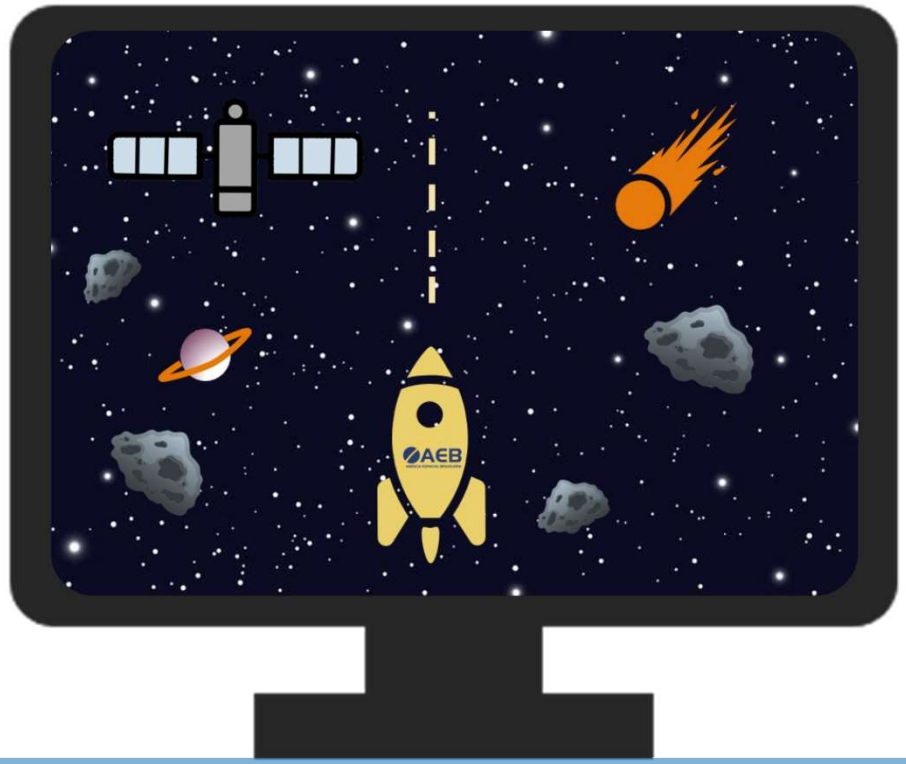


Minicurso de Programação de Jogos em Python

Aula 4
Criação dos Player, Hazard e outras movimentações

Acesse

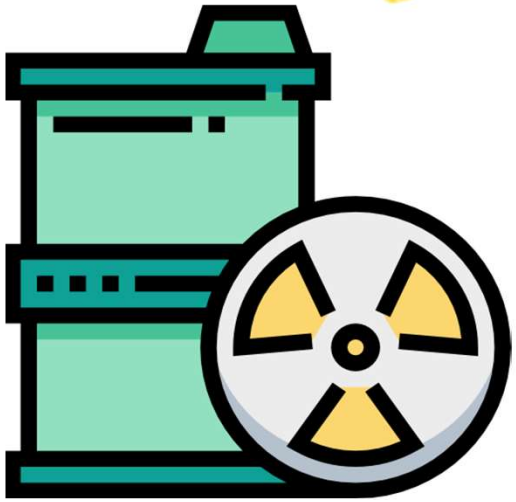
aebescolavirtual.aeb.gov.br/

Professores: Frederick Campbell - Aluno de Graduação do ITA
Isabela Vieira - Aluna de Pós-Graduação do ITA

PROGRAMA ESPACIAL BRASILEIRO
AEB escola
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Tópicos

- Objetos do jogo.
- Criação do player.
- Movimentação do player.
- Restrição da movimentação do player.
- Criação e movimentação de ameaças (hazards) para o jogador.



Recapitulando ...

- Vamos lembrar o que foi feito até o momento no jogo **Fuga Espacial**

  - Criação do objeto game, que é uma instância da classe **Game**
  - Criação do objeto background, que é uma instância da classe **Background**
  - O desenho do background (ou plano de fundo) e das margens esquerda e direita, que formam o cenário do jogo (“corredor entre asteroides”)
  - Movimentação do background

Recapitulando ...

Desenho do background e das margens esquerda e direita, que formam o cenário do jogo

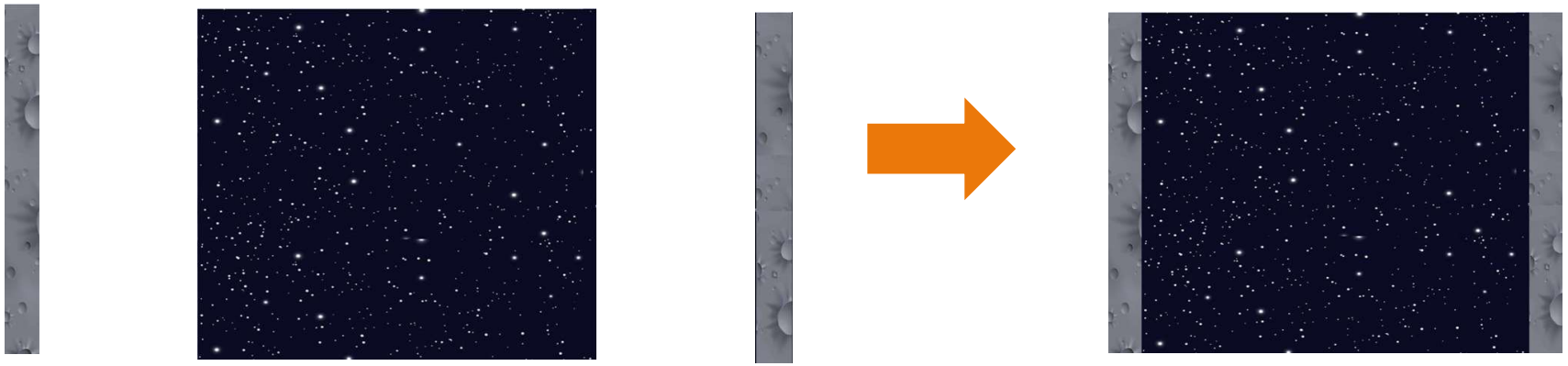
- Os objetos como background, fontes, personagens e praticamente tudo em um jogo em Python são desenhados na tela através do método blit (**b**it **b**lock **t**ransfer): screen.blit(background, (0, 0))
  - "Bliting": copiar dados de imagem de uma superfície para outra
  - O blit desenha os objetos, mas estes só serão exibidos após a execução método pygame.display.update()

(Kinsley & McGugan, 2015)

Recapitulando ...

Desenho do background e das margens esquerda e direita, que formam o cenário do jogo

- No jogo Fuga Espacial, o background é formado por 3 imagens:



Margem esquerda

Imagem de fundo

Margem direita

Cenário do jogo

Recapitulando ...

Movimentação do background

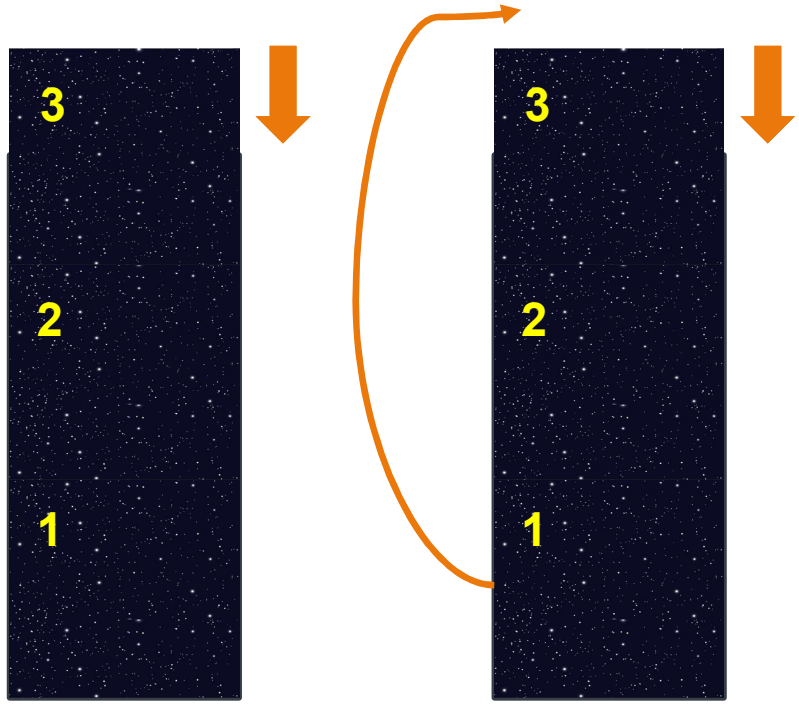
- Em um jogo digital, a ideia é criar a ilusão do movimento
  - Então o background não se movimenta de fato
- No Python, pode-se utilizar o descolamento de tiles para a movimentação de objetos, como o background
- O deslocamento pode ser horizontal, vertical ou diagonal, dependendo do jogo

(Kinsley & McGugan, 2015)

Recapitulando ...

Movimentação do background

- No jogo Fuga Espacial, os tiles são do background
- Eles criam a ilusão de movimento vertical, de cima para baixo



Recapitulando ...

- O jogo Fuga Espacial possui, no momento, a seguinte estrutura de blocos de código:

```
"""
Jogo: Fuga Espacial
Descrição: Um grupo de di
    A nave precisa se desv
"""

import pygame

class Background:...
# Background

class Game:...
# Game

# Cria o objeto game e chama o loop básico
game = Game("resolution", "fullscreen")
game.loop()
```

Cada complemento do jogo promove atualizações em um ou mais “blocos de código”

Importação de módulos

Classes

Início do Programa

Objetos do jogo

PROGRAMA ESPACIAL BRASILEIRO

AEB escola

AEB AGÊNCIA ESPACIAL BRASILEIRA

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Objetos do jogo

- Além do cenário, outros itens são indispensáveis para o jogo.
- Eles representam o jogador ou jogadores e os desafios que deverão ser superados.
  - A forma do elemento dependerá do enredo do jogo.
    - Alguns jogos usam figuras geométricas como elementos
    - Outros fazem uso de sprites, que são objetos animados, por exemplo.
  - O PyGame oferece recursos para a criação e manipulação desses objetos, como a classe Sprite.
    - Para utilizar esses recursos, basta que o programador importe o módulo do pygame.

Objetos do jogo

- O Pygame oferece métodos para o desenho e manuseio de elementos como Figuras Geométricas:
  - O método pygame.draw.rect() a criação de um retângulo


```
rect_1 = pygame.draw.rect(screen, (255, 0, 0), (x_1, y_1, rect_width, rect_height))  # vermelho
rect_2 = pygame.draw.rect(screen, (0, 255, 0), (x_2, y_2, rect_width, rect_height))  # verde
```

  - O método pygame.draw.circle() a criação de um círculo
  - O método pygame.draw.polygon() a criação de um polígono

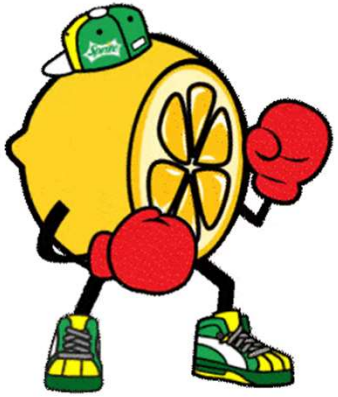
Objetos do jogo

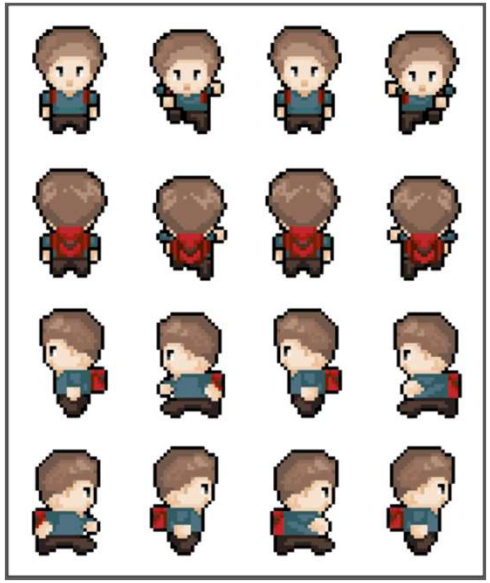
Tenor

- Outro elemento bastante utilizado é o sprite
  - Eles podem ser animados e se movimentar no jogo
  - A ideia é agrupar várias imagens que indicam sequência do objeto para dar a ilusão de movimento do mesmo.

Objetos do jogo

- A classe Sprite fornece métodos ou funções para tartar essas imagens.
- O programador pode criar as imagens de sprites
- Exemplos





Tenor

Criação do player

Vamos agora desenhar objetos fundamentais para a dinâmica do jogo Fuga Espacial, começando pelo Player

Criação do player

PROGRAMA ESPACIAL BRASILEIRO

AEB escola

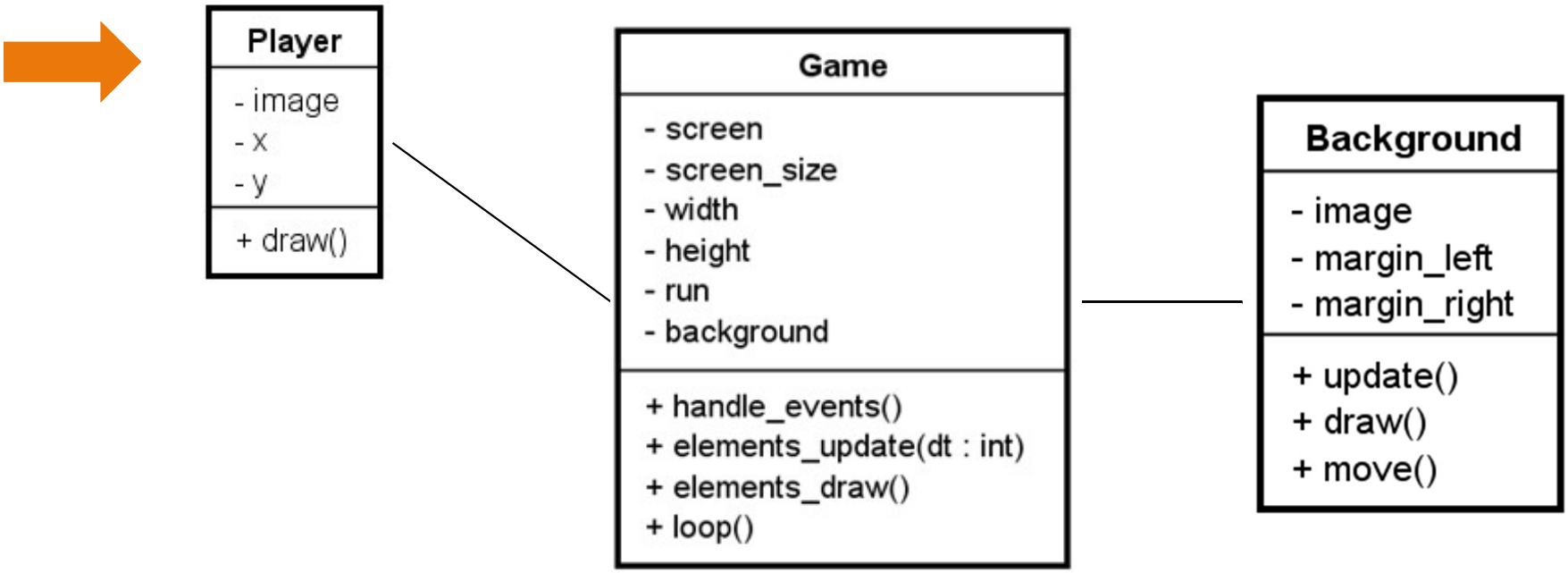
AEB AGÊNCIA ESPACIAL BRASILEIRA

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Atualizando o Diagrama de Classes

- Para começar, vamos inserir a classe **Player** no Diagrama de Classes do jogo:



Criação do player

tenor.com

- De forma geral, um programa de computador passa por várias atualizações durante seu desenvolvimento
- Cada item adicionado pode requerer mudanças em várias linhas do código fonte, seja para inclusão, remoção ou alteração de comandos

Criação do player

- Para a inclusão do player no jogo, blocos do código serão alterados

```
"""
Jogo: Fuga Espacial
Descrição: Um grupo de diplomatas escapam de uma fortaleza estelar a bordo de uma nave danificada.
   A nave precisa se desviar das ameaças e sobreviver até atingir a zona de segurança diplomática.
"""

import pygame

class Background:...
# Background

class Game:...
# Game

# Cria o objeto game e chama o loop básico
game = Game("resolution", "fullscreen")
game.loop()
```

Importação de módulos

Classes

Início do Programa

Criação do player

- Para a inclusão do player no jogo, blocos do código serão alterados

```
"""
Jogo: Fuga Espacial
Descrição: Um grupo de diplomatas escapam de uma fortaleza estelar a bordo de uma nave danificada.
                                    viver até atingir a zona de segurança diplomática.
```

- Por exemplo,
  - A classe Player será criada
  - A classe Game será atualizada na parte dos atributos e na função loop

Vamos ver como isso ocorre …

Importação de módulos

Classes

Início do Programa

```
game.loop()
```

Lembretes da POO

- Para definir uma classe, você usa a palavra-chave class, seguida do nome da classe e dos dois pontos (:).
- Construtor é um método especial de classe, que é sempre denominado __init__ .
- O primeiro parâmetro do construtor é sempre self (uma palavra-chave refere-se à própria classe).
- O construtor é chamado sempre que se cria um objeto da classe.
- Atribui valores do parâmetro às propriedades do objeto que será criado.

Criação do player

- Para definir uma classe, você usa a palavra-chave class , seguida do nome da classe e dos dois pontos (:).

Criação do player

- Criação da classe Player.

```
57  class Player:
58      """
59      Esta classe define Jogador
60      """
```

 # criando a classe

Criação do player

- Criação da classe Player.

```
57  class Player:
58      """
59      Esta classe define Jogador
60      """
61      image = None
62      x = None
63      y = None
```

 # criando a classe

 # inicializar atributos

Criação do player

- Para definir uma classe, você usa a palavra-chave class , seguida do nome da classe e dos dois pontos (:).

- Construtor é um método especial de classe, que é sempre denominado __init__ .

Criação do player

- Criação da classe Player.

```
57  class Player:
58      """
59      Esta classe define Jogador
60      """
61      image = None
62      x = None
63      y = None
64
65      def __init__(self, x, y):
66          player_fig = pygame.image.load("Images/player.png")
67          player_fig.convert()
68          player_fig = pygame.transform.scale(player_fig, (90, 90))
69          self.image = player_fig
70          self.x = x
71          self.y = y
72
73      # __init__()
```

 # criando a classe

 # inicializar atributos

 # função __init__()

Criação do player

- Para definir uma classe, você usa a palavra-chave class , seguida do nome da classe e dos dois pontos (:).
- Construtor é um método especial de classe, que é sempre denominado __init__ .
- O primeiro parâmetro do construtor é sempre self (uma palavra-chave refere-se à própria classe).
- O construtor é chamado sempre que se cria um objeto da classe.
- Atribui valores do parâmetro às propriedades do objeto que será criado.

```
nave = Player(1, 4) # exemplo de como criar um objeto da classe Player
```

Criação do player

- Criação da classe Player.

```
57  class Player:
58      """
59      Esta classe define Jogador
60      """
61      image = None
62      x = None
63      y = None
64
65      def __init__(self, x, y):
66          player_fig = pygame.image.load("Images/player.png")
67          player_fig.convert()
68          player_fig = pygame.transform.scale(player_fig, (90, 90))
69          self.image = player_fig
70          self.x = x
71          self.y = y
72
73      # __init__()
75      # Desenhar Player
76      def draw (self, screen, x, y):
77          screen.blit(self.image, (x, y))
78
79  # Player
```

 # criando a classe

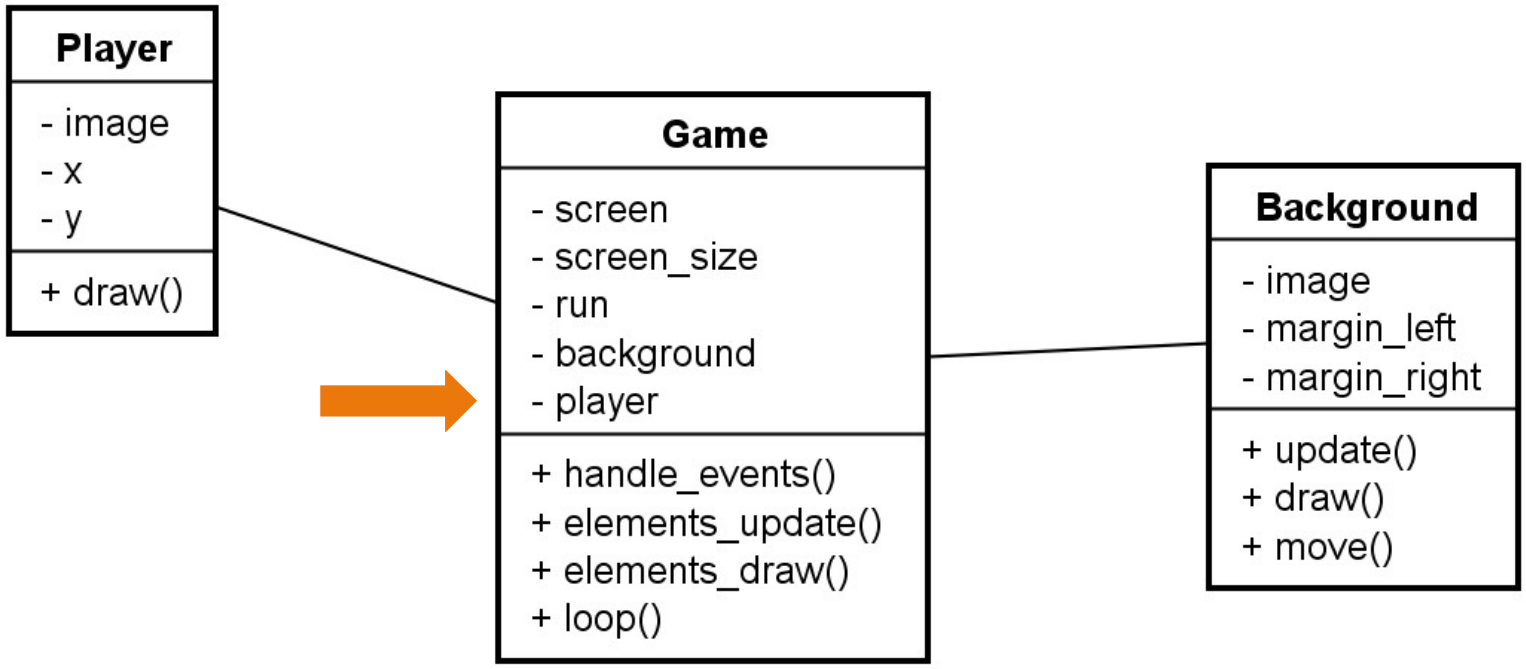
 # inicializar atributos

 # função __init__()

 # método para desenhar o player

Atualizando o Diagrama de Classes

- Vamos inserir o atributo player na classe Game :



Criação do player

- Vamos inserir o atributo player na classe Game :

```
81  class Game:
82      screen = None
83      screen_size = None
84      width = 800
85      height = 600
86      run = True
87      background = None
88      player = None
89
90      def __init__(self, size, fullscreen):
```

 # atributo player

Criação do player

- Criação do Player na classe Game

```
137         # Criar o Plano de fundo
138         self.background = Background()
139
140         # Posicao do Player
141         x = (self.width - 56) / 2
142         y = self.height - 125
143
144         # Criar o Player
145         self.player = Player(x, y)
```

 # posição do player

 # criar o player

Criação do player

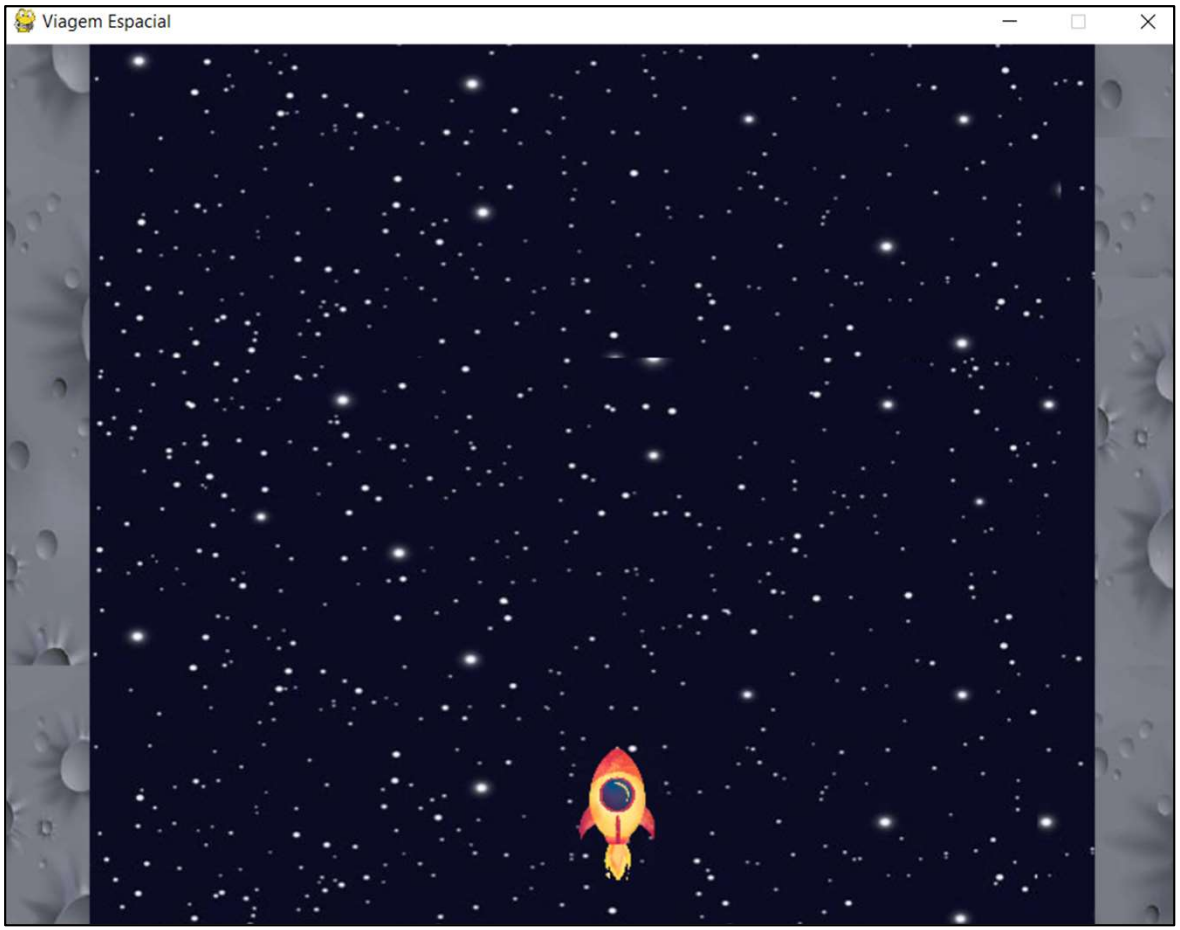
- Chamada da função player.draw() no laço principal da classe Game.

```
165         # adiciona movimento ao background
166         self.background.move(self.screen, self.height, movL_x, movL_y, movR_x, movR_y)
167         movL_y = movL_y + velocidade_background
168         movR_y = movR_y + velocidade_background
169
170         #se a imagem ultrapassar a extremidade da tela, move de volta
171         if movL_y > 600 and movR_y > 600:
172             movL_y -= 600
173             movR_y -= 600
174
175         # Desenhar Player
176         self.player.draw(self.screen, x, y)
177
178         # Atualiza a tela
179         pygame.display.update()
```

 # desenhar o player por meio do método draw

Criação do player

- Criação do player.



Tela do jogo com o player

Movimentação do player

- Na classe Game, serão adicionados atributos

```
81  class Game:
82      screen = None
83      screen_size = None
84      width = 800
85      height = 600
86      run = True
87      background = None
88      player = None
89
90      # movimento do Player
91      DIREITA = pygame.K_RIGHT
92      ESQUERDA = pygame.K_LEFT
93      mudar_x = 0.0
94
95      def __init__(self, size, fullscreen):
```

 # movimento do player

Movimentação do Player

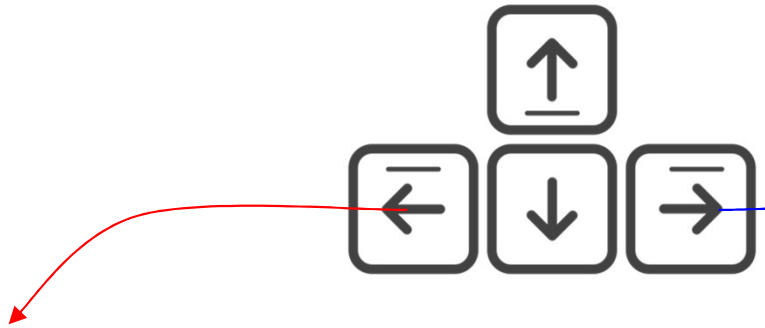
LEMBRE-SE!

A função **handle_events()** é responsável por lidar com os eventos que ocorrem durante o jogo, como o pressionamento de teclas ou cliques do mouse.

Movimentação do Player

O que queremos?

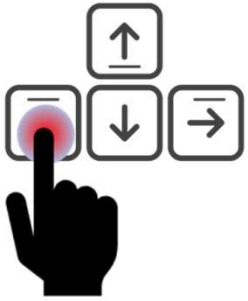
- o Player se movimente ao **pressionarmos** as teclas



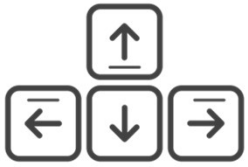
para a esquerda **para a direita**

- o Player **não se move quando a tecla é liberada**

Movimentação do Player



- Se o evento atual for uma tecla **pressionada (KEYDOWN)**:
  - a função verifica se a tecla pressionada é ESQUERDA ou DIREITA
    - Se for **ESQUERDA**, self.mudar_x é configurado para **-3**, movendo o Player para a esquerda.
    - Se a tecla for **DIREITA**, self.mudar_x é configurado para **3**, movendo o Player para a direita.



- Se o evento atual for uma tecla **liberada (KEYUP)**, ESQUERDA ou DIREITA
  - Pára o movimento do Player..

Movimentação do player

- Iremos complementar a função **handle_events(),** dentro da classe **Game**.

```
110     def handle_events(self):
111         """
112         Trata o evento e toma a ação necessária.
113         """
114         for event in pygame.event.get():
115 
116             if event.type == pygame.QUIT:
117                 self.run = False
118 
119             # se clicar em qualquer tecla, entra no if
120             if event.type == pygame.KEYDOWN:
121 
122                 # se clicar na seta da esquerda, anda 3 para a esquerda no eixo x
123                 if event.key == self.ESQUERDA:
124                     self.mudar_x = -3
125                 # se clicar na seta da direita, anda 3 para a direita no eixo x
126                 if event.key == self.DIREITA:
127                     self.mudar_x = 3
128 
129             # se soltar qualquer tecla, não faz nada
130             if event.type == pygame.KEYUP:
131                 if event.key == self.ESQUERDA or event.key == self.DIREITA:
132                     self.mudar_x = 0
133 
134     # handle_events()
```

 # para de executar o jogo se clicarmos no x da janela

 # se uma tecla é pressionada

 # se uma tecla é liberada

Movimentação do player

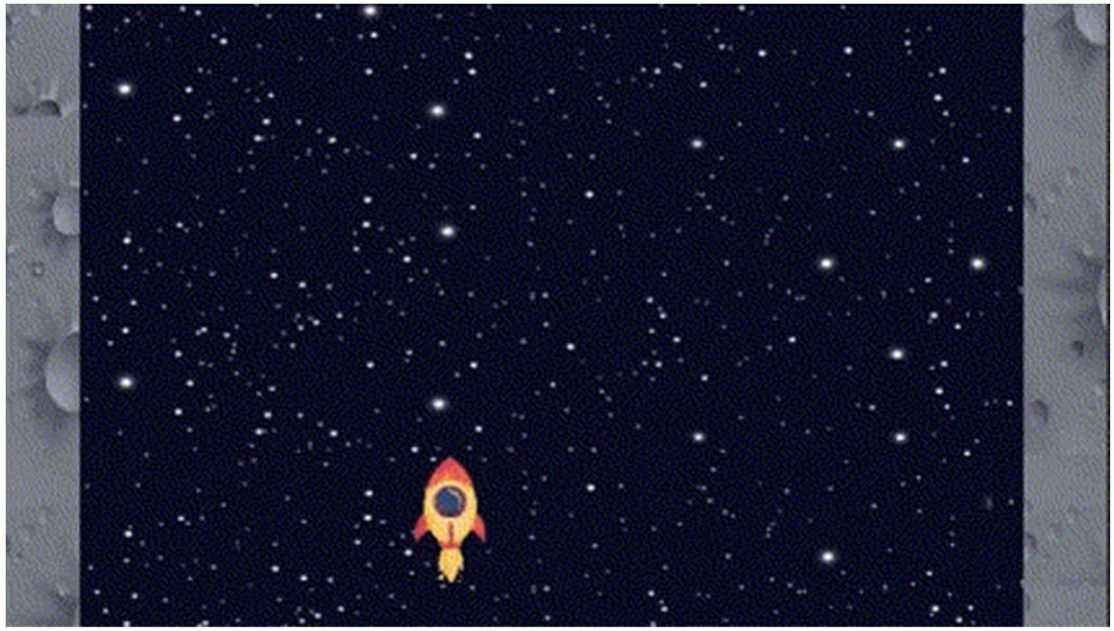
- Ainda na classe **Game**, vamos inserir o código seguinte no **laço principal** dentro da função loop()

```
192         #se a imagem ultrapassar a extremidade da tela, move de volta
193         if movL_y > 600 and movR_y > 600:
194             movL_y -= 600
195             movR_y -= 600
196
197         # Movimentação do Player
198         # Altera a coordenada x da nave de acordo comas mudanças no event_handle() para ela se mover
199         x = x + self.mudar_x
200
201         # Desenha o Player
202         self.player.draw(self.screen, x, y)
203
204         # Atualiza a tela
205         pygame.display.update()
```

 # mudar a posição do player

Movimentação do player

- Movimentação do player



Tela do jogo com o player podendo movimentar-se

Movimentação do player

- Restrição da movimentação do player
  - Na classe Game, serão criados novos atributos

```
82  class Game:
83      screen = None
84      screen_size = None
85      width = 800
86      height = 600
87      run = True
88      background = None
89      player = None
90      render_text_bateulateral = None
91      render_text_perdeu = None
92
93      # movimento do Player
94      DIREITA = pygame.K_RIGHT
95      ESQUERDA = pygame.K_LEFT
96      mudar_x = 0.0
97
98      def __init__(self, size, fullscreen):
```

 # atribuitos adicionais

Movimentação do player

Restrição da movimentação do player (nave)

- O que queremos?
  - Se a nave bater na lateral o jogo pára e uma mensagem aparece na tela

Para isso precisamos:

- definir a fonte da letra e a mensagem que queremos escrever
- quanto tempo a mensagem fica aparecendo na tela e o que acontece com o jogo
- criar a condição que nos diz que a nave bateu na lateral

Movimentação do player

- Restrição da movimentação do player
  - Alteração na função __init__() da classe Game

```
112         # define as fontes
113         my_font = pygame.font.Font("Fonts/Fonte4.ttf", 100)
114
115         # Mensagens para o jogador
116         self.render_text_bateulateral = my_font.render("VOCê BATEU!", 0,(255, 255, 255))  # ("texto",
117         self.render_text_perdeu = my_font.render("GAME OVER!", 0, (255, 0, 0))  # ("texto, opaco/trans
118     # init()
119
120     def handle_events(self):
```

 # definir as fontes

 # mensagens na tela do jogo

Movimentação do player

- Restrição da movimentação do player
  - Incluir o módulo time

```
7    import pygame
8    import time # uso da função-membro time.sleep(...) in loop
```

 # módulo time

Movimentação do player

- Restrição da movimentação do player
  - O código a seguir será inserido no laço principal, na classe Game.

```
210             # Desenha o Player
211             self.player.draw(self.screen, x, y)
212
213             # Restrições do movimento do Player
214             # Se o Player bate na lateral não é Game Over
215             if x > 760 - 92 or x < 40 + 5:
216                 self.screen.blit(self.render_text_bateulateral, (80, 200))
217                 pygame.display.update()  # atualizar a tela
218                 time.sleep(3)
219                 self.loop()
220                 self.run = False
221
222             # Atualiza a tela
223             pygame.display.update()
```

 # restrição do movimento

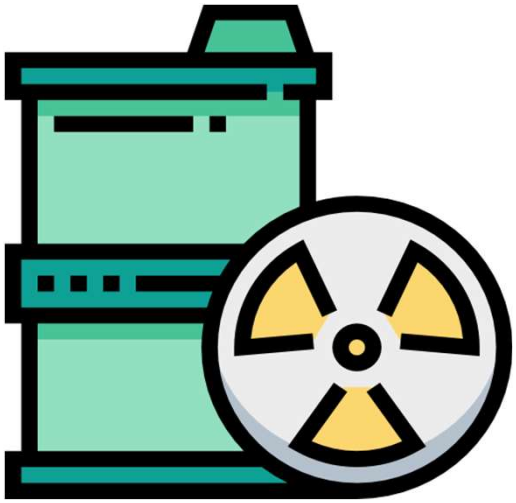
Movimentação do player

- Restrição da movimentação do player

Tela do jogo com o player

colidindo com o limite das bordas

Hazard (Ameaça)



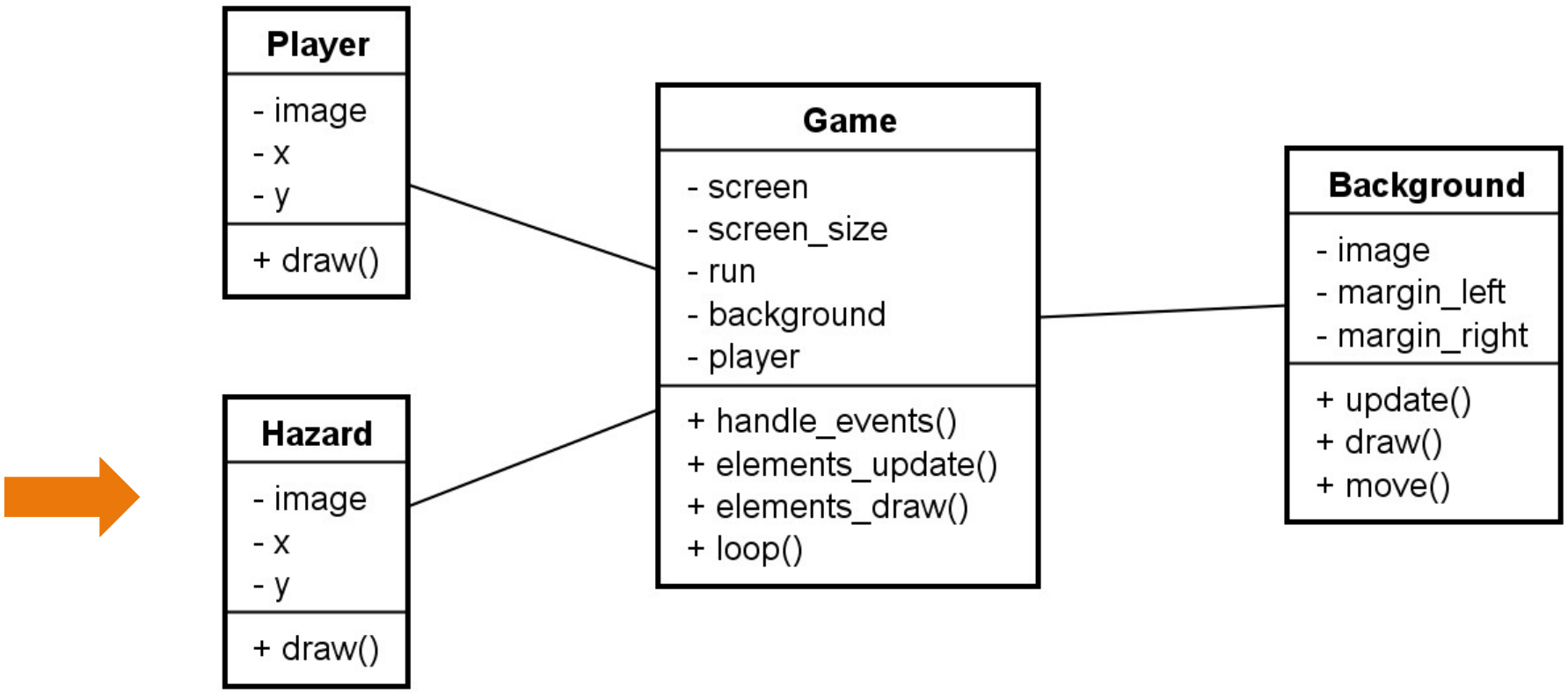
Outro objeto essencial para a dinâmica do jogo Fuga Espacial é o Hazard

Hazard e outras movimentações

- **Hazard** refere-se aos perigos do jogo, ou seja, às dificuldades que aparecerão, assim como suas configurações, que irão tornar o jogo mais desafiador.

Atualizando o Diagrama de Classes

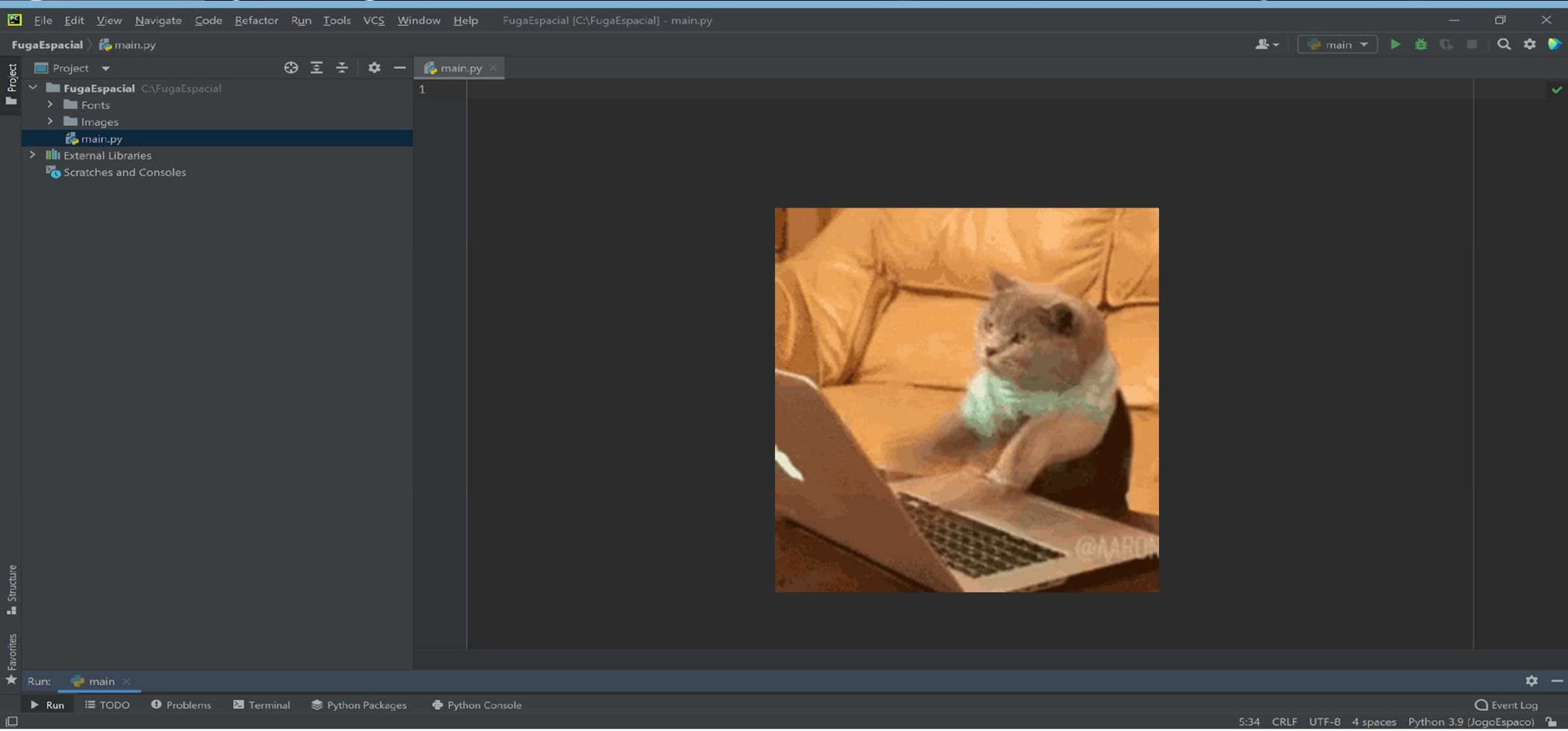
- Vamos agora inserir a classe **Hazard** no Diagrama de Classes do jogo:



Codificação dos Hazards

- De forma geral, um programa de computador passa por várias atualizações durante seu desenvolvimento.
- Cada item adicionado pode requerer mudanças em várias linhas do código fonte, seja para inclusão, remoção ou alteração de comandos.

Vamos ao código!



Uma vez atualizado o Diagrama de Classes, vamos implementar a classe Hazard

tenor.com

- Primeiramente, vamos importar uma nova biblioteca, a biblioteca **random**, que insere elementos aleatórios quando necessário no código, escrevendo **import random** na declaração de bibliotecas.

```
7   import pygame
8   import time # uso da função-membro time.sleep(...) in loop
9   import random # importar o random                    ange (...) em loop
```

Criação de Hazard

- O código abaixo mostra a classe **Hazard** com seus atributos e __init__().

```
82  class Hazard:
83      """
84      Esta classe define Ameaça ao Jogador
85      """
86      image = None
87      x = None
88      y = None
89
90      def __init__(self, img, x, y):
91          hazard_fig = pygame.image.load(img)
92          hazard_fig.convert()
93          hazard_fig = pygame.transform.scale(hazard_fig, (130, 130))
94          self.image = hazard_fig
95          self.x = x
96          self.y = y
97      # __init__()
```

 # inicializar atributos

 # configurações

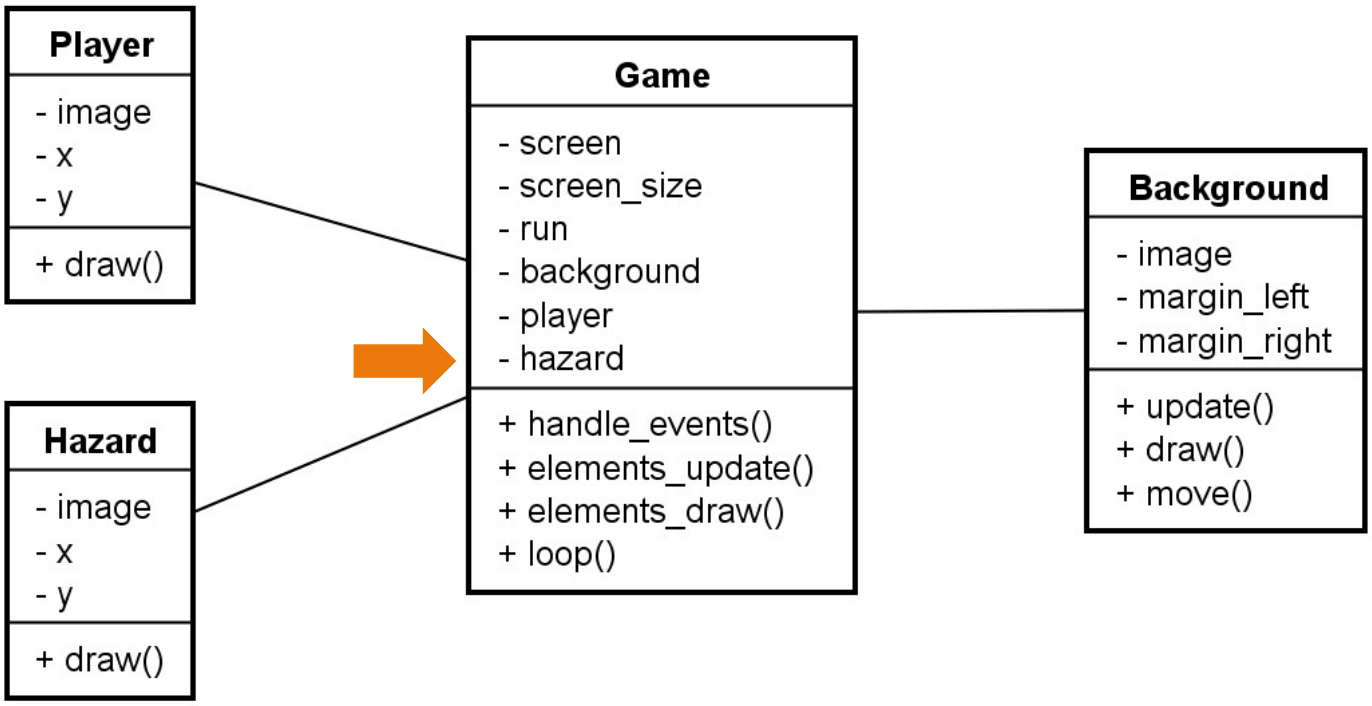
Criação de Hazard

- Agora, vamos concluir a criação da classe **Hazard** inserindo, logo após o __init__(), a função draw(), que é responsável por colocar os hazards na tela do jogo.

```
99      # Desenhar Hazard
100     def draw (self, screen, x, y):
101         screen.blit(self.image, (x, y))
102     #draw()
103 # Hazard:
```

Atualizando o Diagrama de Classes

- Vamos inserir o atributo hazard na classe **Game**:



Criação de Hazard

- ATENÇÃO: observe que na classe **Game**, a inserção da variável “hazard” é feita, inicializada por “hazard = []”.

```
105  class Game:
106      screen = None
107      screen_size = None
108      width = 800
109      height = 600
110      run = True
111      background = None
112      player = None
113      hazard = []          # novo atributo
114      render_text_bateulateral = None
115      render_text_perdeu = None
```

Criação de Hazard

- Observe que é necessário definir a velocidade do hazard, conforme a seguinte instrução, que será colocada na função loop():

```
178     def loop(self):
179         """
180         Esta função contém o laço principal
181         """
182         # variáveis para movimento de Plano de Fundo/Background
183         velocidade_background = 10
184         velocidade_hazard = 10
```

 # atribuição das velocidades

Criação de Hazard

- Ainda na função loop() da classe **Game**, criamos as variáveis de localização dos hazards, aleatoriamente no eixo x e constante no eixo y, conforme observa-se pela figura. Além disso, adiciona-se às variáveis de comprimento e largura.

```
186         hzrd = 0
187         h_x = random.randrange(125, 660)
188         h_y = -500
189
190         # Info Hazard
191         h_width = 100
192         h_height = 110
```

 # localização

 # dimensões

Criação de Hazard

- **Agora, criaremos os hazards, seguindo as seguintes instruções:**

- "self.hazard" se refere a uma lista de hazards que é um atributo da classe Game.

- ".append()" é um método que adiciona um novo elemento à lista.

- "("Images/satelite.png", h_x, h_y)" são argumentos passados para o construtor da classe **Hazard** para a instanciação dos 5 objetos hazard do jogo. Os argumentos são o arquivo de imagem e as coordenadas do objeto.

```
212         # Criar os Hazards
213         self.hazard.append(Hazard("Images/satelite.png", h_x, h_y))
214         self.hazard.append(Hazard("Images/nave.png", h_x, h_y))
215         self.hazard.append(Hazard("Images/cometaVermelho.png", h_x, h_y))
216         self.hazard.append(Hazard("Images/meteoros.png", h_x, h_y))
217         self.hazard.append(Hazard("Images/buracoNegro.png", h_x, h_y))
```

Criação de Hazard

 # adicionar elemento à lista

```
212         # Criar os Hazards
213         self.hazard.append(Hazard("Images/satelite.png", h_x, h_y))
214         self.hazard.append(Hazard("Images/nave.png", h_x, h_y))
215         self.hazard.append(Hazard("Images/cometaVermelho.png", h_x, h_y))
216         self.hazard.append(Hazard("Images/meteoros.png", h_x, h_y))
217         self.hazard.append(Hazard("Images/buracoNegro.png", h_x, h_y))
```

 # atributo de classe

 # argumentos

KD0

Hazard e outras movimentações

- A partir de agora, definimos os movimentos dos hazards, em relação à posição e velocidade, aleatórios e não; além disso, definimos onde o hazard vai aparecer, recomeçando a posição do obstáculo e da faixa.

```
263         # adicionando movimento ao hazard
264         h_y = h_y + velocidade_hazard / 4
265         self.hazard[hzrd].draw(self.screen, h_x, h_y)
266         h_y = h_y + velocidade_hazard
267
268         # definindo onde hazard vai aparecer, recomeçando a posição do obstaculo e da faixa
269         if h_y > self.height:
270             h_y = 0 - h_height
271             h_x = random.randrange(125, 650 - h_height)
272             hzrd = random.randint(0, 4)
273
274         # Atualiza a tela
275         pygame.display.update()
```

Slide 60

KD0 Esse slide foi descolado para cá

Karla DF; 2023-06-20T12:52:09.924

Hazard e outras movimentações

Aqui está uma explicação linha por linha:

- "h_y = h_y + velocidade hazard / 4" atualiza a posição vertical do hazard, adicionando uma fração da velocidade do hazard a cada quadro. Isso fará com que o hazard se mova para baixo na tela.

- "self.hazard[hzrd].draw(hzrd, h_x, h_y)" chama um método "draw()" da classe Harzard para desenhar o hazard em nova posição na tela.

```
                                                    # atualização de posição e velocidade
263             # adicionando movimento ao hazard
264             h_y = h_y + velocidade_hazard / 4
265             self.hazard[hzrd].draw(self.screen, h_x, h_y)
                                                    # método para desenhar o hazard
```

Hazard e outras movimentações

- Aqui está uma explicação linha por linha:

- Isso é necessário para garantir que o hazard continue se movendo em uma velocidade constante a cada quadro.

```
265             self.hazard[hzrd].draw(self.screen, h_x, h_y)
266             h_y = h_y + velocidade_hazard
267                                 # atualização de posição e velocidade
```

Hazard e outras movimentações

- Aqui está uma explicação linha por linha:

- "if h_y > self.height:" verifica se o hazard passou da altura da tela.

- "h_y = 0 – h_height" redefine a posição vertical do hazard para o topo da tela (0) menos a altura do próprio hazard.

```
                # condição de contorno
268             # definindo onde hazard vai aparecer, recomeçando a posição do obstaculo e da faixa
269             if h_y > self.height:
270                 h_y = 0 - h_height
                                        # atualização de posição
```

Hazard e outras movimentações

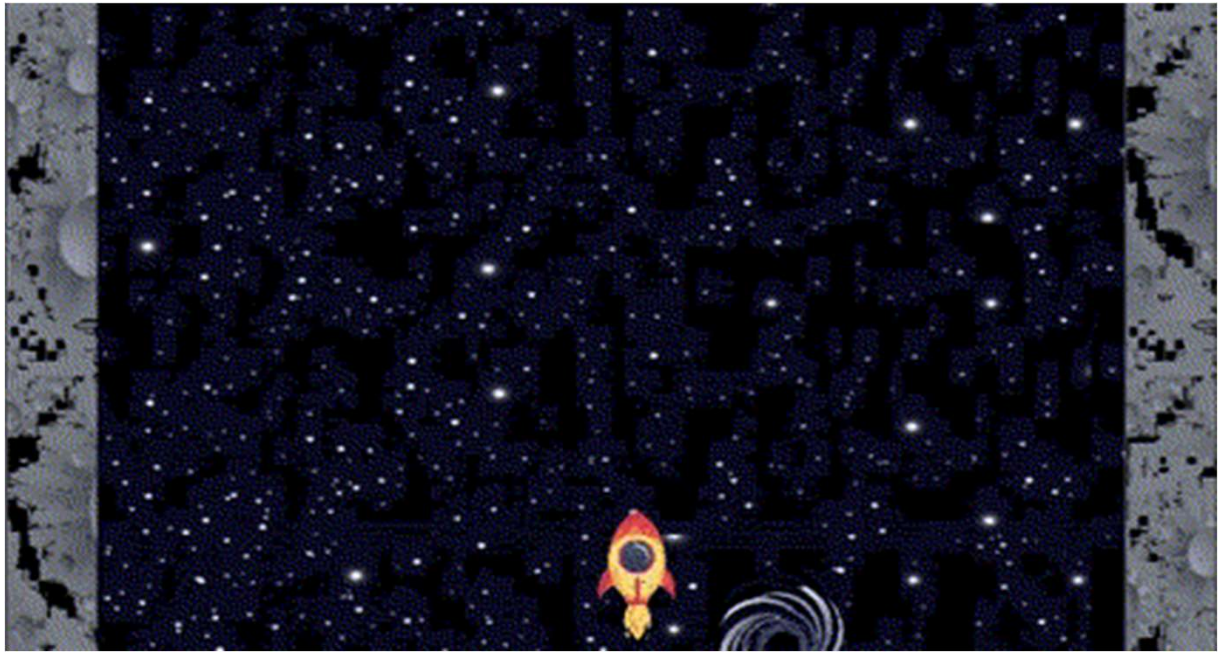
- Aqui está uma explicação linha por linha:

- Em resumo, este comando adiciona movimento ao objeto "**Hazard**", atualiza sua posição na tela, e reinicia sua posição e tipo aleatoriamente quando ele passa da altura da tela.

```
# seleciona um hazard aleatório
271                 h_x = random.randrange(125, 650 - h_height)
272                 hzrd = random.randint(0, 4)
```

VAMOS EXECUTAR O JOGO AGORA?!

Executando o código



Tela do jogo com hazards que tornam o jogo mais interativo e desafiador

Fonte: Imagem de catalyststuff no Freepik

Obrigada!

Até a próxima Aula!

PROGRAMA ESPACIAL BRASILEIRO

AEB escola

AEB AGÊNCIA ESPACIAL BRASILEIRA

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO