STEM²D

Go girls!

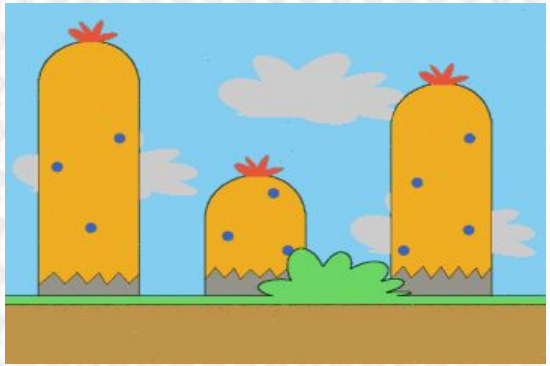

SMS
CODE
like a
girl

PARALAXE

- Para dar a ideia de movimento e progressão no jogo, além de implementar um algoritmo para o aparecimento, utilizaremos uma técnica de computação gráfica conhecida como **rolagem de paralaxe**.

PARALAXE

- Essa técnica consiste em mover as imagens de plano de fundo (tela de fundo) mais lentamente que imagens em primeiro plano (personagem principal, inimigos, vidas, etc.), criando uma ilusão de profundidade em uma cena 2D e aumentando a sensação de imersão na experiência virtual.



PARALAXE

- Para fazer isso, trabalhamos com uma imagem de fundo cuja largura é praticamente o dobro da largura da tela de jogo e, sempre, adicionamos à tela duas imagens de fundo: uma iniciando na posição **(imagemDeFundoX, 0)** da tela e outra iniciando na posição **(imagemDeFundoX2, 0)**, em que **imagemDeFundoX** corresponde à 0 e **imagemDeFundoX2** corresponde à largura da imagem de fundo.

PARALAXE

- Isso pode ser observado no método **desenharTelaBasica** da classe **Tela**, o qual desenha os elementos comuns a todas as telas do jogo.

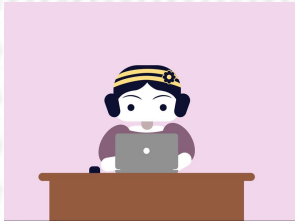
```
# desenha os elementos comuns a toda a tela, isto eh, a tela de fundo e o botao de audio
    def desenharTelaBasica(self, game):

        game.janela.blit(self.imagemDeFundo, (self.imagemDeFundoX, 0))
        game.janela.blit(self.imagemDeFundo, (self.imagemDeFundoX2, 0))

        # carrega a imagem do botao de audio de acordo com o status de audio do jogo
        if game.comAudio:
            self.botaoSom = pygame.image.load(os.path.join('Imagens', 'audio_ligado.png'))
        else:
            self.botaoSom = pygame.image.load(os.path.join('Imagens', 'audio_desligado.png'))
        game.janela.blit(self.botaoSom, (1200, 20))
```

PARALAXE

- Essa adição das imagens de fundo na tela utilizando os parâmetros **imagemDeFundoX** e **imagemDeFundoX2** viabiliza que implementemos a rolagem de paralaxe na classe que representa a tela de jogo (arquivo **TelaDeJogo.py**). Para cumprir esse objetivo, enquanto a tela de jogo estiver sendo executada, faremos as seguintes instruções:
  - Decrementar os valores das variáveis **imagemDeFundoX** e **imagemDeFundoX2**, com o intuito de fazer a imagem de fundo se mover para esquerda ao longo do tempo.



PARALAXE

- Alterar os valores das coordenadas **imagemDeFundoX** e **imagemDeFundoX2** quando toda a largura de uma das imagens de fundo estiver em posições negativas para posições positivas novamente, gerando uma espécie de laço de repetição de imagens de fundo.

PARALAXE

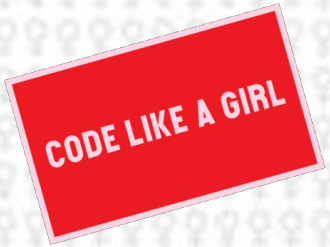
- Este código deve ser transcrito para o seu jogo!

```
def atualizar(self, game):
    ...
    # making background move
    self.imagemDeFundoX -= 2
    if self.imagemDeFundoX < self.imagemDeFundo.get_width() * -1:
        self.imagemDeFundoX = self.imagemDeFundo.get_width()
```

EXERCÍCIO

- Complete o método abaixo, de forma a viabilizar o mesmo comportamento para a variável **imagemDeFundoX2**. **Dica:**

```
def atualizar(self, game):
    ...
    # making background move
    self.imagemDeFundoX -= 2
    if self.imagemDeFundoX < self.imagemDeFundo.get_width() * -1:
        self.imagemDeFundoX = self.imagemDeFundo.get_width()
```



ATIRAR

- Foi implementado o aparecimento aleatório de alguns inimigos na tela de jogo. Estes inimigos irão atirar nela e, para se defender, ela deverá atirar neles. Por causa disso, iremos implementar o método **atirar** da classe **Jogador** (arquivo **Jogador.py**).

- O aparecimento de um tiro na tela ocorrerá toda vez que este método **atirar** da classe **Jogador** for chamado, o que ocorrerá toda vez que o usuário pressionar a barra de espaço.

ATIRAR

- Este aparecimento é semelhante ao de um obstáculo na tela, isto é, basta adicionar na lista **game.tiros**, utilizando o método **append**, um novo objeto **Tiro**, especificando sua inicial na tela de jogo, sua imagem e sua velocidade de deslocamento.

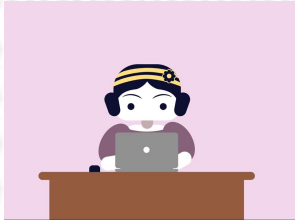
EXERCÍCIO

- Implemente o método **atirar** da classe **Jogador** (arquivo **Jogador.py**) de forma a viabilizar o aparecimento de tiros na tela de jogo. **Dica:**

```
def criarCenario(self, game):
    r = random.randrange(0, game.aparecimentoElementos)
    if r < 8:
         if len(game.obstaculos) == 0:
             game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 562 + 8*(r%4), pygame.image.load(os.path.join('Imagens',
'obstaculo_1_1.png')), 10 + game.dvel))
```

COLISÃO COM TIROS

- Com a inserção do tiro no jogo, teremos novos tipos de colisão:
  - Tiro do jogador com o inimigo;
  - Tiro do inimigo com o jogador;
  - Tiro do inimigo com o tiro do jogador;



COLISÃO COM TIROS

- Como esta é a nossa última aula, não achamos que seria viável que vocês implementassem todas estas colisões. Por causa disso, as colisões entre tiro do inimigo e jogador, bem como entre tiros já estão implementadas, como você pode verificar no método **checarColisoes** da classe Tiro (arquivo **Tiro.py**).

COLISÃO COM TIROS

- Não obstante, implementaremos a colisão entre os tiros e os inimigos. Basicamente, quando um tiro colide com um inimigo temos duas situações possíveis:
  - Se a variável **inimigo.vidas** for maior que 1, apenas iremos decrementá-la em 1 unidade, retirando uma vida do inimigo. Posteriormente, o tiro deve desaparecer da tela.
  - Do contrário (se a variável **inimigo.vidas** for nula), ao colidir com um tiro o inimigo irá desaparecer, indicando a vitória da jogadora.

RELEMBRANDO COMO FUNCIONA A DETECÇÃO DE COLISÕES

- ***Sprite.spritecollide / sprite.collide_mask / sprite.spritecollideany***

Retorna uma lista contendo todas as imagens do **objeto2** que colidiram com a imagem do **objeto1**

Imagem

Vetor de imagens

```
if self.rect.colliderect(objeto1):
  colisoes = pygame.spritecollide(objeto1, objeto2, False)
  callback = pygame.sprite.collide_mask
  colisao = pygame.sprite.spritecollideany(objeto1,colisoes,callback)
  if colisao:
    # fazer uma ação após a identificação de uma colisão
```

Cria as máscaras

Verifica se houve colisão usando o método passado, que no caso é o **collide_mask**, retornando True ou False.

EXERCÍCIO

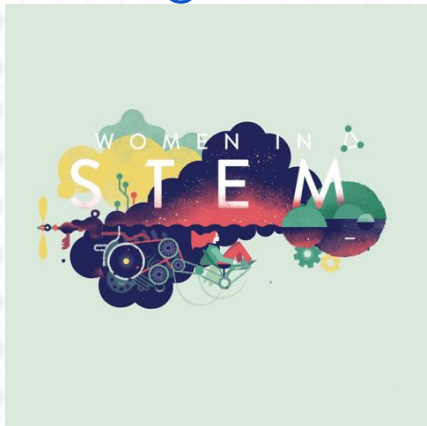
- Implemente a colisão entre tiros e inimigos no método **checarColisoes** da classe Tiro (arquivo **Tiro.py**), de acordo com as especificações do Slide 16. **Dica:**

```
if self.rect.colliderect(objeto1):
  colisoes = pygame.spritecollide(objeto1, objeto2, False)
  callback = pygame.sprite.collide_mask
  colisao = pygame.sprite.spritecollideany(objeto1,colisoes,callback)
  if colisao:
    # fazer uma ação após a identificação de uma colisão
```

SONORIZAÇÃO

- Neste momento, nosso jogo está quase finalizado (YAY!!!), mas ainda faltam os efeitos sonoros, tão essenciais para deixar a experiência do usuário completa.

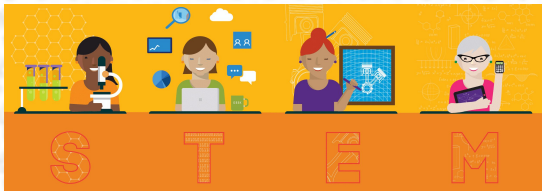
- Para implementar a sonorização, teremos auxílio de alguns métodos próprios do *Pygame*.



SONORIZAÇÃO

- Todo o áudio do nosso jogo é regido por uma variável denominada **pygame.mixer** que atuará como uma espécie de "rádio" do jogo.

- Este rádio possuirá dois canais, que são denominados **channels** para o *Pygame*. Basicamente, utilizaremos o *channel 0* para tocar a música de fundo de jogo e o *channel 1* para tocar efeitos sonoros específicos (como quando o jogador colide com uma vida).

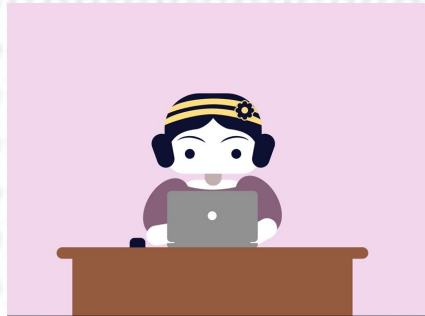


SONORIZAÇÃO

- Utilizaremos dois principais métodos para manipular o som:
  - **pygame.mixer.Channel(X).play(pygame.mixer.Sound(musica), Y):** basicamente, este método faz com que o áudio cujo nome do arquivo está armazenado na variável **musica** seja tocado no canal de número **X**. O valor **Y** indica se a música deve ser tocada apenas uma vez, o que ocorre para Y=0; ou se a música deve ficar repetindo para sempre, o que ocorre para Y= -1.

SONORIZAÇÃO

- Utilizaremos dois principais métodos para manipular o som:
  - **pygame.mixer.Channel(X).set_volume(Z):** altera o volume do som que está sendo tocado no channel de número **X** para a intensidade **Z**. Basicamente, **Z** varia entre 0 e 1, sendo Z=0 sem som e Z=1 som no volume máximo.



SONORIZAÇÃO

- O primeiro método que implementaremos na classe **AdministradorDeAudio** (arquivo **AdministradorDeAudio.py**) é o **tocarMusicaDeFundo**.

- Para fazer isso, basicamente, verificaremos se o áudio do jogo está ativado, isto é, se a variável **game.comAudio** é verdadeira e, se isso ocorrer, chamaremos o método **play** para tocar a música de fundo, bem como colocaremos o volume do canal de música de fundo (*channel 0*) para trabalhar em volume máximo.

SONORIZAÇÃO

- Este código deve ser transcrito para o seu jogo!

```
def tocarMusicaDeFundo(self, musica, game):
    # lembrar de checar se o jogo esta com audio ou nao
    # usar 1 canal para a musica de fundo e outro para os efeitos sonoros
    if (game.comAudio):
        pygame.mixer.Channel(0).play(pygame.mixer.Sound(musica), -1)
        pygame.mixer.Channel(0).set_volume(1)
```

SONORIZAÇÃO

- Para que a música de fundo toque, de fato, é preciso chamar o método **tocarMusicaDeFundo**, nos construtores de todas as telas do jogo.

- Este código deve ser transcrito para o seu jogo nos construtores das telas: **TelaDeInicio**, **TelaDeInstrucoes**, **TelaDeJogo** e **TelaDeFim**. Exemplo para a **TelaDeInicio**:

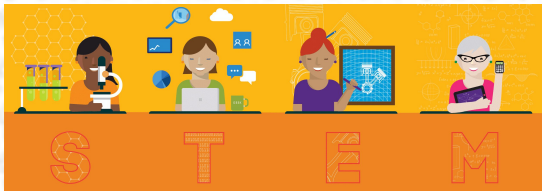
```
class TelaDeInicio(Tela):
   def __init__(self, game):
     # outras declarações
     game.administradorDeAudio.tocarMusicaDeFundo(os.path.join('Musica', '<nome da musica>'), game)
```

SONORIZAÇÃO

- O segundo método que implementaremos na classe **AdministradorDeAudio** (arquivo **AdministradorDeAudio.py**) é o **tocarEfeitoSonoro**.

SONORIZAÇÃO

- Para fazer isso, basicamente, verificaremos se o áudio do jogo está ativado, isto é, se a variável **game.comAudio** é verdadeira e, se isso ocorrer, chamaremos o método **play** para tocar a música de fundo, bem como colocaremos o volume do canal de efeitos sonoros (*channel 1*) para trabalhar em volume máximo e o volume do canal de música de fundo (*channel 0*) para trabalhar com 30% do volume máximo.



EXERCÍCIO

- Implemente o método **tocarEfeitoSonoro** na classe **AdministradorDeAudio** (arquivo **AdministradorDeAudio.py**).
  Dica:

```
def tocarMusicaDeFundo(self, musica, game):
    # lembrar de checar se o jogo esta com audio ou nao
    # usar 1 canal para a musica de fundo e outro para os efeitos sonoros
    if (game.comAudio):
        pygame.mixer.Channel(0).play(pygame.mixer.Sound(musica), -1)
        pygame.mixer.Channel(0).set_volume(1)
```

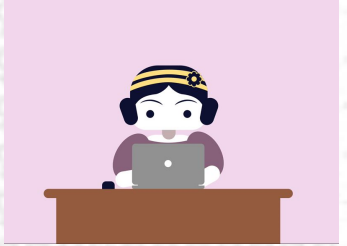
SONORIZAÇÃO

- Para que o efeito sonoro toque, de fato, é preciso chamar o método **tocarEfeitoSonoro**, nos principais casos de colisão do jogo.

- Este código deve ser transcrito para o seu jogo nos métodos **checarColisoes** da **Vida**, do **Impulsionador** e do **Obstaculo**. Exemplo para o **Obstaculo**:

```
pygame.mixer.pause()
game.administradorDeAudio.tocarEfeitoSonoro(os.path.join('Musica', 'death.wav'), game)
pygame.time.wait(3100)
game.administradorDeAudio.tocarMusicaDeFundo(os.path.join('Musica', '<nome da musica>'), game)
```

SONORIZAÇÃO



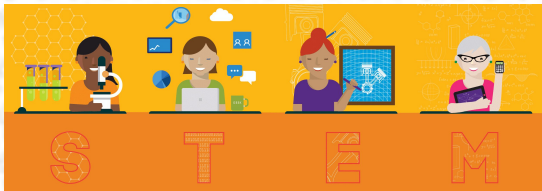
- O terceiro método que implementaremos na classe **AdministradorDeAudio** (arquivo **AdministradorDeAudio.py**) é o **deixarSomMudo**.

- Para fazer isso, basicamente, colocaremos o volume do canal de música de fundo (*channel 0*) e do canal de efeitos sonoros (*channel 1*) para trabalhar em volume mínimo. Além disso, tornaremos a variável **game.comAudio** falsa.

SONORIZAÇÃO

- Este código deve ser transcrito para o seu jogo!

```
def deixarSomMudo(self, game):
        pygame.mixer.Channel(0).set_volume(0)
        pygame.mixer.Channel(1).set_volume(0)
        game.comAudio = False
```



SONORIZAÇÃO

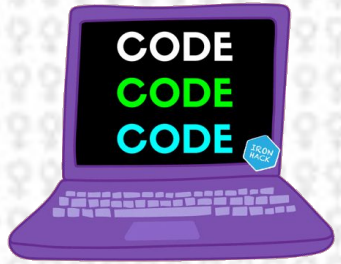
- O último método que implementaremos na classe **AdministradorDeAudio** (arquivo **AdministradorDeAudio.py**) é o **deixarSomTocar**.

- Para fazer isso, basicamente, colocaremos o volume do canal de música de fundo (*channel 0*) e do canal de efeitos sonoros (*channel 1*) para trabalhar em volume máximo. Além disso, tornaremos a variável **game.comAudio** verdadeira.

EXERCÍCIO

- Implemente o método **deixarSomTocar** na classe **AdministradorDeAudio** (arquivo **AdministradorDeAudio.py**).
  Dica:

```
def deixarSomMudo(self, game):
        pygame.mixer.Channel(0).set_volume(0)
        pygame.mixer.Channel(1).set_volume(0)
        game.comAudio = False
```




STEM2D
Go girls!
Muito obrigada!