Vitollino Documentation

Release 1.1.0

Carlo Oliveira

Apr 02, 2018

Contents

Aqui vamos ter uma introdução rápida de como programar jogos para Web usando Python. Na verdade vamos usar o Brython que é o Python que funciona dentro de um navegador web como o Firefox.

CHAPTER 1

Manual

1.1 Vitollino - Classes Principais

See also:

Auxiliares *Vitollino - Classes Auxiliares*

1.1.1 Jogo

See also:

Cena	*_spy.vitollino.vitollino.Cena*
Elemento	*_spy.vitollino.vitollino.Elemento*
Salao	*_spy.vitollino.vitollino.Salao*
Sala	*_spy.vitollino.vitollino.Sala*
Texto	*_spy.vitollino.vitollino.Texto*
Portal	*_spy.vitollino.vitollino.Portal*
Inventario	*_spy.vitollino.vitollino.Inventario*

class _spy.vitollino.vitollino.**Jogo**

Bases: object

algo
Acessa a classe Elemento

cena
Acessa a classe Cena

nota
Acessa a classe Texto

quarto
Acessa a classe Sala

sala
Acessa a classe Salao

1.1.2 Musica

class _spy.vitollino.vitollino.**Musica**(*sound*, *loop=True*, *autoplay=True*, *sound_type='audio/mpeg'*)

Bases: object

1.1.3 Inventario

See also:

Elemento *_spy.vitollino.vitollino.Elemento*

class _spy.vitollino.vitollino.**Inventario**(*tela=<browser.BrythonMock object>*)

Bases: object

Os objetos que estão de posse do jogador.

Parameters **tela** – Div do HTML onde o inventário será anexado

GID = '00000000000000000000'

bota(*nome_item*, *item=''*, *acao=None*)

Os objetos que estão de posse do jogador.

```
>>> inv.bota("uma_coisa")
>>> "uma_coisa" in inv.inventario
True
```

Parameters

- **nome_item** – uma string com o nome do item, ele será criado e colocado no inventário
- **item** – URL da imagem do item nomeado por nome_item
- **acao** – ação associada com o item nomeado quando ele é clicado

desmonta(*_=0*)

inicia()

monta(*_=0*)

mostra(*_=0*)

score(*casa*, *carta*, *move*, *ponto*, *valor*)

static send(*operation*, *data*, *action=<function Inventario.<lambda>>*, *method='POST'*)

tira(*nome_item*)

1.1.4 Cena

class _spy.vitollino.vitollino.**Cena**(*img=''*, *esquerda=<CenaNula>*, *direita=<CenaNula>*, *meio=<CenaNula>*, *vai=None*, *nome=''*, *xy=(0, 0)*, *score={}*, ***kwargs*)

Bases: object

Use para construir uma cena.

```
from _spy.vitollino import Cena

cena_esq = Cena(img="esq.jpg")
cena_mei = Cena(img="mei.jpg", cena_esq)
cena_mei.vai()
```

Parameters

- **img** (*str*) – URL da imagem
- **esquerda** (Cena) – Cena que está à esquerda desta
- **direita** (Cena) – Cena que está à direita desta
- **meio** (Cena) – Cena que está à frente desta
- **vai** – Função a ser chamada no lugar da self.vai nativa

bota(*nome_item*)

static c(***cenas*)

portal(*esquerda=None*, *direita=None*, *meio=None*, ***kwargs*)

static q(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

static s(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

sai(*saida*)

score(***kwargs*)

tira(*item*)

vai(*ev=<NoEvent>*)

vai_direita(*_=0*)

vai_esquerda(*_=0*)

vai_meio(*_=0*)

1.1.5 Sala

class _spy.vitollino.vitollino.**Sala**(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

Bases: object

static c(***cenas*)

leste

norte

oeste

p()

sul

1.1.6 Salao

class _spy.vitollino.vitollino.**Salao**(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

Bases: *_spy.vitollino.vitollino.Sala*

static c(***cenas*)

leste

norte

oeste

p()

sul

1.1.7 Elemento

class _spy.vitollino.vitollino.**Elemento**(*img=''*, *vai=None*, *style={}*, *tit=''*, *alt=''*, *cena=Inventario*, *score={}*, ***kwargs*)

Bases: object

Um objeto de interação que é representado por uma imagem em uma cena.

papel = Elemento(img="papel.png", tit="caderno de notas", vai=pega_papel, style=dict(left=350, top=550, width=60))

Parameters

- **img** – URL de uma imagem
- **vai** – função executada quando se clica no objeto
- **style** – dicionário com dimensões do objeto {"left": ..., "top": ..., width: ..., height: ...}
- **tit** – Texto que aparece quando se passa o mouse sobre o objeto
- **alt** – Texto para leitores de tela
- **cena** – cena alternativa onde o objeto vai ser colocado
- **score** – determina o score para este elemento
- **kwargs** – lista de parametros nome=URL que geram elementos com este nome e a dada imagem

classmethod c(***kwargs*)

entra(*cena*, *style={}*)

limbo = <browser.BrythonMock object>

score(***kwargs*)

1.1.8 Portal

class _spy.vitollino.vitollino.**Portal**(*cena=None*, *debug_=False*, ***kwargs*)

Bases: object

L = {'min-height': '60%', 'width': '10%', 'cursor': 'e-resize', 'left': '90%', 'po

N = {'min-height': '20%', 'width': '60%', 'cursor': 'n-resize', 'left': '20%', 'po

O = {'min-height': '60%', 'width': '10%', 'cursor': 'w-resize', 'left': 0, 'positi

PORTAIS = {'S': {'min-height': '10%', 'width': '60%', 'bottom': 0, 'left': '20%',

S = {'min-height': '10%', 'width': '60%', 'bottom': 0, 'left': '20%', 'position':

Z = {'min-height': '10%', 'margin': '0%', 'width': '10%', 'cursor': 'zoom-in', 'po

p(***kwargs*)

vai(**_*)

1.1.9 Popup

class _spy.vitollino.vitollino.**Popup**(*cena*, *tit=''*, *txt=''*, *vai=None*, ***kwargs*)

Bases: object

POP = <Popup>

static d(*cena*, *tit=''*, *txt=''*)

vai()

1.1.10 Texto

class _spy.vitollino.vitollino.**Texto**(*cena=<CenaNula>*, *tit=''*, *txt=''*, ***kwargs*)

Bases: *_spy.vitollino.vitollino.Popup*

POP = <Popup>

d(*cena*, *tit=''*, *txt=''*)

esconde(*ev=<NoEvent>*)

mostra(*tit=''*, *txt=''*, ***kwargs*)

static texto(*tit=''*, *txt=''*, ***kwars*)

vai(*ev=<NoEvent>*)

1.1.11 Labirinto

class _spy.vitollino.vitollino.**Labirinto**(*c=<CenaNula>*, *n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*)

Bases: object

static m(*cenas*)

1.1.12 Dropper

class _spy.vitollino.vitollino.**Dropper**(*dropper*)

Bases: object

drag_start(*ev*)

mouse_over(*ev*)

1.1.13 Dragger

class _spy.vitollino.vitollino.**Dragger**(*dragger*)
Bases: object

ACTION = ''

POINTER = ''

drag_start(*ev*)

highten(*dy*)

mouse_down(*ev*)

static mouse_over(*ev*)

mouse_up(_)

no_mouse_move(*ev*)

pre_mouse_down(*ev*)

pre_mouse_move(*ev*)

static pre_mouse_over(*ev*)

pre_mouse_up(_)

widen(*dx*)

1.1.14 Droppable

class _spy.vitollino.vitollino.**Droppable**(*droppable*, *dropper_name=''*, *action=None*, *cursor=None*)
Bases: object

drag_over(*ev*)

drop(*ev*)

1.1.15 Cursor

class _spy.vitollino.vitollino.**Cursor**(*alvo*, *cena=<browser.BrythonMock object>*)
Bases: object

Note: Biblioteca Cliente para criação de Jogos.

1.2 Vitollino - Classes Auxiliares

See also:

Classes Principais *Vitollino - Classes Principais*

1.2.1 NoEv

See also:

Elemento *_spy.vitollino.vitollino.singleton()*

class _spy.vitollino.vitollino.**NoEv**
Bases: *_spy.vitollino.vitollino.NoEv*

Representa um evento vazio.

```
>>> print(ev.x, ev.y)
-100 -100
```

stopPropagation()

x = -100

y = -100

1.2.2 SalaCenaNula

See also:

Elemento *_spy.vitollino.vitollino.singleton()*

class _spy.vitollino.vitollino.**SalaCenaNula**
Bases: *_spy.vitollino.vitollino.SalaCenaNula*

Define uma Sala ou uma Cena vazia.

```
>>> cena = Cena(SalaCenaNula())  # A próxima cena
>>> uma_cena = Cena(SalaCenaNula(), cena)  # Cena nula à esquerda, proxima no meio
>>> uma_cena.vai_esquerda()  # tenta navegar para a cena à esquerda
>>> # não vai, pois a cena é nula e não deixa que se navegue para ela
>>> print(INVENTARIO.cena == cena)
True
```

Deve ser usado quando um parâmetro requer uma cena mas não deve ter uma cena válida ali.

init()

portal(*_, **__)

vai()

1.2.3 parametrized

_spy.vitollino.vitollino.**parametrized**(*dec*)

1.2.4 wraps_class_to_mimic_wrapped

_spy.vitollino.vitollino.**wraps_class_to_mimic_wrapped**(*original_cls*)
Empacota uma classe decoradora para que apareça corretamente nos documentos.

```
>>> @wraps_class_to_mimic_wrapped
... class Exemplo:
...     ...
...
>>> print(Exemplo.__doc__)
Atualiza wrapper_cls para se assemelhar à classe original_cls.
```

Parameters **original_cls** – A Classe a ser empacotada

Returns O empacotador da classe

1.2.5 singleton

_spy.vitollino.vitollino.**singleton**(*cls_to_decorate*)

Decora um classe para ser um singleton e retornar sempre a mesma instância.

```
>>> @singleton
... class Mono:
...     def __init__(self):
...             self.x = 0
...
>>> Mono().x, Mono().x = 1, 2
>>> print(Mono().x == Mono().x, Mono().x)
True 2
```

Parameters **cls_to_decorate** – A classe para ser definida como singleton

Returns O decorador de singleton

1.2.6 main

_spy.vitollino.vitollino.**main**()

1.2.7 setup

_spy.vitollino.vitollino.**__setup__**()

Note: Biblioteca Cliente para criação de Jogos.

CHAPTER 2

Tutorial

2.1 Vitollino - Jogo da Marcela

2.1.1 Treinamento de Manuseio Alimentar

See also:

Jardim	*Vitollino - Jardim Radical*
Vitollino	*Primeiro Cenário do Jogo*

Treinamento de Manuseio de Alimentos.

Gerador de labirintos e jogos tipo *'novel'*.

Vitollino em Github

class lab.views.marcela.**Config**

Bases: object

CONFIGURA Dicionário de configuração das cenas.

```
CONFIGURA = dict(origem=["vestiário#armário#Asseio#Por o avental", True,␣
↪dict(left=429))
```

Origem, Destino, Título e Texto	Com Popup	Hot Spot
"Origem#Destino#Título#Texto"	True or False	{"top":1}

Origem, Destino, Título e Texto String com partes separadas por **#**

Origem Nome da cena de origem, precisa ser uma chave de **CONFIGURA**

Destino Nome da cena de destino, precisa ser uma chave de **CONFIGURA**

Título Título do popup de texto

Texto Texto do popup de texto

Com Popup Determina se vai aparecer o popup de texto na transição de cenas

Hot Spot Dicionário com as dimensões do hot spot que vai receber o click

```
dict(left=429, top=112, width=109, height=300)
```

CONFIGURA = {'vestiário': ['vestiário#abriu_o_armário#Asseio#Você deve por o aventa

class lab.views.marcela.**JogoMarcela**(*legendas=['vestiário', 'abriu_o_armário', 'o_asseio', 'acionou_a_pia', 'molhando_as_mãos', 'usando_sabão', 'as_bactérias', 'enxaguando', 'secando', 'descontaminando', 'saindo'], momentos=[(1, 1), (1, 2), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (3, 1)]*)

Bases: object

Constroi o jogo completo da Marcela.

Parameters

- **legendas** – lista contendo nomes das cenas *[<nome da cena>, . . .]*
- **momentos** – lista de tuplas indicando o quadro e o momento *[(<q0>, <m0>), . . .]*

static _cria_cenas(*cenas*)

Cria um conjunto de objetos **Cena** a partir de um dicionário.

Usa a função cria de criar cenas do Vitollino: *JOGO.c.c*

:param cenas:dicionário contendo *<nome da cena>: <url da imagem>* :return: cenário, uma lista de quadros criados

static _decorador_do_vai_do_texto(*port*)

Decorador do texto para refinamento, publica as dimensões do cursor no popup de texto.

Parameters **port** – portal que vai ser decorado

Returns Nenhum

_inicia_jogo()

Configura o momento inicial e ativa a primeira tela.

Returns Nenhum

configura_momentos(*cena*)

Configura a cena do momento para ativar seu portal segundo os dados em **CONFIGURA**.

configura_portal_com_texto Portal decorado com texto.

@JOGO.n.texto decorador que adiciona um popup de texto. A ação do portal acontece quando se fecha o popup.

```
um_portal = configura_portal_com_texto("vestiário", "armário", hot_
↪spot=dict(left=10, top=90)):
```

Parameters **cena** – Nome do momento a ser configurado, tem que ser chave de **CONFIGURA**.

Returns Proxi da cena construída pela classe interna **PreviaDoMomento**.

class lab.views.marcela.**PreviaDoMomento**(*jogo*, *destino*)
Bases: object

Cria um proxi para a cena que ativa os portais somente quando é montrada pelo comando vai.

self.destino nome da cena que surge ao clicar no portal.

self._destino objeto cena referente à cena destino, recuperada como atributo da classe Cena.

Parameters

- **jogo** – Instância da classe JogoMarcela
- **destino** – nome da cena destino

vai(*_, **__)
Mostra a cena destino e configura o portal nela.

Parameters

- _ – captura lista de argumento para evitar erros
- __ – captura dicionário de argumento para evitar erros

Returns Nenhum

lab.views.marcela.**main**(*_)
Chamada do jogo, feita a partir do HTML.

Parameters _ – Parametros recebidos do HTML

Returns Instância do Jogo da Marcela

Note: Ambiente de treinamento para manuseio alimentar.

2.2 Vitollino - Jardim Radical

2.2.1 Aventura no Jardim Botânico

See also:

Marcela	*Vitollino - Jogo da Marcela*
Vitollino	*Primeiro Cenário do Jogo*

ROTEIRO DO GAME JARDIM RADICAL.

Gerador de labirintos e jogos tipo *'novel'*.

Sequência de forma lógica para o funcionamento do jogo, ou seja, "costurar" as histórias das cenas.

See also:

Vitollino em Github

class lab.views.jardim.**Config**
Bases: object

CONFIGURA Dicionário de configuração das cenas.

```
CONFIGURA = dict(origem=["vestiário#armário#Asseio#Por o avental", True,␣
↪dict(left=429))
```

Origem, Destino, Título e Texto	Com Popup	Hot Spot
"Origem#Destino#Título#Texto"	True or False	{"top":1}

Origem, Destino, Título e Texto String com partes separadas por **#**

Origem Nome da cena de origem, precisa ser uma chave de **CONFIGURA**

Destino Nome da cena de destino, precisa ser uma chave de **CONFIGURA**

Título Título do popup de texto

Texto Texto do popup de texto

Com Popup Determina se vai aparecer o popup de texto na transição de cenas

Hot Spot Dicionário com as dimensões do hot spot que vai receber o click

```
dict(left=429, top=112, width=109, height=300)
```

CONFIGURA = {'insetivoras': ['insetivoras#jambeiros#insetivoras#jambeiros', True, {'t

class lab.views.jardim.**JogoJardim**(*legendas=['abrico', 'academia', 'aleiapalmeiras', 'bambus', 'bambuzal', 'bromeliario', 'cascata', 'clarisse', 'comoro', 'entrada', 'gruta', 'guarita', 'herborizada', 'insetivoras', 'jambeiros', 'japones', 'lago', 'macaco', 'mexicano', 'mirante', 'narciso', 'orquidario', 'ossanha', 'palmeiras', 'pesquisa', 'portal', 'portao', 'relogio', 'serpabrandao', 'tartarugas', 'tetis', 'tomjobim', 'vitoriaregia', 'xochipili']*)

Bases: `object`

Constroi o jogo completo do Jardim.

Parameters **legendas** – lista contendo nomes das cenas *[<nome da cena>, . . .]*

static _cria_cenas(*cenas*)

Cria um conjunto de objetos **Cena** a partir de um dicionário.

Usa a função cria de criar cenas do Vitollino: *JOGO.c.c*

:param cenas:dicionário contendo *<nome da cena>: <url da imagem>* :return: cenário, uma lista de quadros criados

static _decorador_do_vai_do_texto(*port*)

Decorador do texto para refinamento, publica as dimensões do cursor no popup de texto.

Parameters **port** – portal que vai ser decorado

Returns Nenhum

_inicia_jogo()

Configura o momento inicial e ativa a primeira tela.

Returns Nenhum

configura_momentos(*cena*)

Configura a cena do momento para ativar seu portal segundo os dados em **CONFIGURA**.

configura_portal_com_texto Portal decorado com texto.

@JOGO.n.texto decorador que adiciona um popup de texto. A ação do portal acontece quando se fecha o popup.

```
um_portal = configura_portal_com_texto("vestiário", "armário", hot_
↪spot=dict(left=10, top=90)):
```

Parameters **cena** – Nome do momento a ser configurado, tem que ser chave de **CONFIGURA**.

Returns Proxi da cena construída pela classe interna **PreviaDoMomento**.

class lab.views.jardim.**PreviaDoMomento**(*jogo*, *destino*)

Bases: object

Cria um proxi para a cena que ativa os portais somente quando é montrada pelo comando vai.

self.destino nome da cena que surge ao clicar no portal.

self._destino objeto cena referente à cena destino, recuperada como atributo da classe Cena.

Parameters

- **jogo** – Instância da classe JogoMarcela
- **destino** – nome da cena destino

vai(*_, **__)

Mostra a cena destino e configura o portal nela.

Parameters

- _ – captura lista de argumento para evitar erros
- __ – captura dicionário de argumento para evitar erros

Returns Nenhum

lab.views.jardim.**main**(*_)

Chamada do jogo, feita a partir do HTML.

Parameters _ – Parametros recebidos do HTML

Returns Instância do Jogo da Marcela

Note: Aventura no Jardim Botânico.

2.3 Primeiro Cenário do Jogo

Vamos começar importando o módulo vitollino para criar um jogo baseado na biblioteca Vitollino. Neste módulo vamos usar a classe Cena, que vai permitir a criação da primeira cena, o lago das tartarugas:


```
from _spy.vitollino import Cena

TARTARUGAS = "https://activufrj.nce.ufrj.br/studio/labase/lago.jpg?disp=inline&size=G"

def main():
    uma_cena = Cena(img=TARTARUGAS)
    uma_cena.vai()

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

Note: Ainda é um programa bem simples.

CHAPTER 3

Desenvolvimento

3.1 Vitollino - Introdução

3.2 Vitollino - Módulos

Vitollino é programado em Brython

Funcionalidades Documentadas:

3.2.1 Vitollino - Jogo de Novelas

Vitollino

Gerador de labirintos e jogos tipo *'novel'*.

Gerador de labirintos e jogos tipo *'novel'*.

Vitollino em Github

class _spy.vitollino.vitollino.**Bloco**

Bases: object

conta_pecas(*valor_peca*)

inicia_de_novo()

nao_monta()

vai()

class _spy.vitollino.vitollino.**Cena**(*img=''*, *esquerda=<CenaNula>*, *direita=<CenaNula>*, *meio=<CenaNula>*, *vai=None*, *nome=''*, *xy=(0, 0)*, *score={}*, ***kwargs*)

Bases: object

Use para construir uma cena.

```
from _spy.vitollino import Cena

cena_esq = Cena(img="esq.jpg")
cena_mei = Cena(img="mei.jpg", cena_esq)
cena_mei.vai()
```

Parameters

- **img** (*str*) – URL da imagem
- **esquerda** (Cena) – Cena que está à esquerda desta
- **direita** (Cena) – Cena que está à direita desta
- **meio** (Cena) – Cena que está à frente desta
- **vai** – Função a ser chamada no lugar da self.vai nativa

bota(*nome_item*)

static c(***cenas*)

portal(*esquerda=None*, *direita=None*, *meio=None*, ***kwargs*)

static q(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

static s(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

sai(*saida*)

score(***kwargs*)

tira(*item*)

vai(*ev=<NoEvent>*)

vai_direita(*_=0*)

vai_esquerda(*_=0*)

vai_meio(*_=0*)

class _spy.vitollino.vitollino.**Cursor**(*alvo*, *cena=<browser.BrythonMock object>*)

Bases: object

class _spy.vitollino.vitollino.**Dragger**(*dragger*)

Bases: object

ACTION = ''

POINTER = ''

drag_start(*ev*)

highten(*dy*)

mouse_down(*ev*)

static mouse_over(*ev*)

mouse_up(*_*)

no_mouse_move(*ev*)

pre_mouse_down(*ev*)

pre_mouse_move(*ev*)

static pre_mouse_over(*ev*)

pre_mouse_up(*_*)

widen(*dx*)

class _spy.vitollino.vitollino.**Droppable**(*droppable*, *dropper_name=''*, *action=None*, *cursor=None*)

Bases: object

drag_over(*ev*)

drop(*ev*)

class _spy.vitollino.vitollino.**Dropper**(*dropper*)

Bases: object

drag_start(*ev*)

mouse_over(*ev*)

class _spy.vitollino.vitollino.**Elemento**(*img=''*, *vai=None*, *style={}*, *tit=''*, *alt=''*, *cena=Inventario*, *score={}*, ***kwargs*)

Bases: object

Um objeto de interação que é representado por uma imagem em uma cena.

papel = Elemento(img="papel.png", tit="caderno de notas", vai=pega_papel, style=dict(left=350, top=550, width=60))

Parameters

- **img** – URL de uma imagem
- **vai** – função executada quando se clica no objeto
- **style** – dicionário com dimensões do objeto {"left": ..., "top": ..., width: ..., height: ...}
- **tit** – Texto que aparece quando se passa o mouse sobre o objeto
- **alt** – Texto para leitores de tela
- **cena** – cena alternativa onde o objeto vai ser colocado
- **score** – determina o score para este elemento
- **kwargs** – lista de parametros nome=URL que geram elementos com este nome e a dada imagem

classmethod c(***kwargs*)

entra(*cena*, *style={}*)

limbo = <browser.BrythonMock object>

score(***kwargs*)

class _spy.vitollino.vitollino.**Folha**(*texto*, *ht_ml*, *tela*, *left*)

Bases: object

drag_start(*ev*)

mouse_over(*ev*)

class _spy.vitollino.vitollino.**Inventario**(*tela=<browser.BrythonMock object>*)
Bases: object

Os objetos que estão de posse do jogador.

Parameters **tela** – Div do HTML onde o inventário será anexado

GID = '00000000000000000000'

bota(*nome_item*, *item=''*, *acao=None*)
Os objetos que estão de posse do jogador.

```
>>> inv.bota("uma_coisa")
>>> "uma_coisa" in inv.inventario
True
```

Parameters

- **nome_item** – uma string com o nome do item, ele será criado e colocado no inventário
- **item** – URL da imagem do item nomeado por nome_item
- **acao** – ação associada com o item nomeado quando ele é clicado

desmonta(*_=0*)

inicia()

monta(*_=0*)

mostra(*_=0*)

score(*casa*, *carta*, *move*, *ponto*, *valor*)

static send(*operation*, *data*, *action=<function Inventario.<lambda>>*, *method='POST'*)

tira(*nome_item*)

class _spy.vitollino.vitollino.**Jogo**
Bases: object

algo
Acessa a classe Elemento

cena
Acessa a classe Cena

nota
Acessa a classe Texto

quarto
Acessa a classe Sala

sala
Acessa a classe Salao

class _spy.vitollino.vitollino.**Labirinto**(*c=<CenaNula>*, *n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*)
Bases: object

static m(*cenas*)

class _spy.vitollino.vitollino.**Musica**(*sound*, *loop=True*, *autoplay=True*, *sound_type='audio/mpeg'*)

Bases: object

class _spy.vitollino.vitollino.**NoEv**

Bases: *_spy.vitollino.vitollino.NoEv*

Representa um evento vazio.

```
>>> print(ev.x, ev.y)
-100 -100
```

class _spy.vitollino.vitollino.**Point**(*x*, *y*)

Bases: list

px()

class _spy.vitollino.vitollino.**Popup**(*cena*, *tit=''*, *txt=''*, *vai=None*, ***kwargs*)

Bases: object

POP = <Popup>

static d(*cena*, *tit=''*, *txt=''*)

vai()

class _spy.vitollino.vitollino.**Portal**(*cena=None*, *debug_=False*, ***kwargs*)

Bases: object

L = {'min-height': '60%', 'width': '10%', 'cursor': 'e-resize', 'left': '90%', 'po

N = {'min-height': '20%', 'width': '60%', 'cursor': 'n-resize', 'left': '20%', 'po

O = {'min-height': '60%', 'width': '10%', 'cursor': 'w-resize', 'left': 0, 'positi

PORTAIS = {'S': {'min-height': '10%', 'width': '60%', 'bottom': 0, 'left': '20%',

S = {'min-height': '10%', 'width': '60%', 'bottom': 0, 'left': '20%', 'position':

Z = {'min-height': '10%', 'margin': '0%', 'width': '10%', 'cursor': 'zoom-in', 'po

p(***kwargs*)

vai(**_*)

class _spy.vitollino.vitollino.**Sala**(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

Bases: object

static c(***cenas*)

leste

norte

oeste

p()

sul

class _spy.vitollino.vitollino.**SalaCenaNula**

Bases: *_spy.vitollino.vitollino.SalaCenaNula*

Define uma Sala ou uma Cena vazia.

```
>>> cena = Cena(SalaCenaNula())  # A próxima cena
>>> uma_cena = Cena(SalaCenaNula(), cena)  # Cena nula à esquerda, proxima no meio
>>> uma_cena.vai_esquerda()  # tenta navegar para a cena à esquerda
>>> # não vai, pois a cena é nula e não deixa que se navegue para ela
>>> print(INVENTARIO.cena == cena)
True
```

Deve ser usado quando um parâmetro requer uma cena mas não deve ter uma cena válida ali.

class _spy.vitollino.vitollino.**Salao**(*n=<CenaNula>*, *l=<CenaNula>*, *s=<CenaNula>*, *o=<CenaNula>*, *nome=''*, ***kwargs*)

Bases: *_spy.vitollino.vitollino.Sala*

static c(***cenas*)

p()

class _spy.vitollino.vitollino.**Suporte**(*bloco*, *ht_ml*, *tela*, *left*, *certa*)

Bases: object

drag_over(*ev*)

drop(*ev*)

class _spy.vitollino.vitollino.**Texto**(*cena=<CenaNula>*, *tit=''*, *txt=''*, ***kwargs*)

Bases: *_spy.vitollino.vitollino.Popup*

esconde(*ev=<NoEvent>*)

mostra(*tit=''*, *txt=''*, ***kwargs*)

static texto(*tit=''*, *txt=''*, ***kwars*)

vai(*ev=<NoEvent>*)

_spy.vitollino.vitollino.**main**()

_spy.vitollino.vitollino.**parametrized**(*dec*)

_spy.vitollino.vitollino.**singleton**(*cls_to_decorate*)

Decora um classe para ser um singleton e retornar sempre a mesma instância.

```
>>> @singleton
... class Mono:
...     def __init__(self):
...             self.x = 0
...
>>> Mono().x, Mono().x = 1, 2
>>> print(Mono().x == Mono().x, Mono().x)
True 2
```

Parameters **cls_to_decorate** – A classe para ser definida como singleton

Returns O decorador de singleton

_spy.vitollino.vitollino.**wraps_class_to_mimic_wrapped**(*original_cls*)

Empacota uma classe decoradora para que apareça corretamente nos documentos.

```
>>> @wraps_class_to_mimic_wrapped
... class Exemplo:
...     ...
...
```

```
>>> print(Exemplo.__doc__)
Atualiza wrapper_cls para se assemelhar à classe original_cls.
```

Parameters **original_cls** – A Classe a ser empacotada

Returns O empacotador da classe

Note: Biblioteca Cliente para Jogos.

3.2.2 Flying Circus - Jogo Phaser

Braser

class _spy.circus.braser.**Braser**(*x=800*, *y=600*, *mode=None*, *name='pydiv'*, ***_*)
Bases: object

Brython object-oriented wrapper for js Phaser.

Parameters

- **x** – Canvas width.
- **y** – Canvas height.
- **mode** – Canvas mode.
- **name** – Game name.
- **keyargs** – Extra arguments

PHASER = <MagicMock name='mock.Phaser' id='140423796501304'>

create(*_)
Create element.

preload(*_)
Preload element.

subscribe(*subscriber*)
Subscribe elements for game loop.

Parameters **subscriber** –

update(*_)
Update element.

Note: Biblioteca Cliente para Jogos.

Game

class _spy.circus.game.**Actor**
Bases: *_spy.circus.game.Circus*

Define um ator, personagem ou cenário.

class _spy.circus.game.**Circus**

Bases: object

Interface com o engenho de games Phaser.

BRASER = None

create()

classmethod created()

enable(*item*)

group()

image(*name*, *img*)

Parameters

- **name** –
- **img** –

Returns

preload()

sprite(*name*, *x=0*, *y=0*)

Parameters

- **name** –
- **x** –
- **y** –

Returns

spritesheet(*name*, *img*, *x=0*, *y=0*, *s=1*)

Parameters

- **name** –
- **img** –
- **x** –
- **y** –
- **s** –

Returns

start_system()

tween(*sprite*, *time*, *tfunction='Linear'*, *autostart=True*, *delay=0*, *repeat=-1*, *yoyo=False*, ***kwd*)

Parameters

- **sprite** –
- **time** –
- **tfunction** –
- **autostart** –
- **delay** –

- **repeat** –
- **yoyo** –
- **kwd** –

Returns

update()

See also:

Module *_spy.circus.braser*

Note: Biblioteca Cliente para Jogos.

Masmorra

class _spy.circus.circus.**DesafioA**(*masmorra=[['LS', 'JN', 'KO'], ['IO', 'FN', 'IL'], ['GS', 'JS', 'GL']], off=0*)

Bases: object

ODD = False

create()

preload()

update()

class _spy.circus.circus.**Hero**(*gamer*)

Bases: object

create()

preload()

update()

class _spy.circus.circus.**Magic**(*masmorra*, *x*, *y*, *vx*, *vy*, *d*)

Bases: object

create()

kill()

preload()

update()

class _spy.circus.circus.**Masmorra**

Bases: object

create()

classmethod created()

posiciona_monstro(*m*, *x*, *y*)

preload()

update()

class _spy.circus.circus.**Monster**(*masmorra*)

Bases: object

create()

preload()

redirect(*play*, *dd*)

update()

_spy.circus.circus.**circus**(*desafio=1, param=[['LS', 'JN', 'KO'], ['IO', 'FN', 'IL'], ['GS', 'JS', 'GL']]*)

_spy.circus.circus.**desafio0**(*masmorra*)

_spy.circus.circus.**desafio3**(*mmap*)

_spy.circus.circus.**desafio4**(*mmap*)

_spy.circus.circus.**desafio5**(*mmap*)

_spy.circus.circus.**desafio6**(*mmap*)

_spy.circus.circus.**desafio7**(*mmap*)

_spy.circus.circus.**desafio8**(*mmap*)

spy.circus.circus.**main**(*=None*)

_spy.circus.circus.**posiciona_monstro**(*m*, *x*, *y*)

_spy.circus.circus.**random**() → x in the interval [0, 1).

See also:

Module *_spy.circus.game*

Note: Biblioteca Cliente para Jogos.

CHAPTER 4

Lançamentos

4.1 Notas de Lançamento V. 1.1.0

Vitollino

4.1.1 Milestone

Gatíneo - Ativar modo multiusuário

- [ ] *Aspecto #1*: Console do mentor
- [ ] *Aspecto #2*: Equipes de jogadores
- [ ] *Aspecto #3*: Import reload
- [ ] *Aspecto #4*: Gerência de mentoria
- [ ] *Aspecto #5*: Módulo de supervisão de mentoria
- [ ] *Aspecto #6*: Modelagem gráfica colaborativa

4.1.2 Aspectos do Lançamento

Destaques dos Aspectos

Início da documentação do tutorial

Aspecto #1

Console do mentor Este console permite que o mentor do tutorial receba notificações de dificuldades que os estudantes estão tendo com seus projetos.

Aspecto #2

Equipes de jogadores Permite que os jogadores formem equipes e resolvam conjuntamente uma aventura.

Aspecto #3

Import reload Esta função permite que o cache de módulos seja invalidada, permitindo que se use o código atualizado produzido por outro estudante deste projeto.

Aspecto #4

Gerência de mentoria Facilita que diversos mentores se dividam no atendimento dos problemas.

Aspecto #5

Módulo de supervisão de mentoria Cria uma hierarquia entre os mentores para que exista um mentor supervisor, para quem problemasmais complexos possam ser encaminhados.

Aspecto #6

Modelagem gráfica colaborativa Uma ferramenta de modelagem rudimentar que permite a construção de um modelo de solução colaborativamente.

4.1.3 Melhoramentos

Facilidades para a formação de equipes.

Melhoramento #1

Menu inicial para a formação de equipes Este menu permite que os jogaores vão se agregando a times já formados ou iniciem um novo time.

4.1.4 Consertos

Nenhum conserto notável.

4.1.5 Questões e Problemas Conhecidos

A funcionalidade ainda é muito simples, requer melhorias.

4.1.6 Lançamentos Anteriores e Posteriores

Lançamento Anterior: Milestone Catioro *Lançamento 1.0.0*

Próximo Lançamento: Milestone Ratíneo *Lançamento 1.2.0*

4.2 Notas de Lançamento V. 1.0.0

Vitollino

4.2.1 Milestone

Catioro - Montagem do Ambiente de Jogo

- [ ] *Aspecto #1*: Navegação através de portais
- [ ] *Aspecto #2*: Construção de mapas de labirinto
- [ ] *Aspecto #3*: Adiciona música e sons
- [ ] *Aspecto #4*: Texto decorador
- [ ] *Aspecto #5*: Montagem interativa com cursor móvel
- [ ] *Aspecto #6*: Banco de dado Redis via Walrus

4.2.2 Aspectos do Lançamento

Destaques dos Aspectos

Início da documentação do tutorial

Aspecto #1

Navegação através de portais A navegação entre cenários se dá através de cursores indicativos que flutuam sobre o local do portal.

Aspecto #2

Construção de mapas de labirinto Um labirinto pode ser construído montando uma matriz de duas dimensões. Cada célula da matriz irá se conectar coms seus vizinhos nos quatro pontos cardeais.

Aspecto #3

Adiciona música e sons Uma música de fundo pode ser adicionada ao jogo.

Aspecto #4

Texto decorador Um portal pode ser decorado com um popup de texto que aparece antes de transitar para outra cena.

Aspecto #5

Montagem interativa com cursor móvel Para facilitar a montagem de cenários pode-se habilitar um cursor interativo que irá fornecer as coordenadas e o tamanho do hot spot que irá ativar a ação quando clicado.

Aspecto #6

Banco de dado Redis via Walrus Um banco Redis foi adicionadoao servidor. Ele é interfaceado pelo adaptador python definido pelo pacote Walrus.

4.2.3 Melhoramentos

Reúne Vitollino e Braser no mesmo repositório.

Melhoramento #1

Tutorial vitollino sendo portado para Restrutured Text.

4.2.4 Consertos

Nenhum conserto notável.

4.2.5 Questões e Problemas Conhecidos

A funcionalidade ainda é muito simples, requer melhorias.

Uma nova versão deve integrar melhor as duas modalidades de tutorial.

4.2.6 Lançamentos Anteriores e Posteriores

Próximo Lançamento: A ser definido *Lançamento 1.1.0*

4.3 Notas de Lançamento Futuro

Vitollino

4.3.1 Milestones

Catioro - Criar e personalizar equipes

Gatíneo - Ativar modo multiusuário

Ratíneo - Ativar modo cooperativo

4.3.2 Aspectos dos Lançamentos

Destaques dos Aspectos

Início da documentação do tutorial

Milestone Catioro

- [ ] Navegação através de portais
- [ ] Construção de mapas de labirinto
- [ ] Adiciona música e sons
- [ ] Texto decorador
- [ ] Montagem interativa com cursor móvel
- [ ] Banco de dado Redis via Walrus

Milestone Gatíneo

- [ ] Console do mentor
- [ ] Equipes de jogadores
- [ ] Import reload
- [ ] Gerência de mentoria
- [ ] Módulo de supervisão de mentoria
- [ ] Modelagem gráfica colaborativa

Milestone Ratíneo

4.3.3 Lançamentos Anteriores

Milestone Catioro *Lançamento 1.0.0*

Milestone Gatineo *Lançamento 1.1.0*

CHAPTER 5

Indices e Tabelas

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z