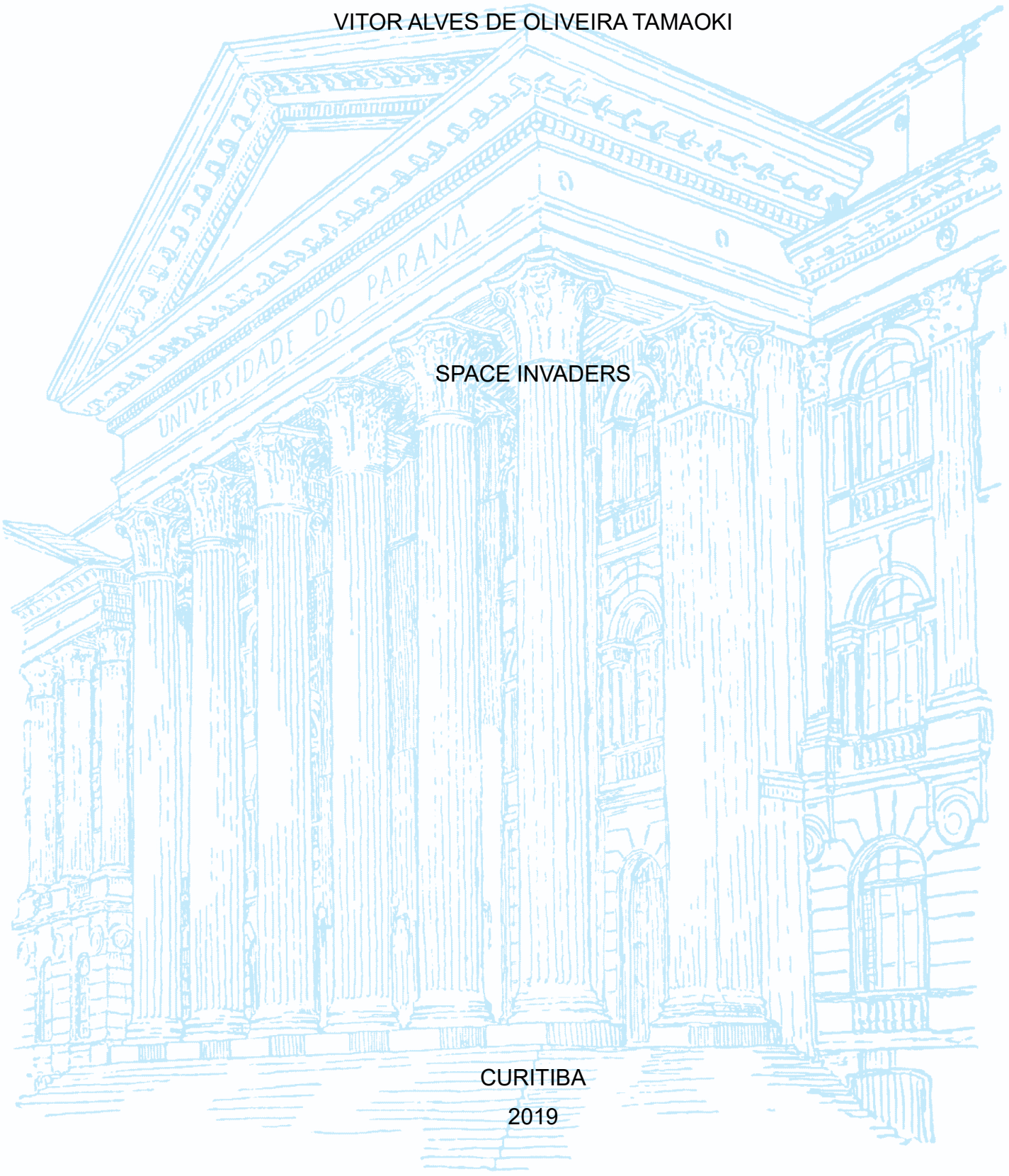
UNIVERSIDADE FEDERAL DO PARANÁ
CURSO DE AGRONOMIA

CATHERINE DA SILVEIRA FLEISCHMANN
VITOR ALVES DE OLIVEIRA TAMAOKI



SPACE INVADERS

CURITIBA
2019

CATHERINE DA SILVEIRA FLEISCHMANN(GRR: 20196427)
VITOR ALVES DE OLIVEIRA TAMAOKI(GRR: 20194231)

SPACE INVANDERS EM PYTHON

Relatório do trabalho final da disciplina Fundamentos de Programação de Computadores, do Curso de Graduação em Agronomia da Universidade Federal do Paraná.

Orientador: Professor Jackson Antônio do Prado Lima.

CURITIBA

2019

SUMÁRIO

RESUMO

O presente trabalho refere-se à criação de um jogo em Python, de maneira simples e com a utilização de poucas bibliotecas foi possível criar uma versão do jogo Space Invaders. A jogabilidade ocorre através de comandos básicos: direita, esquerda e espaço, tornando o jogo acessível até para crianças. O objetivo do Space Invaders é que o usuário impeça que naves inimigas atinjam a sua, para se defender o jogador utiliza a barra de espaços, comando para que a nave lance um laser em direção às naves invasoras, caso o jogador não consiga impedi-las de atingi-lo o jogo se encerra.

Lista de Figuras

Desenvolvimento

Para o desenvolvimento do programa foi necessária a utilização da biblioteca *turtle,* que auxilia no desenvolvimento da tela de jogo.

Figura 1 – Biblioteca turtle

O primeiro passo após a importação do módulo turtle, foi criar e organizar a tela de jogo, definindo a cor do background – variável tela.bgcolor(“black”) ,tamanho da tela – variável tela.stworldcoordinates(400, - 400, 400, 400) – e inserindo os formatos a serem visualizados pelo usuário.

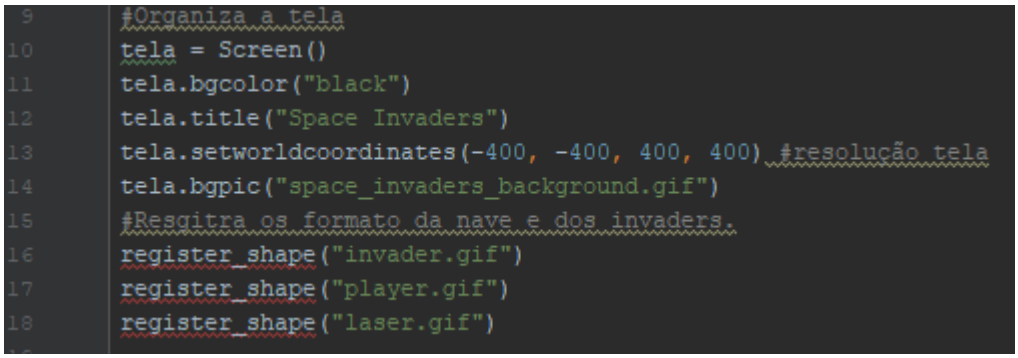


Figura 2 – Organizar a tela

A seguir, foi criado um campo onde ocorre o jogo, delimitado por uma borda um pouco menor que o tamanho de tela total definido no passo acima. Nos passos seguintes foi-se definindo o layout geral do jogo, criando o campo onde é mostrada a pontuação do usuário e desenhando a nave. As variáveis nave.setposition(0, -280) e mov_nave = 15, definem a posição inicial da nave do usuário e a velocidade de movimentação da mesma.

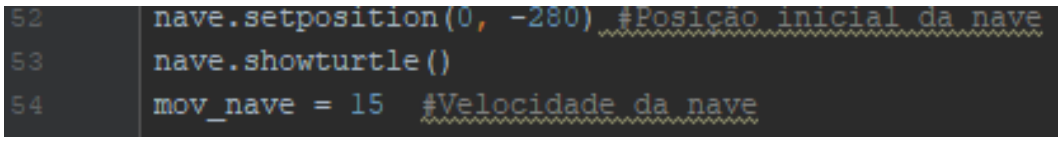


Figura 3 – Posição e velocidade da nave

Para a criação de naves inimigas, foi definido um número de 5 naves inimigas, criamos uma lista de inimigos[ ] e um laço de repetição que adiciona os inimigos na lista.

```
n_inimigos = 5
#Lista de inimigos
inimigos = []
#Adiciona os inimigos na lista.
for i in range(n_inimigos):
    #Acrescenta turtles dentro da lista
    inimigos.append(Turtle())
```

Figura 4 – Lista inimigos

Para determinar o que acontece com a nave inimiga quando atingida pelo usuário, foi ciado o laço *for inimigo in inimigos*, em seguida, com a criação das variáve*is x* e y junto a utilização do comando *setpos* tornou-se possível determinar que quando atingida pelo usuário a nave inimiga é direcionada a uma posição aleatório do campo de jogo.

```
for inimigo in inimigos:

    inimigo.shape("invader.gif")
    inimigo.color("red")
    inimigo.speed(0)
    inimigo.penup()
    x = random.randint(-200, 200)
    y = random.randint(100, 250)
    inimigo.setpos(x, y)
```

Figura 5 – Laço posição aleatória inimigos

Em seguida, foram definidos a velocidade das naves inimigas e criado laser. Para definir a forma como a nave do jogador se movimenta em campo, foram criadas as seguintes funções:

```
def esquerda():
    x = nave.xcor()
    x -= mov_nave
    if x < -280:
        x = - 280
    nave.setx(x)
def direita():
    x = nave.xcor()
    x += mov_nave
    if x > 280:
        x = 280
    nave.setx(x)
```

Figura 6 – Funções de movimentação da nave

As funções *def esquerda()* e *def direita(),* determinam o comportamento das naves em suas movimentações para a esquerda e para direita, onde a condição *if* serve para impedir que elas ultrapassem os limites da borda pré-definida do campo.

Para a criação e utilização do laser, foram necessárias duas funções, uma para quando o usuário atira e outra para caso o laser colida com uma nave inimiga. Na

função *def solta_laser*, primeiramente foi utilizada a função global *est_laser* – que já havia sido definida previamente – para facilitar o trabalho em caso de necessidade de edição no código do laser. Em seguida, foi criada uma condição *if* que determina a posição do laser em relação às naves quando atirado e onde foi adicionado o som de quando o laser é disparado.

```
def solta_laser():
    global est_laser #Define como uma variavel global, caso precise de mudança.
    if est_laser == "ready":
        winsound.PlaySound("laser", winsound.SND_ASYNC)
        est_laser = "fire"
    #Posição do laser em relação a nave:
        x = nave.xcor()
        y = nave.ycor()
        laser.setpos(x, y + 10)
        laser.showturtle()
```

Figura 7 – Função laser

Para determinar se houve ou não colisão do laser com a nave inimiga, foi criad*a* a função *def eColisão(t1, t2)* que tem como objetivo calcular a distância entre o laser e o inimigo, caso a distância entre eles seja inferior a 15 pixels, o programa deve retornar a informação de que a nave foi atingida.

```
def eColisao(t1, t2):
    distancia = sqrt(pow(t1.xcor()-t2.xcor(), 2) + pow(t1.ycor()-t2.ycor(), 2))
    if distancia < 15:
        return True
    else:
        return False
```

Figuras 8 e 9 – Função colisão

O laço *while true* é a parte principal para o funcionamento do jogo, dentro dele encontram-se os laços e condicionais a seguir:

O primeiro laço dentro do *while true, for inimigo in inimigos* faz com que as naves inimigas mudem de posição quando atingem a borda do campo, dentro das condicionais *if inimigo.xcor() < 280* e *if inimigo.xcor() < 280* as naves são movimentas para baixo ao atingirem a borda e mudam a direção de seu avanço, da direita para a esquerda e vice-versa.

```
while True:
    for inimigo in inimigos:
        #Movimenta o inimigo.
        x = inimigo.xcor()
        x += mov_inimigo
        inimigo.setx(x) #muda para a nova posição
        # Quando atinge a borda desce/sobe e muda o sentido do movimento .
        if inimigo.xcor() > 280:
            #Movimenta todos os inimigos para baixo
            for e in inimigos:
                y = e.ycor()
                y -= 40
                e.sety(y)
            #Muda direção
            mov_inimigo *= -1
```

```
if inimigo.xcor() < -280:
    # Movimenta todos os inimigos para baixo
    for e in inimigos:
        y = e.ycor()
        y -= 40
        e.sety(y)
    # Muda direção
    mov_inimigo *= -1
```

Figuras 10 e 11 – While true

Após os laços e condições que determinam a movimentação das naves inimigas, há uma condição para definir o comportamento do programa quando o laser atinge a nave. Dentro da condicional *if eColisao (laser, inimigo)* foi adicionado o áudio que corresponde com a nave sendo atingida, depois de atingir o alvo o laser é resetado, o jogador pode atirar novamente e a nave atingida é movida para um ponto aleatório do campo, ao atingir uma nave inimiga são acrescidos ao campo *score* dez pontos.

```
if eColisao(laser, inimigo):
    winsound.PlaySound("explosion", winsound.SND_ASYNC)
    #reseta o laser após a colisão com o alvo
    laser.hideturtle()
    est_laser = "ready" #permite o laser ser atirado novamente após a colisão com o alvo
    laser.setposition(0, -400)
    #reseta o inimigo
    x = random.randint(-200, 200)  # Posição aleatoria do inimigo
    y = random.randint(100, 250)  # Posição aleatoria do inimigo
    inimigo.setpos(x, y)
    #Atualiza o score
    score += 10
    scorestring = "Score: %s" %score
    score_pen.clear()   #Limpa o score
    score_pen.write(scorestring, False, align="left", font=("Arial", 10, "normal"))
```

Figura 12 – Condicional de colisão e pontuação

Caso o laser não atinja nenhuma nave ele é resetado e as naves permanecem seu percurso atual, e caso a nave do usuário seja atingida por uma nave inimiga o jogo se encerra e aparece na tela a mensagem “Game Over”.

```
    #Colisão entre a nave e o inimigo
    if eColisao(nave, inimigo):
        nave.hideturtle()
        inimigo.hideturtle()
        print("Game Over")
        break

#Movimento do laser:
if est_laser == "fire":
    y = laser.ycor()
    y += mov_laser
    laser.sety(y)
#Verifica se o laser atingiu a borda:
if laser.ycor() > 275:
    laser.hideturtle()
    est_laser = "ready"
```

Figura 13 – Game over e movimentação do laser

Conclusão

Para a criação do Space Invaders sem a utilização do Pygame, foi necessário que buscássemos opções de biblioteca que permitissem a criação do jogo. Com a utilização do módulo turtle conseguimos criar a interface e com os conhecimentos adquiridos em aula, nos foi permitido criar laços e funções para a parte interativa do programa. Apesar de simples, o código do jogo possui os elementos aprendidos durante o semestre e possui espaço para atualizações e melhorias, tais como: maior velocidade das naves, número limitado de vidas para o usuário, etc.

Referências

https://projects.codeclubworld.org/p-BR/09_python/04/Turtle%20Power.html

https://docs.python.org/3/library/turtle.html

http://christianthompson.com/