python.

See also:

Este código é uma modificação do código descrito em *Organizando a Taba*

Pontos cardeais

Estas tuplas nomeadas formam um vetor de pares ordenados que corespondem às componentes em x e y dos vetores unitários dos pontos cardeais na Rosa dos Ventos.



```
from collections import namedtuple as nt

Ponto = nt("Ponto", "x y")
"""Par de coordenadas na direção horizontal (x) e vertiacal (y)."""
Rosa = nt("Rosa", "n l s o")
"""Rosa dos ventos com as direções norte, leste, sul e oeste."""
```

Classe Indio - Com Direção

Cria o personagem principal na arena do Kwarwp na posição definida.

param imagem A figura representando o índio na posição indicada.

param x Coluna em que o elemento será posicionado.

param y Linha em que o elemento será posicionado.

param cena Cena em que o elemento será posicionado.

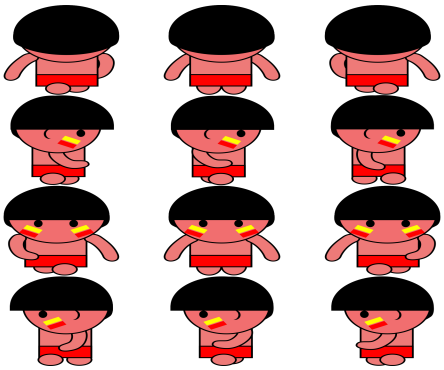
param taba Representa a taba onde o índio faz o desafio.

```
AZIMUTE = Rosa(Ponto(0, -1),Ponto(1, 0),Ponto(0, 1),Ponto(-1, 0),)
"""Constante com os pares ordenados que representam os vetores unitários dos pontos␣
↪cardeais."""

def __init__(self, imagem, x, y, cena, taba):
    self.lado = lado = Kwarwp.LADO
    self.azimute = self.AZIMUTE.n
    """índio olhando para o norte"""
    self.taba = taba
    self.vaga = self
    self.posicao = (x//lado,y//lado)
    self.indio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
    self.x = x
    """Este x provisoriamente distingue o índio de outras coisas construídas com esta␣
↪classe"""
    if x:
        self.indio.siz = (lado*3, lado*4)
        """Define as proporções da folha de sprites"""
        self.mostra()
```

Os Métodos Direcionais e a Fala

O método **mostra** modifica a exibição da *folha de sprites*, posicionando o canto superior esquerdo (origem) em uma coordenada negativa em relação à janela de exibição. Desta forma, a posição correta do índio vai aparecer na tela.



Note: O termo Sprite vem do latim spiritus que significa "Espíritos", mas também pode significar "fada" ou "duende". No âmbito da computação gráfica (Quer seja em games ou não) são os quadros de movimento que são desenhados individualmente com uma pequena variação entre si, mas obedecendo um padrão sequencial que quando disposto

numa ordem coerente acaba gerando uma animação de movimento quando exibidas em sucessão. Ver externamente Definição de Sprite Sheet

Os métodos **esquerda ()** e **direita ()** trocam a direção (azimute) para o qual o índio está olhando. Eles usam uma propriedade das listas em Python de circularidade dos índices negativos. Quando se tenta acessar uma posição de uma lista com um índice negativo, a posição é contada do fim da lista para o princípio.

```
def mostra(self):
    """ Modifica a figura (Sprite) do índio mostrando para onde está indo.
    """
    sprite_col = sum(self.posicao) % 3
    """Faz com que três casas adjacentes tenha valores diferentes para a coluna do␣
↪sprite"""
    sprite_lin = self.AZIMUTE.index(self.azimute)
    """A linha do sprite depende da direção dque índio está olhando"""
    self.indio.pos = (-self.lado*sprite_col, -self.lado*sprite_lin)

def esquerda(self):
    """ Faz o índio mudar da direção em que está olhando para a esquerda.
    """
    self.azimute = self.AZIMUTE[self.AZIMUTE.index(self.azimute)-1]
    self.mostra()

def direita(self):
    """ Faz o índio mudar da direção em que está olhando para a direita.
    """
    self.azimute = self.AZIMUTE[self.AZIMUTE.index(self.azimute)-3]
    self.mostra()

def fala(self, texto=""):
    """ O índio fala um texto dado.

    :param texto: O texto a ser falado.
    """
    self.taba.fala(texto)
```

Os Protocolos de Saída

O ìndio teve que ser modificado para incorporar um novo duplo despacho de saída. Ele terá que consultar primeiro a vaga onde está para saber se pode sair

Ao receber de um evento o comando **anda ()**, ele terá que consultar com um **sair ()** a vaga onde está. Em uma vaga normal ele recebe o **siga ()**, segue em frente e executa o seu **_anda ()** original. Se ele entrou numa vaga que tinha uma armadilha, agora a vaga onde está é a armadilha. Em uma armadilha leniente, segue normalmente. Numa armadilha rígida, o seu pedido de **sair ()** é ignorado e ele não recebe a resposta **siga ()**.

```
def anda(self):
    """Objeto tenta sair, tem que consultar a vaga onde está"""
    self.vaga.sair()

def sair(self):
    """Objeto de posse do índio tenta sair e é autorizado"""
    self.vaga.ocupante.siga()

def siga(self):
```

(continues on next page)

(continued from previous page)

```
    """Objeto tentou sair e foi autorizado"""
    self._anda()

def _anda(self):
    """ Faz o índio caminhar na direção em que está olhando.
    """
    destino = (self.posicao[0]+self.azimute.x, self.posicao[1]+self.azimute.y)
    """A posição para onde o índio vai depende do vetor de azimute corrente"""
    taba = self.taba.taba
    if destino in taba:
        vaga = taba[destino]
        """Recupera na taba a vaga para a qual o índio irá se transferir"""
        vaga.acessa(self)
```

Kwarwp - Oca e Piche

A classe Kwarwp vai ser modificada para agregar novas fábricas. Além do *Classe Indio - Com Direção* e do *Vazio - A Vaga* teremos a *Oca - O Destino* e o *Piche - A Armadilha*

Jogo para ensino de programação.

param vitollino Empacota o engenho de jogo Vitollino.

param mapa Um texto representando o mapa do desafio.

param medidas Um dicionário usado para redimensionar a tela.

```
class Kwarwp():
   VITOLLINO = None
   ...
   self.o_indio = None
   """Instância do personagem principal, o índio, vai ser atribuído pela fábrica do␣
↪índio"""
   ...
```

See also:

Veja o código anterior da classe no tutorial *Organizando a Taba*

Dicionário com Oca e Piche

O método **cria ()** define as fábricas de componentes.

No dicionário pode se ver que **"&"** agora remete a **maloc** e **"@"** remete a **barra**. Uma outra alteração é que a construção do **sol** agora se liga ao tratador de evento **esquerda**. Isto permite que se experimente andar com o índio no cenário. Note que agora o **ceu** foi convertido em atributo de instância. Por isso agora ele é referido como **self.ceu**. O céu será referenciado por outros objetos que precisam escrever textos, e o céu é onde será escrito.

param mapa Um texto representando o mapa do desafio.

```
def cria(self, mapa=""):
    """ Fábrica de componentes.

    :param mapa: Um texto representando o mapa do desafio.
    """
```

(continues on next page)

(continued from previous page)

```
    Fab = nt("Fab", "objeto imagem")
    """Esta tupla nomeada serve para definir o objeto construido e sua imagem."""

    fabrica = {
    "&": Fab(self.maloc, f"{IMGUR}dZQ8liT.jpg"), # OCA
    "^": Fab(self.indio, f"{IMGUR}UCWGCKR.png"), # INDIO
    ".": Fab(self.vazio, f"{IMGUR}npb9Oej.png"), # VAZIO
    "_": Fab(self.coisa, f"{IMGUR}sGoKfvs.jpg"), # SOLO
    "#": Fab(self.coisa, f"{IMGUR}ldI7IbK.png"), # TORA
    "@": Fab(self.barra, f"{IMGUR}tLLVjfN.png"), # PICHE
    "~": Fab(self.coisa, f"{IMGUR}UAETaiP.gif"), # CEU
    "*": Fab(self.coisa, f"{IMGUR}PfodQmT.gif"), # SOL
    "|": Fab(self.coisa, f"{IMGUR}uwYPNlz.png")  # CERCA
    }
    """Dicionário que define o tipo e a imagem do objeto para cada elemento."""
    mapa = mapa if mapa != "" else self.mapa
    """Cria um cenário com imagem de terra de chão batido, céu e sol"""
    mapa = self.mapa
    lado = self.lado
    cena = self.v.c(fabrica["_"].imagem)
    self.ceu = self.v.a(fabrica["~"].imagem, w=lado*self.col, h=lado-10, x=0, y=0,␣
↪cena=cena, vai=self.executa,
                        style={"padding-top": "10px", "text-align": "center"})
    """No argumento *vai*, associamos o clique no céu com o método **executa ()**␣
↪desta classe.
          O *ceu* agora é um argumento de instância e por isso é referenciado como␣
↪**self.ceu**.
    """
    sol = self.v.a(fabrica["*"].imagem, w=60, h=60, x=0, y=40, cena=cena, vai=self.
↪esquerda)
    """No argumento *vai*, associamos o clique no sol com o método **esquerda ()**␣
↪desta classe."""
    self.taba = {(i, j): fabrica[imagem].objeto(fabrica[imagem].imagem, x=i*lado,␣
↪y=j*lado+lado, cena=cena)
           for j, linha in enumerate(mapa) for i, imagem in enumerate(linha)}
    """Posiciona os elementos segundo suas posições i, j na matriz mapa"""
    cena.vai()
    return cena
```

Comandos para o Índio

O método **fala ()** é usado por objetos que emitem mensagens. Ele instrumentaliza o céu para que um texto em html seja escrito nele.

O método **esquerda ()** invoca sua contrapartida na instância de **Indio**. O método **executa ()** invoca sua contrapartida na instância de **Indio**.

```
def fala(self, texto=""):
    """ O Kwarwp é aqui usado para falar algo que ficará escrito no céu.
    """
    self.ceu.elt.html = texto
    pass

def esquerda(self, *_):
    """ Ordena a execução do roteiro do índio.
```

(continues on next page)

(continued from previous page)

```
    """
    self.o_indio.esquerda()

def executa(self, *_):
    """ Ordena a execução do roteiro do índio.
    """
    self.o_indio.executa()
```

Fabricando a Oca e o Piche

O método **maloc ()** invoca a criação da *Oca - O Destino*. O método **barra ()** invoca a criação do *Piche - A Armadilha*.

```
def maloc(self, imagem, x, y, cena):
    """ Cria uma maloca na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.

    Cria uma vaga vazia e coloca o componente dentro dela.
    """
    coisa = Oca(imagem, x=0, y=0, cena=cena, taba=self)
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=coisa)
    return vaga

def barra(self, imagem, x, y, cena):
    """ Cria uma armadilha na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.

    Cria uma vaga vazia e coloca o componente dentro dela.
    """
    coisa = Piche(imagem, x=0, y=0, cena=cena, taba=self)
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=coisa)
    return vaga
```

Ocupante e Vaga Nulos

O Kwarwp é aqui usado como um ocupante objeto nulo, usado ao fabricar espaços vazios O pedido de ocupar é ignorado.

```
def sai(self, *_):
    """ O Kwarwp é aqui usado como uma vaga falsa, o pedido de sair é ignorado.
    """
    pass

def ocupa(self, *_):
    """ O Kwarwp é aqui usado como um ocupante falso, o pedido de ocupar é ignorado.
    """
    pass
```

Vazio - A Vaga

O Vazio vai ser atualizado aqui para funcionar como uma vaga leniente, ou seja, deixa sair quem quiser abandonar a vaga.

A principal ideia aqui vai ser usar o **Vazio** como *classe base* de uma linhagem de herança, onde outras classes vão herdar o seu comportamento. No diagrama abaixo vemos que **Piche** herda de **Vazio** e por sua vez **Oca** herda de **Piche**

Note: O principal mecanismo do recurso da herança é permitir que uma classe possa ser derivada de uma classe base, permitindo que um comportamento mais especifico seja implementado na subclasse. A herança, é também uma importante característica para ao reuso de algoritmos e evitar códigos redundantes que possam tornar difícil a manutenção da base de códigos. Ver externamente O Uso da Herança

---

```
class Vazio():
    """ Cria um espaço vazio na taba, para alojar os elementos do desafio.

        :param imagem: A figura representando o índio na posição indicada.
        :param x: Coluna em que o elemento será posicionado.
        :param y: Cinha em que o elemento será posicionado.
        :param cena: Cena em que o elemento será posicionado.
    """

    def __init__(self, imagem, x, y, cena, ocupante=None):
        self.lado = lado = Kwarwp.LADO
        self.posicao = (x//lado,y//lado-1)
        self.vazio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
        self._nada = Kwarwp.VITOLLINO.a()
        self.acessa = self._acessa
        """O **acessa ()** é usado como método dinâmico, variando com o estado da␣
↪vaga.
        Inicialmente tem o comportamento de **_acessa ()** que é o estado vago,␣
↪aceitando ocupantes"""
        self.ocupante = ocupante or self
        """O ocupante se não for fornecido é encenado pelo próprio vazio, agindo como␣
↪nulo"""
        self.acessa(ocupante)
        self.sair = self._sair
        """O **sair ()** é usado como método dinâmico, variando com o estado da vaga.
        Inicialmente tem o comportamento de **_sair ()** que é o estado leniente,␣
↪aceitando saidas"""
```

O Objeto de Estado Sair

O *Vazio - A Vaga* tem um outro estado de objeto além do **acessa ()**. Este objeto é o **sair ()**, que assume os estados **_sair** quando a vaga está livre ou **_pede_sair ()** quando está ocupada.

```
def _sair(self):
    """Objeto tenta sair e secebe autorização para seguir"""
    self.ocupante.siga()

def _pede_sair(self):
    """Objeto tenta sair e consulta o ocupante para seguir"""
    self.ocupante.sair()
```

Piche - A Armadilha

O piche vai funcionar como uma forma especializada do *Vazio - A Vaga*

```
class Piche(Vazio):
    """ Poça de Piche que gruda o índio se ele cair nela.

        :param imagem: A figura representando o índio na posição indicada.
        :param x: Coluna em que o elemento será posicionado.
        :param y: Cinha em que o elemento será posicionado.
        :param cena: Cena em que o elemento será posicionado.
        :param taba: Representa a taba onde o índio faz o desafio.
    """

def __init__(self, imagem, x, y, cena, taba):
    self.taba = taba
    self.vaga = taba
    self.lado = lado = Kwarwp.LADO
    self.posicao = (x//lado,y//lado-1)
    self.vazio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=0, y=0, cena=cena)
    self._nada = Kwarwp.VITOLLINO.a()
    self.acessa = self._acessa
    """O **acessa ()** é usado como método dinâmico, variando com o estado da vaga.
    Inicialmente tem o comportamento de **_acessa ()** que é o estado vago, aceitando␣
↪ocupantes"""
    self.sair = self._sair
    """O **sair ()** é usado como método dinâmico, variando com o estado da vaga.
    Inicialmente tem o comportamento de **_sair ()** que é o estado vago, aceitando␣
↪ocupantes"""

def ocupa(self, vaga):
    """ Pedido por uma vaga para que ocupe a posição nela.

    :param vaga: A vaga que será ocupada pelo componente.

    No caso do piche, requisita que a vaga seja ocupada por ele.
    """
    self.vaga.sai()
    self.posicao = vaga.posicao
    vaga.ocupou(self)
    self.vaga = vaga

def _pede_sair(self):
    """Objeto tenta sair mas não é autorizado"""
    self.taba.fala("Você ficou preso no piche")
```

Oca - O Destino

A Oca vai funcionar como uma forma especializada do *Piche - A Armadilha*

```
class Oca(Piche):
    """ A Oca é o destino final do índio, não poderá sair se ele entrar nela.

        :param imagem: A figura representando o índio na posição indicada.
        :param x: Coluna em que o elemento será posicionado.
        :param y: Cinha em que o elemento será posicionado.
```

(continues on next page)

(continued from previous page)

```
        :param cena: Cena em que o elemento será posicionado.
        :param taba: Representa a taba onde o índio faz o desafio.
    """

    def _pede_sair(self):
        """Objeto tenta sair mas não é autorizado"""
        self.taba.fala("Você chegou no seu objetivo")

    def _acessa(self, ocupante):
        """ Atualmente a posição está vaga e pode ser acessada pelo novo ocupante.

        A responsabilidade de ocupar definitivamente a vaga é do candidato a ocupante
        Caso ele esteja realmente apto a ocupar a vaga e deve cahamar de volta ao␣
↪vazio
        com uma chamada ocupou.

            :param ocupante: O canditato a ocupar a posição corrente.
        """
        self.taba.fala("Você chegou no seu objetivo")
        ocupante.ocupa(self)
```

Incluindo a Tora

Para incluir a Tora, decidimos repartir o módulo com um outro, o **kwarwp.kwarwpart**

Neste tutorial incuímos uma nova classe e seus comportamentos, a **Tora**. Ela tem comportamento que se assemelham ao **Vazio** e usaremos a herança.

See also:

Este código é uma modificação do código descrito em *Melhorando o Índio*. O código da classe Tora pode ser visto em *A Tora e Outras Partes*.

Importando o Módulo Kwarwpart

As várias partes do jogo foram transferidas para outro módulo. Para podermos usar neste módulo teremos que importar.

```
from collections import namedtuple as nt
from kwarwp.kwarwpart import Vazio, Piche, Oca, Tora, NULO

IMGUR = "https://imgur.com/"
"""Prefixo do site imgur."""
MAPA_INICIO = """
@....&
......
......
.#.^..
"""
```

Classe Indio - Pega e Larga

Cria o personagem principal na arena do Kwarwp na posição definida.

param imagem A figura representando o índio na posição indicada.

param x Coluna em que o elemento será posicionado.

param y Linha em que o elemento será posicionado.

param cena Cena em que o elemento será posicionado.

param taba Representa a taba onde o índio faz o desafio.

```
AZIMUTE = Rosa(Ponto(0, -1),Ponto(1, 0),Ponto(0, 1),Ponto(-1, 0),)
"""Constante com os pares ordenados que representam os vetores unitários dos pontos␣
↪cardeais."""

def __init__(self, imagem, x, y, cena, taba):
    self.lado = lado = Kwarwp.LADO
    self.azimute = self.AZIMUTE.n
    """índio olhando para o norte"""
    self.taba = taba
    self.vaga = self
    self.posicao = (x//lado,y//lado)
    self.indio = Kwarwp.VITOLLINO.a(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
    self.x = x
    """Este x provisoriamente distingue o índio de outras coisas construídas com esta␣
↪classe"""
    if x:
        self.indio.siz = (lado*3, lado*4)
        """Define as proporções da folha de sprites"""
        self.mostra()
```

Interação do Índio com a Tora

O método **pega** () é usado pelo índio para adquirir e carregar a tora. O método **pega** () invoca sua contrapartida **pegar** () na instância de **Tora**. A tora responde à requisição invocando **ocupar** () no **Indio**.

O método **larga** () invoca **acessa** () na instância de **Vaga** onde a Tora deve ser colocada, mas passa como parâmetro o ocupante em vez de si próprio. De resto é executado o double dispatch como no anda, sendo que "andarilho" é a **Tora**.

See also:

Este código é uma modificação do código descrito em *Melhorando o Índio*. O código da classe **Tora** e o diagrama do protocolo podem ser visto em *Tora - O Tronco Carregável*.

```
def pega(self):
    """tenta pegar o objeto que está diante dele"""
    destino = (self.posicao[0]+self.azimute.x, self.posicao[1]+self.azimute.y)
    """A posição para onde o índio vai depende do vetor de azimute corrente"""
    taba = self.taba.taba
    if destino in taba:
        vaga = taba[destino]
        """Recupera na taba a vaga para a qual o índio irá se transferir"""
        vaga.pegar(self)

def larga(self):
    """tenta largar o objeto que está segurando"""
    destino = (self.posicao[0]+self.azimute.x, self.posicao[1]+self.azimute.y)
    """A posição para onde o índio vai depende do vetor de azimute corrente"""
    taba = self.taba.taba
    if destino in taba:
        vaga = taba[destino]
```

(continues on next page)

(continued from previous page)

```
        """Recupera na taba a vaga para a qual o índio irá se transferir"""
        # self.ocupante.largar(vaga)
        vaga.acessa(self.ocupante)

def ocupou(self, ocupante):
    """ O candidato à vaga decidiu ocupá-la e efetivamente entra neste espaço.

    :param ocupante: O canditato a ocupar a posição corrente.

    Este ocupante vai entrar no elemento do Vitollino e definitivamente se tornar
    o ocupante da vaga. Com isso ele troca o estado do método acessa para primeiro
    consultar a si mesmo, o ocupante corrente usando o protocolo definido em
    **_valida_acessa ()**

    """
    self.indio.ocupa(ocupante)
    self.ocupante = ocupante
```

Kwarwp - Tora

A classe Kwarwp vai ser modificada para agregar novas fábricas. Teremos a construção de instância de *Tora - O Tronco Carregável* como uma nova fábrica definida em *Fabricando a Tora*.

Jogo para ensino de programação.

param vitollino Empacota o engenho de jogo Vitollino.

param mapa Um texto representando o mapa do desafio.

param medidas Um dicionário usado para redimensionar a tela.

```
class Kwarwp():
   VITOLLINO = None
   ...
   self.o_indio = None
   """Instância do personagem principal, o índio, vai ser atribuído pela fábrica do␣
↪índio"""
   ...
```

See also:

Veja o código anterior da classe no tutorial *Melhorando o Índio*

Dicionário com Tora

O método **cria ()** define as fábricas de componentes.

No dicionário pode se ver que **"#"** agora remete ao método fábrica **atora ()** que é explicado em *Fabricando a Tora*.

param mapa Um texto representando o mapa do desafio.

```
def cria(self, mapa=""):
    """ Fábrica de componentes.

    :param mapa: Um texto representando o mapa do desafio.
    """
```

(continues on next page)

(continued from previous page)

```
    Fab = nt("Fab", "objeto imagem")
    """Esta tupla nomeada serve para definir o objeto construido e sua imagem."""

    fabrica = {
    "&": Fab(self.maloc, f"{IMGUR}dZQ8liT.jpg"), # OCA
    "^": Fab(self.indio, f"{IMGUR}UCWGCKR.png"), # INDIO
    ".": Fab(self.vazio, f"{IMGUR}npb9Oej.png"), # VAZIO
    "_": Fab(self.coisa, f"{IMGUR}sGoKfvs.jpg"), # SOLO
    "#": Fab(self.atora, f"{IMGUR}0jSB27g.png"), # TORA
    "@": Fab(self.barra, f"{IMGUR}tLLVjfN.png"), # PICHE
    "~": Fab(self.coisa, f"{IMGUR}UAETaiP.gif"), # CEU
    "*": Fab(self.coisa, f"{IMGUR}PfodQmT.gif"), # SOL
    "|": Fab(self.coisa, f"{IMGUR}uwYPNlz.png")  # CERCA
    }
    ... # ver tutorial anterior
```

Fabricando a Tora

O método **atora ()** invoca a criação da *Tora - O Tronco Carregável*. No **Vitollino**, um clique no elemento invoca o seu método **vai ()**. Neste método mostramos a associação do clique da tora com o **larga ()** do **Indio**, associando uma função **lambda** com o vai do índio. O **lambda** é um método anônimo para encapsular a chamada do larga sem que ele seja invocado imediatamente.

A mesma manobra foi feita com o índio associando o clique nele com o seu próprio método **pega ()**

```
def atora(self, imagem, x, y, cena):
    """ Cria uma tora na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.

    Cria uma vaga vazia e coloca o componente dentro dela.
    """
    coisa = Tora(imagem, x=0, y=0, cena=cena, taba=self)
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=coisa)
    coisa.vazio.vai = lambda *_: self.o_indio.larga()
    """o vazio.vai é associado ao método larga do índio"""
    return vaga

def indio(self, imagem, x, y, cena):
    """ Cria o personagem principal na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.
    """
    self.o_indio = Indio(imagem, x=1, y=0, cena=cena, taba=self)
    """ O índio tem deslocamento zero, pois é relativo à vaga.
        O **x=1** serve para distinguir o indio de outros derivados.
    """
    self.o_indio.indio.vai = lambda *_: self.o_indio.pega()
    """o índio.vai é associado ao seu próprio metodo pega"""
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=self.o_indio)
    return vaga
```

A Tora e Outras Partes

Esta página descreve como foi feito o módulo **kwarwp.kwarwpart**. Para incluir a **Tora**, decidimos mover várias classes, tirando do módulo **kwarwp.kwarapp** e trazendo para este novo módulo. Procure construir este módulo com as dicas do tutorial. Caso tenha muita dificuldade em fazer funcionar, consulte o código aqui:

- *kwarwp.kwarwpart.Vazio* Espaço vago na arena do desafio.
- *kwarwp.kwarwpart.Oca* Destino final da aventura.
- *kwarwp.kwarwpart.Piche* Uma armadilha para prender o índio.
- *kwarwp.kwarwpart.Tora* Uma tora que o índio pode pegar.
- *kwarwp.kwarwpart.Nulo* Objeto nulo passivo a todas as requisições.

Nesta parte do tutorial mostramos uma nova classe e seus comportamentos, a **Tora**. Ela tem comportamento que se assemelham ao **Vazio** e mais especificamente ao do seu descendente **Piche** e para isso usaremos a herança.

See also:

Este código é uma modificação do código descrito em *Melhorando o Índio*. O código da classe **Indio** adaptada para interagir com a **Tora** pode ser visto em *Incluindo a Tora*.

Tora - O Tronco Carregável

O objeto Tora foi criado neste novo módulo herdando do objeto **Piche**. A definição do **Piche** pode ser visto em *Melhorando o Índio*.

Protocolos de Interação com a Tora

A Tora tem que ser implementada incorporando um novo duplo despacho de saída. Ela terá que interagir com o índio para poder ser carregada por ele.

O método **pega** () é usado pelo índio para adquirir e carregar a tora. O método **pega** () invoca sua contrapartida **pegar** () na instância de **Tora**. A tora responde à requisição invocando **ocupar** () no **Indio**.

O método **larga** () invoca **acessa** () na instância de **Vaga** onde a Tora deve ser colocada, mas passa como parâmetro o ocupante em vez de si próprio. De resto é executado o double dispatch como no anda, sendo que "andarilho" é a **Tora**.

Note: O principal mecanismo do recurso da herança é permitir que uma classe possa ser derivada de uma classe base, permitindo que um comportamento mais especifico seja implementado na subclasse. A herança, é também uma importante característica para ao reuso de algoritmos e evitar códigos redundantes que possam tornar difícil a manutenção da base de códigos. Ver externamente O Uso da Herança

---

```
class Tora(Piche):
    """  A Tora é um pedaço de tronco cortado que o índio pode carregar ou empurrar.

        :param imagem: A figura representando o índio na posição indicada.
        :param x: Coluna em que o elemento será posicionado.
        :param y: Linha em que o elemento será posicionado.
        :param cena: Cena em que o elemento será posicionado.
        :param taba: Representa a taba onde o índio faz o desafio.
    """

    def pegar(self, requisitante):
```

(continues on next page)

(continued from previous page)

```
        """ Consulta o ocupante atual se há permissão para pegar e entregar ao␣
↪requistante.

            :param requistante: O ator querendo pegar o objeto.
        """
        vaga = requisitante
        self.vaga.sai()
        # self.posicao = vaga.posicao
        vaga.ocupou(self)
        self.vaga = vaga

    @property
    def posicao(self):
        """ A propriedade posição faz parte do protocolo do double dispatch com o␣
↪Indio .

        No caso da tora, retorna o a posição do atributo **self.vaga**.
        """
        return self.vaga.posicao

    @posicao.setter
    def posicao(self, _):
        """ A propriedade posição faz parte do protocolo do double dispatch com o␣
↪Indio .

        No caso da tora, é uma propriedade de somente leitura, não executa nada.
        """
        pass

    @property
    def elt(self):
        """ A propriedade elt faz parte do protocolo do Vitollino para anexar um␣
↪elemento no outro .

        No caso da tora, retorna o elt do elemento do atributo **self.vazio**.
        """
        return self.vazio.elt

    def _acessa(self, ocupante):
        """ Pedido de acesso a essa posição, delegada ao ocupante pela vaga.

        :param ocupante: O componente candidato a ocupar a vaga já ocupada pelo índio.

        No caso da tora, ela age como um obstáculo e não prossegue com o protocolo.
        """
        pass
```

Os Objetos Piche e Oca

Este objetos foram copiados sem alteração do módulo original para cá. A única alteração foi no **Piche** que teve que importar localmente o **Kwarwp** que estava no mesmo móduo e agora ficou no **kwarwp.kwarapp**.

See also:

Este código é uma modificação do código descrito em *Melhorando o Índio*.

```
class Piche(Vazio):
    """ Poça de Piche que gruda o índio se ele cair nela.

        :param imagem: A figura representando o índio na posição indicada.
        :param x: Coluna em que o elemento será posicionado.
        :param y: Cinha em que o elemento será posicionado.
        :param cena: Cena em que o elemento será posicionado.
        :param taba: Representa a taba onde o índio faz o desafio.
    """

    def __init__(self, imagem, x, y, cena, taba):
        from kwarwp.kwarapp import Kwarwp
        """Importando localmente o Kwarwp para evitar referência circular."""
        ... # copie o resto do tutorial anterior


class Oca(Piche):
    """  A Oca é o destino final do índio, não poderá sair se ele entrar nela.

        :param imagem: A figura representando o índio na posição indicada.
        :param x: Coluna em que o elemento será posicionado.
        :param y: Cinha em que o elemento será posicionado.
        :param cena: Cena em que o elemento será posicionado.
        :param taba: Representa a taba onde o índio faz o desafio.
    """

    def _pede_sair(self):
        """Objeto tenta sair mas não é autorizado"""
        ... # copie o resto do tutorial anterior
```

O Objeto Nulo

O Objeto Nulo foi extraído em uma classe própria.

```
class Nulo:
    """Objeto nulo que responde passivamente a todas as requisições."""
    def __init__(self):
        self.pegar = self.ocupa = self.nulo

    def nulo(self, *_, **__):
        """Método nulo, responde passivamente a todas as chamadas.

        :param _: aceita todos os argumentos posicionais.
        :param __: aceita todos os argumentos nomeados.
        :return: retorna o próprio objeto nulo.
        """
        return self

NULO = Nulo()
```

Proxy - Passo a Passo

Ao se colocar mais de um comando no método **Indio.executa** (), todos executam de imediato e não se observa o que acontece entre a primeira posição do índio e a última. Para que a execução dos comandos não seja imediata, temos que intermediar o fornecimento de comandos entre o indio e o vitollino. Para isso usaremos dois padrões o Proxy e o

Command. O Proxy será o intemediário que vai regular o fornecimento dos comandos ao Vitollino. O Command vai tratar cada comando como um objeto que poderá ser manipulado pelo Proxy.

Nesta parte do tutorial mostramos uma nova classe e seus comportamentos, a **Tora**. Ela tem comportamento que se assemelham ao **Vazio** e mais especificamente ao do seu descendente **Piche** e para isso usaremos a herança.

See also:

Este código é uma modificação do código descrito em *A Tora e Outras Partes*.

JogoProxy - Intermediando Comandos

A classe **JogoProxy** foi criada para estabelecer um controle no uso do **Jogo** Vitollino. Em vez de enviar os comandos diretamente para o Vitollino, esta classe tem um buffer que vai armazenando todos os comandos. Quando se quer executar um comando, um clique retira um comando do buffer e o executa.

Como a classe **Jogo** no vitolino implementa o padrão fábrica, **JogoProxy** també é uma fábrica criando proxies para **Cena** e **Elemento** invocados por **Jogo.c** e **Jogo.a**. Para facilitar a implementação da fila de comandos foi acrescentado um **JogoProxy.e** que sinaliza que este será o dono da fila de comandos.

A fila de comandos guarda uma coleção de objetos do padrão command. No entanto, devido ao fato de que toda função ou método no Python é um objeto, nenhuma infraestrutura extra é nescessária, o próprio método é guardado na fila de comandos.

See also:

O proxy, fábrica e comando são padrões descritos no livro Gang of Four. Veja os link externos O Padrão Proxy, Factory Method e Command

```
class JogoProxy():
""" Proxy que enfileira comandos gráficos.

:param vitollino: Empacota o engenho de jogo Vitollino.
:param elt: Elemento que vai ser encapsulado pelo proxy.
:param proxy: Referência para o objeto proxy parente.
:param master: Determina se este elemento vai ser mestre de comandos.
"""

def __init__(self, vitollino=None, elt=None, proxy=None, master=False):
    class AdaptaElemento(vitollino.a):
        """ Adapta um Elemento do Vitollino para agrupar ocupa e pos.

        """

        def ocupa(self, ocupante=None, pos=(0, 0)):
            # super().elt.pos = pos
            #vitollino.a.pos.fset(self, pos)
            ocupante = ocupante or NULO
            ocupante.pos = pos
            # print(f"AdaptaElemento pos: {self.pos}")
            super().ocupa(ocupante) if ocupante else None

    self.v = vitollino
    self.proxy = proxy or self
    self.master = master # or NULO
    self._corrente = self
    self.comandos = []
    self._ativa = False
```

(continues on next page)

(continued from previous page)

```
    """Cria um referência para o jogo do vitollino"""
    self.ae = AdaptaElemento
    """Cria um referência o Adapador de Eelementos"""
    self.elt = elt

@property
def siz(self):
    """Propriedade tamanho"""
    return self.elt.siz

def a(self, *args, **kwargs):
    """Método fábrica - Encapsula a criação de elementos

    :param args: coleção de argumentos posicionais.
    :param kwargs: coleção de argumentos nominais.
    :return: Proxy para um Elemento construído com estes argumentos.

    """
    return JogoProxy(elt=self.ae(*args, **kwargs), vitollino=self.v, proxy=self)

def e(self, *args, **kwargs):
    """Método fábrica - Encapsula a criação de elementos ativos, que executam scripts

    :param args: coleção de argumentos posicionais.
    :param kwargs: coleção de argumentos nominais.
    :return: Proxy para um Elemento construído com estes argumentos.

    """
    return JogoProxy(elt=self.ae(*args, **kwargs), vitollino=self.v, proxy=self,␣
↪master=True)

def cria(self):
    """Fábrica do JogoProxy"""
    return self

@property
def corrente(self):
    """Retorna o proxy master acertado no parente"""
    return self.proxy._corrente

@corrente.setter
def corrente(self, mestre):
    """Estabelece o proxy master"""
    self._corrente = mestre

def ativa(self):
    """Ativa bufferização do JogoProxy"""
    # JogoProxy.ATIVA = True
    self._ativa = True
    self.proxy.corrente = self

def lidar(self, metodo_command):
    """Lida com modo de operação do JogoProxy - bufferizado ou não"""
    self.ativa() if self.master else None
    print(self._ativa, self.proxy._ativa, metodo_command)
    self.corrente._enfileira(metodo_command) if self.proxy._ativa else self._
↪executa(metodo_command)
```

(continues on next page)

(continued from previous page)

```
def c(self, *args, **kwargs):
    """Método fábrica - Encapsula a criação de cenas - apenas delega.

    :param args: coleção de argumentos posicionais.
    :param kwargs: coleção de argumentos nominais.
    :return: Uma Cena do Vitollino construída com estes argumentos.

    """
    return self.v.c(*args, **kwargs)

@siz.setter
def siz(self, value):
    """Propriedade tamanho"""
    self.elt.siz = value

@property
def pos(self):
    """Propriedade posição"""
    return self.elt.pos

@property
def x(self):
    """Propriedade posição x"""
    return self.elt.x

@property
def y(self):
    """Propriedade posição y"""
    return self.elt.y

@pos.setter
def pos(self, value):
    """Propriedade posição"""
    def _command(val=value):
        self.elt.pos = val
    self.lidar(_command)

def ocupa(self, ocupante=None, pos=(0, 0)):
    """Muda a posição e atitude de um elemento"""
    def _command(val=ocupante):
        destino = val.elt if val else None
        self.elt.ocupa(destino, pos)
    self.lidar(_command)

def _enfileira(self, metodo_command):
    """Coloca um comando na fila"""
    self.comandos.append(metodo_command)

def _executa(self, metodo_command):
    """Executa imediamente um comando, não põe na fila"""
    metodo_command()

def executa(self, *_):
    """Tira e executa um comando na fila"""
    self.comandos.pop(0)() if self.comandos else None
```

Kwarwp Com Proxy

O Kwarwp é melhorado para suportar novos desenhos de índio, incluindo a Índia e o Pajé*.

See also:

Este código é uma modificação do código descrito em *A Tora e Outras Partes*.

```
class Kwarwp():
""" Jogo para ensino de programação.

    :param vitollino: Empacota o engenho de jogo Vitollino.
    :param mapa: Um texto representando o mapa do desafio.
    :param medidas: Um dicionário usado para redimensionar a tela.
    :param indios: Uma coleção com outros índios e outros comportamentos.
"""

def __init__(self, vitollino=None, mapa=None, medidas={}, indios=()):
    Vazio.VITOLLINO = self.v = vitollino()
    self.vitollino = vitollino
    """Referência estática para obter o engenho de jogo."""
    self.mapa = (mapa or MAPA_INICIO).split()
    """Cria um matriz com os elementos descritos em cada linha de texto"""
    self.taba = {}
    """Cria um dicionário com os elementos traduzidos a partir da interpretação do␣
↪mapa"""
    self.o_indio = NULO
    self.os_indios = []
    """Instância do personagem principal, o índio, vai ser atribuído pela fábrica do␣
↪índio"""
    self.lado, self.col, self.lin = 100, len(self.mapa[0]), len(self.mapa)+1
    """Largura da casa da arena dos desafios, número de colunas e linhas no mapa"""
    Vazio.LADO = self.lado
    """Referência estática para definir o lado do piso da casa."""
    w, h = self.col *self.lado, self.lin *self.lado
    medidas.update(width=w, height=f"{h}px")
    self.indios = deque(indios or [Indio])
    self.cena = self.cria(mapa=self.mapa) if vitollino else None

def cria(self, mapa=""):
    """ Fábrica de componentes.

    :param mapa: Um texto representando o mapa do desafio.
    """
    Fab = nt("Fab", "objeto imagem")
    """Esta tupla nomeada serve para definir o objeto construido e sua imagem."""

    fabrica = {
    "&": Fab(self.maloc, f"{IMGUR}dZQ8liT.jpg"), # OCA
    "^": Fab(self.indio, f"{IMGUR}UCWGCKR.png"), # INDIO
    "$": Fab(self.indio, f"{IMGUR}nvrwu0r.png"), # INDIA
    "p": Fab(self.indio, f"{IMGUR}HeiupbP.png"), # PAJE
    ".": Fab(self.vazio, f"{IMGUR}npb9Oej.png"), # VAZIO
    "_": Fab(self.coisa, f"{IMGUR}sGoKfvs.jpg"), # SOLO
    "#": Fab(self.atora, f"{IMGUR}0jSB27g.png"), # TORA
    "@": Fab(self.barra, f"{IMGUR}tLLVjfN.png"), # PICHE
    "~": Fab(self.coisa, f"{IMGUR}UAETaiP.gif"), # CEU
    "*": Fab(self.coisa, f"{IMGUR}PfodQmT.gif"), # SOL
```

(continues on next page)

(continued from previous page)

```
    "|": Fab(self.coisa, f"{IMGUR}uwYPNlz.png")  # CERCA
    }
    """Dicionário que define o tipo e a imagem do objeto para cada elemento."""
    mapa = mapa if mapa != "" else self.mapa
    """Cria um cenário com imagem de terra de chão batido, céu e sol"""
    mapa = self.mapa
    lado = self.lado
    cena = self.v.c(fabrica["_"].imagem)
    self.ceu = self.v.a(fabrica["~"].imagem, w=lado*self.col, h=lado-10, x=0, y=0,␣
↪cena=cena, vai=self.passo,
                        style={"padding-top": "10px", "text-align": "center"})
    """No argumento *vai*, associamos o clique no céu com o método **executa ()**␣
↪desta classe.
       O *ceu* agora é um argumento de instância e por isso é referenciado como␣
↪**self.ceu**.
    """
    sol = self.v.a(fabrica["*"].imagem, w=60, h=60, x=0, y=40, cena=cena, vai=self.
↪executa)
    """No argumento *vai*, associamos o clique no sol com o método **esquerda ()**␣
↪desta classe."""
    self.taba = {(i, j): fabrica[imagem].objeto(fabrica[imagem].imagem, x=i*lado,␣
↪y=j*lado+lado, cena=cena)
        for j, linha in enumerate(mapa) for i, imagem in enumerate(linha)}
    """Posiciona os elementos segundo suas posições i, j na matriz mapa"""
    cena.vai()
    return cena

def passo(self, *_):
    """ Ordena a execução do roteiro do índio.
    """
    # self.o_indio.esquerda()
    # self.v.executa()
    # self.o_indio.passo()

    [indio.passo() for indio in self.os_indios]


def executa(self, *_):
    """ Ordena a execução do roteiro do índio.
    """
    # self.v.ativa()
    # JogoProxy.ATIVA = True
    # self.o_indio.ativa()
    # self.o_indio.executa()
    # [indio.ativa() and indio.executa() for indio in self.os_indios]
    self.os_indios[0].ativa()
    self.v.ativa()
    self.os_indios[0].executa()

def indio(self, imagem, x, y, cena):
    """ Cria o personagem principal na arena do Kwarwp na posição definida.

    :param x: coluna em que o elemento será posicionado.
    :param y: linha em que o elemento será posicionado.
    :param cena: cena em que o elemento será posicionado.
    """
    self.o_indio = self.indios[0](imagem, x=1, y=0, cena=cena, taba=self,␣
↪vitollino=self.v)
```

(continues on next page)

(continued from previous page)

```
    """ O índio tem deslocamento zero, pois é relativo à vaga.
        O **x=1** serve para distinguir o indio de outros derivados.
    """
    self.o_indio.indio.vai = lambda *_: self.o_indio.pega()
    """o índio.vai é associado ao seu próprio metodo pega"""
    vaga = Vazio("", x=x, y=y, cena=cena, ocupante=self.o_indio)
    self.os_indios.append(self.o_indio)
    self.indios.rotate()
    """recebe a definição do próximo índio"""
    return vaga
```

Indio Com Proxy

O Indio é melhorado para operar com o JogoProxy.

See also:

Este código é uma modificação do código descrito em *A Tora e Outras Partes*.

```
class Indio():
""" Cria o personagem principal na arena do Kwarwp na posição definida.

    :param imagem: A figura representando o índio na posição indicada.
    :param x: Coluna em que o elemento será posicionado.
    :param y: Cinha em que o elemento será posicionado.
    :param cena: Cena em que o elemento será posicionado.
    :param taba: Representa a taba onde o índio faz o desafio.
    :param vitollino: Recebe referência para o vitollino ou proxy.
"""
AZIMUTE = Rosa(Ponto(0, -1),Ponto(1, 0),Ponto(0, 1),Ponto(-1, 0),)
"""Constante com os pares ordenados que representam os vetores unitários dos pontos␣
↪cardeais."""

def __init__(self, imagem, x, y, cena, taba, vitollino=None):
    self.vitollino = vitollino or Vazio.VITOLLINO
    self.lado = lado = Vazio.LADO
    self.azimute = self.AZIMUTE.n
    """índio olhando para o norte"""
    self.taba = taba
    self.vaga = self
    self.ocupante = NULO
    self.posicao = (x//lado,y//lado)
    self.indio = self.vitollino.e(imagem, w=lado, h=lado, x=x, y=y, cena=cena)
    self.x = x
    """Este x provisoriamente distingue o índio de outras coisas construídas com esta␣
↪classe"""
    if x:
        self.indio.siz = (lado*3, lado*4)
        """Define as proporções da folha de sprites"""
        self.gira()

def ativa(self):
    """ Ativa o proxy do índio para enfileirar comandos.
    """
    #self.vitollino.ativa()
```

(continues on next page)

(continued from previous page)

```
    self.indio.ativa()


def passo(self):
    self.indio.executa()
```

2.2 Indices and tables

- genindex
- modindex
- search

2.3 Manual Kwarwp

Jogo para aprendizado de programação.

Author Carlo E. T. Oliveira

Affiliation Universidade Federal do Rio de Janeiro

Homepage Projeto Kwarwp

2.3.1 Descrição

Um jogo de aventura que se joga aprendendo e criando programas na linguagem Python.

Este ambiente facilita a aprendizagem da linguagem Python.

O jogo é dirigido principalmente ao ensino de programação de computadores para jovens e crianças do ensino médio e fundamental.

2.3.2 Documentação

Kwarwp

Um jogo de aventura que se joga aprendendo e criando programas na linguagem Python.

Este ambiente facilita a aprendizagem da linguagem Python.

O jogo é dirigido principalmente ao ensino de programação de computadores para jovens e crianças do ensino médio e fundamental.

Documentação

Descrição dos módulos do Kwarwp.

Jogo para ensino de programação Python.

Classes neste módulo:

- *Kwarwp* Jogo para ensino de programação.
- *Indio* Personagem principal do jogo.
- *JogoProxy* Proxy que enfileira comandos gráficos.

Changelog

New in version 20.08.b1: Adicionou *JogoProxy* para realizar o passo a passo. Capacidade de gerenciar mais de um índio.

Changed in version 20.08.b0: Moveu constantes de classe VITOLLINO, LADO para Vazio. Moveu `Vazio`, `Oca`, `Piche` para kwarwpart.

New in version 20.08.a3: Movimentação do índio para *Indio.esquerda()* e *Indio.direita()*. Fala do índio: *Indio.fala()*. classes Oca e Piche, double dispatch para sair.

New in version 20.08.a2: Classe Vazio que recebe cada componente do mapa. Movimentação do índio é feita pulando para outro vazio.

New in version 20.08.a1: Classe Indio que executa roteiro e anda.

New in version 20.08.a0: Fábrica de componentes classe Indio. Usa um mapa de caracteres para colocar os elementos.

New in version 20.07: classe Kwarwp.

`kwarwp.kwarapp.`**`IMGUR = 'https://imgur.com/'`**
Prefixo do site imgur.

`class` `kwarwp.kwarapp.`**`Indio`**(*imagem*, *x*, *y*, *cena*, *taba*, *vitollino=None*)
Cria o personagem principal na arena do Kwarwp na posição definida.

Parameters

- **`imagem`** – A figura representando o índio na posição indicada.
- **`x`** – Coluna em que o elemento será posicionado.

- **y** – Cinha em que o elemento será posicionado.
- **cena** – Cena em que o elemento será posicionado.
- **taba** – Representa a taba onde o índio faz o desafio.

AZIMUTE = Rosa(n=Ponto(x=0, y=-1), l=Ponto(x=1, y=0), s=Ponto(x=0, y=1), o=Ponto(x=-1,
Constante com os pares ordenados que representam os vetores unitários dos pontos cardeais.

acessa(*ocupante*)
Pedido de acesso a essa posição, delegada ao ocupante pela vaga.

Parameters **ocupante** – O componente candidato a ocupar a vaga já ocupada pelo índio.

No caso do índio, ele age como um obstáculo e não prossegue com o protocolo.

anda()
Objeto tenta sair, tem que consultar a vaga onde está

ativa()
Ativa o proxy do índio para enfileirar comandos.

azimute = None
índio olhando para o norte

direita()
Faz o índio mudar da direção em que está olhando para a direita.

elt
A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento no outro .

No caso do índio, retorna o elt do elemento do atributo **self.indio**.

empurra()
Objeto tenta sair, tem que consultar a vaga onde está

esquerda()
Faz o índio mudar da direção em que está olhando para a esquerda.

executa()
Roteiro do índio. Conjunto de comandos para ele executar.

fala(*texto=''*)
O índio fala um texto dado.

Parameters **texto** – O texto a ser falado.

gira()
Modifica a figura (Sprite) do índio mostrando para onde está indo.

larga()
tenta largar o objeto que está segurando

mostra(*vaga=None*)
Modifica a figura (Sprite) do índio mostrando para onde está indo.

ocupa(*vaga*, *_)
Pedido por uma vaga para que ocupe a posição nela.

Parameters **vaga** – A vaga que será ocupada pelo componente.

No caso do índio, requisita que a vaga seja ocupada por ele.

ocupou(*ocupante*)
O candidato à vaga decidiu ocupá-la e efetivamente entra neste espaço.

Parameters **ocupante** – O canditato a ocupar a posição corrente.

Este ocupante vai entrar no elemento do Vitollino e definitivamente se tornar o ocupante da vaga. Com isso ele troca o estado do método acessa para primeiro consultar a si mesmo, o ocupante corrente usando o protocolo definido em **_valida_acessa ()**

passo()

pega()
tenta pegar o objeto que está diante dele

sai()
Rotina de saída falsa, o objeto Indio é usado como uma vaga nula.

sair()
Objeto de posse do índio tenta sair e é autorizado

siga()
Objeto tentou sair e foi autorizado

x = None
Este x provisoriamente distingue o índio de outras coisas construídas com esta classe

class kwarwp.kwarapp.**JogoProxy**(*vitollino=None*, *elt=None*, *proxy=None*, *master=False*)
Proxy que enfileira comandos gráficos.

Parameters

- **vitollino** – Empacota o engenho de jogo Vitollino.
- **elt** – Elemento que vai ser encapsulado pelo proxy.
- **proxy** – Referência para o objeto proxy parente.
- **master** – Determina se este elemento vai ser mestre de comandos.

a(**args*, ***kwargs*)
Método fábrica - Encapsula a criação de elementos

Parameters

- **args** – coleção de argumentos posicionais.
- **kwargs** – coleção de argumentos nominais.

Returns Proxy para um Elemento construído com estes argumentos.

ae = None
Cria um referência o Adapador de Eelementos

ativa()
Ativa Fábrica do JogoProxy

c(**args*, ***kwargs*)
Encapsula a criação de cenas - apenas delega.

Parameters

- **args** – coleção de argumentos posicionais.
- **kwargs** – coleção de argumentos nominais.

Returns Uma Cena do Vitollino construída com estes argumentos.

corrente
Ativa Fábrica do JogoProxy

cria()
Fábrica do JogoProxy

e(**args*, ***kwargs*)
Método fábrica - Encapsula a criação de elementos ativos, que executam scripts

Parameters

- **args** – coleção de argumentos posicionais.
- **kwargs** – coleção de argumentos nominais.

Returns Proxy para um Elemento construído com estes argumentos.

elt
Propriedade elemento

executa(*_)
Tira e executa um comando na fila

lidar(*metodo*)
Lida com modo de operação do JogoProxy

ocupa(*ocupante=None*, *pos=(0, 0)*)
Muda a posição e atitude de um elemento

pos
Propriedade posição

siz
Propriedade tamanho

x
Propriedade posição x

y
Propriedade posição y

class kwarwp.kwarapp.**Kwarwp**(*vitollino=None*, *mapa=None*, *medidas={}*, *indios=()*)
Jogo para ensino de programação.

Parameters

- **vitollino** – Empacota o engenho de jogo Vitollino.
- **mapa** – Um texto representando o mapa do desafio.
- **medidas** – Um dicionário usado para redimensionar a tela.

KW = None

apedra(*imagem*, *x*, *y*, *cena*)
Cria uma pedra na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

Cria uma vaga vazia e coloca o componente dentro dela.

atora(*imagem*, *x*, *y*, *cena*)
Cria uma tora na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

Cria uma vaga vazia e coloca o componente dentro dela.

barra(*imagem*, *x*, *y*, *cena*)
Cria uma armadilha na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

Cria uma vaga vazia e coloca o componente dentro dela.

coisa(*imagem*, *x*, *y*, *cena*)
Cria um elemento na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

Cria uma vaga vazia e coloca o componente dentro dela.

col = None
Largura da casa da arena dos desafios, número de colunas e linhas no mapa

cria(*mapa=''*)
Fábrica de componentes.

Parameters **mapa** – Um texto representando o mapa do desafio.

executa(*_)
Ordena a execução do roteiro do índio.

fala(*texto=''*)
O Kwarwp é aqui usado para falar algo que ficará escrito no céu.

indio(*imagem*, *x*, *y*, *cena*)
Cria o personagem principal na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

lado = None
Largura da casa da arena dos desafios, número de colunas e linhas no mapa

lin = None
Largura da casa da arena dos desafios, número de colunas e linhas no mapa

maloc(*imagem*, *x*, *y*, *cena*)
Cria uma maloca na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

Cria uma vaga vazia e coloca o componente dentro dela.

mapa = None
Cria um matriz com os elementos descritos em cada linha de texto

ocupa(*_)
O Kwarwp é aqui usado como um ocupante falso, o pedido de ocupar é ignorado.

os_indios = None
Instância do personagem principal, o índio, vai ser atribuído pela fábrica do índio

passo(*_)
Ordena a execução do roteiro do índio.

sai(*_)
O Kwarwp é aqui usado como uma vaga falsa, o pedido de sair é ignorado.

taba = None
Cria um dicionário com os elementos traduzidos a partir da interpretação do mapa

vazio(*imagem*, *x*, *y*, *cena*)
Cria um espaço vazio na arena do Kwarwp na posição definida.

Parameters

- **x** – coluna em que o elemento será posicionado.
- **y** – linha em que o elemento será posicionado.
- **cena** – cena em que o elemento será posicionado.

vitollino = None
Referência estática para obter o engenho de jogo.

kwarwp.kwarapp.**MAPA_INICIO = '\n..#^¨..\n'**
Mapa com o posicionamento inicial dos elementos.

class kwarwp.kwarapp.**Ponto**(*x*, *y*)
Par de coordenadas na direção horizontal (x) e vertiacal (y).

x
Alias for field number 0

y
Alias for field number 1

class kwarwp.kwarapp.**Rosa**(*n*, *l*, *s*, *o*)
Rosa dos ventos com as direções norte, leste, sul e oeste.

l
Alias for field number 1

n
Alias for field number 0

o
Alias for field number 3

s
Alias for field number 2

kwarwp.kwarapp.**main**(*vitollino*, *medidas={}*, *mapa=None*, *indios=()*)
Rotina principal que invoca a classe Kwarwp.

Parameters

- **vitollino** – Empacota o engenho de jogo Vitollino.
- **medidas** – Um dicionário usado para redimensionar a tela.

Indices and tables

- genindex
- modindex
- search

Partes do Kwarwp

Um jogo de aventura que se joga aprendendo e criando programas na linguagem Python.

Este ambiente facilita a aprendizagem da linguagem Python.

O jogo é dirigido principalmente ao ensino de programação de computadores para jovens e crianças do ensino médio e fundamental.

Documentação

Descrição dos módulos de partes do Kwarwp.

Jogo para ensino de programação Python.

Classes neste módulo:

- *Vazio* Espaço vago na arena do desafio.
- *Oca* Destino final da aventura.
- *Piche* Uma armadilha para prender o índio.
- *Tora* Uma tora que o índio pode pegar.
- *Nulo* Objeto Nulo para default em argumentos.

Changelog

Changed in version 20.08.b1: modifica *Vazio.ocupou()* para receber também a posição. - *Nulo* Objeto nulo passivo a todas as requisições.

Changed in version 20.08.b0: Moveu constantes de classe VITOLLINO, LADO para Vazio.

New in version 20.08.b0: Moveu *Vazio*, *Oca*, *Piche* para cá. Adicionou *Tora* e classe *Nulo*

class kwarwp.kwarwpart.**Nulo**
Objeto nulo que responde passivamente a todas as requisições.

nulo(*_, **__)
Método nulo, responde passivamente a todas as chamadas.

Parameters

- _ – aceita todos os argumentos posicionais.
- ___ – aceita todos os argumentos nomeados.

Returns retorna o próprio objeto nulo.

class kwarwp.kwarwpart.**Oca**(*imagem*, *x*, *y*, *cena*, *taba*)
A Oca é o destino final do índio, não poderá sair se ele entrar nela.

Parameters

- **imagem** – A figura representando o índio na posição indicada.
- **x** – Coluna em que o elemento será posicionado.
- **y** – Cinha em que o elemento será posicionado.
- **cena** – Cena em que o elemento será posicionado.
- **taba** – Representa a taba onde o índio faz o desafio.

class kwarwp.kwarwpart.**Pedra**(*imagem*, *x*, *y*, *cena*, *taba*)
A Pedra é um uma coisa muito pesada que o índio só consegue empurrar.

Parameters

- **imagem** – A figura representando o índio na posição indicada.
- **x** – Coluna em que o elemento será posicionado.
- **y** – Linha em que o elemento será posicionado.
- **cena** – Cena em que o elemento será posicionado.
- **taba** – Representa a taba onde o índio faz o desafio.

pegar(*requisitante*)
Consulta o ocupante atual se há permissão para pegar e entregar ao requistante.

Parameters **requistante** – O ator querendo pegar o objeto.

class kwarwp.kwarwpart.**Piche**(*imagem*, *x*, *y*, *cena*, *taba*)
Poça de Piche que gruda o índio se ele cair nela.

Parameters

- **imagem** – A figura representando o índio na posição indicada.
- **x** – Coluna em que o elemento será posicionado.
- **y** – Cinha em que o elemento será posicionado.
- **cena** – Cena em que o elemento será posicionado.
- **taba** – Representa a taba onde o índio faz o desafio.

acessa = None
O **acessa ()** é usado como método dinâmico, variando com o estado da vaga. Inicialmente tem o comportamento de **_acessa ()** que é o estado vago, aceitando ocupantes

elt
A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento no outro .

No caso do espaço vazio, vai retornar um elemento que não contém nada.

ocupa(*vaga*)
Pedido por uma vaga para que ocupe a posição nela.

Parameters **vaga** – A vaga que será ocupada pelo componente.

No caso do índio, requisita que a vaga seja ocupada por ele.

sair = None
O **sair ()** é usado como método dinâmico, variando com o estado da vaga. Inicialmente tem o comportamento de **_sair ()** que é o estado vago, aceitando ocupantes

class kwarwp.kwarwpart.**Tora**(*imagem*, *x*, *y*, *cena*, *taba*)
A Tora é um pedaço de tronco cortado que o índio pode carregar ou empurrar.

Parameters

- **imagem** – A figura representando o índio na posição indicada.
- **x** – Coluna em que o elemento será posicionado.
- **y** – Linha em que o elemento será posicionado.
- **cena** – Cena em que o elemento será posicionado.
- **taba** – Representa a taba onde o índio faz o desafio.

elt
A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento no outro .

No caso da tora, retorna o elt do elemento do atributo **self.vazio**.

empurrar(*empurrante*, *azimute*)
Consulta o ocupante atual se há permissão para pegar e entregar ao requistante.

Parameters **requistante** – O ator querendo pegar o objeto.

ocupa(*vaga*)
Pedido por uma vaga para que ocupe a posição nela.

Parameters **vaga** – A vaga que será ocupada pelo componente.

No caso do índio, requisita que a vaga seja ocupada por ele.

pegar(*requisitante*)
Consulta o ocupante atual se há permissão para pegar e entregar ao requistante.

Parameters **requistante** – O ator querendo pegar o objeto.

posicao
A propriedade posição faz parte do protocolo do double dispatch com o Indio .

No caso da tora, retorna o a posição do atributo **self.vaga**.

class kwarwp.kwarwpart.**Vazio**(*imagem*, *x*, *y*, *cena*, *taba*, *ocupante=None*)
Cria um espaço vazio na taba, para alojar os elementos do desafio.

Parameters

- **imagem** – A figura representando o índio na posição indicada.
- **x** – Coluna em que o elemento será posicionado.
- **y** – Cinha em que o elemento será posicionado.

- **cena** – Cena em que o elemento será posicionado.
- **taba** – Referência onde ele pode encontrar a taba.
- **ocupante** – Objeto que ocupa inicialmente a vaga.

LADO = None

VITOLLINO = None

acessa = None
O **acessa ()** é usado como método dinâmico, variando com o estado da vaga. Inicialmente tem o comportamento de **_acessa ()** que é o estado vago, aceitando ocupantes

acessar (*ocupante*, *azimute*)
Faz o índio caminhar na direção em que está olhando.

elt
A propriedade elt faz parte do protocolo do Vitollino para anexar um elemento no outro .

No caso do espaço vazio, vai retornar um elemento que não contém nada.

empurrar (*requisitante*, *azimute*)
Consulta o ocupante atual se há permissão para pegar e entregar ao requistante.

Parameters **requistante** – O ator querendo pegar o objeto.

limpa ()
Pedido por um ocupante para ele seja eliminado do jogo.

ocupa (*vaga*, *_)
Pedido por uma vaga para que ocupe a posição nela.

No caso do espaço vazio, não faz nada.

ocupante = None
O ocupante se não for fornecido é encenado pelo próprio vazio, agindo como nulo

ocupou (*ocupante*, *pos=(0, 0)*)
O candidato à vaga decidiu ocupá-la e efetivamente entra neste espaço.

Parameters

- **ocupante** – O canditato a ocupar a posição corrente.
- **pos** – A posição (atitude) do sprite do ocupante.

Este ocupante vai entrar no elemento do Vitollino e definitivamente se tornar o ocupante da vaga. Com isso ele troca o estado do método acessa para primeiro consultar a si mesmo, o ocupante corrente usando o protocolo definido em **_valida_acessa ()**

pegar (*requisitante*)
Consulta o ocupante atual se há permissão para pegar e entregar ao requistante.

Parameters **requistante** – O ator querendo pegar o objeto.

sai ()
Pedido por um ocupante para que desocupe a posição nela.

sair = None
O **sair ()** é usado como método dinâmico, variando com o estado da vaga. Inicialmente tem o comportamento de **_sair ()** que é o estado leniente, aceitando saidas

Indices and tables

- genindex
- modindex
- search

2.4 Indices and tables

- genindex
- modindex
- search

Laboratório de Automação de Sistemas Educacionais

LABASE - NCE - UFRJ

CHAPTER 3

Indices and tables

genindex	modindex	search

Python Module Index

k

Index