

Relatório do Trabalho 1 de Computação Gráfica

Mateus Charlote, Marcelo Martins

31 de maio de 2024

0. Link para o vídeo

https://www.youtube.com/watch?v=6Fc_IYE6Eew

Controles: UP acelera, DOWN desacelera, LEFT rotaciona para esquerda, RIGHT rotaciona para direita, SPACE atira.

1. Introdução

Este trabalho prático da disciplina de Computação Gráfica tem como objetivo desenvolver um jogo utilizando a biblioteca *OpenGL*. No jogo, o usuário controla um “disparador de tiros” cujo objetivo é eliminar naves inimigas que o cercam e podem destruí-lo. O disparador pode se mover para frente e para trás, além de controlar a direção de seu movimento. Ele também possui um mecanismo para atirar de maneira controlada. Imagine uma versão aprimorada do clássico jogo *Asteroides*. Optamos por manter a temática do espaço e utilizamos uma paleta de cores contrastante entre elas, assim dando um visual bem “arcade” para o jogo. Escolhemos utilizar a linguagem de programação Python para a implementação deste projeto.

2. Objetivo do jogo

Durante uma sessão de jogo, o tempo que o jogador permanece jogando é cuidadosamente registrado. O objetivo principal é sobreviver o máximo de tempo possível, acumulando pontos tanto por derrotar inimigos quanto pela duração de sua sobrevivência. Cada segundo conta e cada inimigo abatido contribui para a pontuação final, criando uma experiência intensa e desafiadora, pois os inimigos são gerados a cada xx segundos. A habilidade do jogador em se manter vivo e eficiente na eliminação de adversários é crucial para alcançar altas pontuações e “vencer” o jogo.

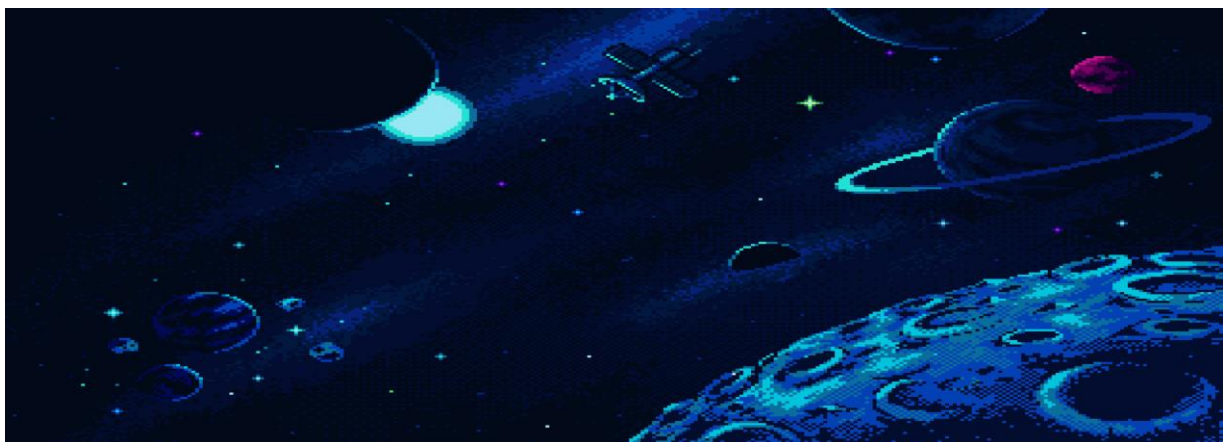
3. Modelos gráficos

Inicialmente, focamos na representação visual do nosso jogo, criando modelos matriciais básicos que seriam utilizados na implementação. Todos esses modelos foram convertidos manualmente em arquivos *.txt* para que pudéssemos usar a função já fornecida.



Plano de fundo do jogo:

Para utilizar uma imagem como *background* em nosso ambiente Python precisamos instalar a biblioteca *pillow* e configurar o *OpenGL* para carregar e desenhar a textura da imagem. Primeiramente foi criada uma variável global chamada *background_texture* responsável por armazenar a textura, e a função *load_texture(path)* que recebe um caminho por parâmetro e carrega a imagem, convertendo-a em uma textura que pode ser usada pelo *OpenGL*. Após o carregamento da textura ser armazenado na variável global utilizamos a função *draw_background()* que verifica se a textura foi carregada corretamente, vincula a textura e desenha um quadrado cobrindo a tela com as coordenadas de textura mapeadas. Este conjunto de funções permite que a imagem seja desenhada como fundo na tela do nosso ambiente gráfico. Imagem em questão:



load_texture(path) && draw_background()

Funções implementadas na classe “principal” utilizando *.open* e *.convert* que a biblioteca *pillow* disponibiliza complementando com *OpenGL*. Ambas as funções possuem código comentado linha por linha para fácil compreensão:

```
1 def load_texture(path):
2     img = Image.open(path) # abre a imagem
3     img_data = img.convert("RGBA").tobytes() # converter a imagem para RGBA e armazena os bytes
4
5     width, height = img.size # pega a largura e altura da imagem
6
7     texture_id = glGenTextures(1) # gera um id para a textura
8     glBindTexture(GL_TEXTURE_2D, texture_id) # faz o bind da textura
9     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, img_data) # carrega a textura
10
11     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR) # define o filtro de minificação
12     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR) # define o filtro de magnificação
13
14     return texture_id # retorna o id da textura
15
16 def draw_background():
17     global background_texture
18
19     if background_texture: # se a textura foi carregada
20         glBindTexture(GL_TEXTURE_2D, background_texture) # faz o bind da textura
21         glEnable(GL_TEXTURE_2D) # habilita a textura
22
23         glBegin(GL_QUADS) # desenha um quadrado com a textura
24         glTexCoord2f(0.0, 0.0) # coordenadas da textura
25         glVertex2f(Min.x, Min.y) # vértice 1
26
27         glTexCoord2f(1.0, 0.0) # coordenadas da textura
28         glVertex2f(Max.x, Min.y) # vértice 2
29
30         glTexCoord2f(1.0, 1.0) # coordenadas da textura
31         glVertex2f(Max.x, Max.y) # vértice 3
32
33         glTexCoord2f(0.0, 1.0) # coordenadas da textura
34         glVertex2f(Min.x, Max.y) # vértice 4
35         glEnd() # finaliza o desenho do quadrado
36
37         glDisable(GL_TEXTURE_2D) # desabilita a textura
38
```

4. Menu e hud

Para proporcionar uma transição suave para os jogadores, criamos a classe *menu.py*. Esta gera graficamente o menu inicial utilizando os recursos da nossa biblioteca *pyOpenGL*. Desenhamos o menu na tela de forma harmoniosa e confortável para os olhos, garantindo uma experiência agradável para o jogador desde o início. Optamos por criar tanto o menu quanto o hud em classes separadas para melhor visualização e fácil implementação. A classe *hud.py* desempenha um papel crucial na nossa solução, pois armazena a vida e pontos do jogador. Além de guardar essas informações, a classe é responsável por exibi-las durante o jogo, mostrando também o tempo para o jogador. Ambos os códigos comentados linha por linha:

Menu.py

```
1  """ Classe Menu """
2  class Menu:
3
4      def menuPrincipal(self):
5          texto = "SPACE TRAVELER"
6          texto2 = "[ 1 ] START GAME"
7          texto3 = "[ ESC ] QUIT GAME"
8
9          glColor3f(0.75, 0.75, 0.75)          # texto cor cinza
10         glRasterPos2f(-20,50)                  # posicao do texto na tela
11         for i in range(len(texto)):            # percorre cada caractere do texto
12             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(texto[i])) # imprime o texto
13
14         glColor3f(0.75, 0.75, 0.75)          # texto2 cor cinza
15         glRasterPos2f(-20,-50)                # posicao do texto2 na tela
16         for i in range(len(texto2)):          # percorre cada caractere do texto
17             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(texto2[i])) # imprime o texto
18
19         glColor3f(0.75, 0.75, 0.75)          # texto3 cor cinza
20         glRasterPos2f(-20,-60)                # posicao do texto3 na tela
21         for i in range(len(texto3)):          # percorre cada caractere do texto
22             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(texto3[i])) # imprime o texto
23
24
25         # Faz uma moldura para o titulo
26         glBegin(GL_LINE_LOOP)                # Desenha um quadrado
27         glVertex2f(-30, 60)                  # Vertice 1
28         glVertex2f(50, 60)                   # Vertice 2
29         glVertex2f(50, 40)                   # Vertice 3
30         glVertex2f(-30, 40)                  # Vertice 4
31         glEnd()
```

Hud.py

```
1  """ Classe Hud """
2  class Hud:
3
4      def __init__(self):                    # Construtor da classe
5          self.vida = 100                   # Vida do jogador
6          self.pontos = 0                   # Pontos do jogador
7
8      def mostraHud(self, tempo):           # Mostra o HUD na tela
9
10         glColor3f(0.75, 0.75, 0.75)       # texto cor cinza
11
12         textoVida = "Life: " + str(self.vida) + "%"
13         glRasterPos2f(-145,140)            # posicao do texto na tela
14         for i in range(len(textoVida)):    # percorre cada caractere do texto
15             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(textoVida[i])) # imprime o texto
16
17         textoPontos = "Score: " + str(int(self.pontos))
18         glRasterPos2f(-145,130)            # posicao do texto na tela
19         for i in range(len(textoPontos)):  # percorre cada caractere do texto
20             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(textoPontos[i])) # imprime o texto
21
22         textoTempo = "Time: " + str(tempo)
23         glRasterPos2f(-75,140)             # posicao do texto na tela
24         for i in range(len(textoTempo)):   # percorre cada caractere do texto
25             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(textoTempo[i])) # imprime o texto
26
27
28         def perdeVida(self, dano):          # Função que diminui a vida do jogador
29             self.vida -= dano               # Diminui a vida do jogador dado o dano
30
31         def ganhaVida(self, cura):          # Função que aumenta a vida do jogador
32             self.vida += cura               # Aumenta a vida do jogador dado a cura
33
34         def ganhaPontos(self, pontos):      # Função que aumenta os pontos do jogador
35             self.pontos += pontos           # Aumenta os pontos do jogador dado os pontos
```

5. Funções alteradas e criadas

Nesta sessão, apresentaremos todas as funções que foram criadas ou modificadas na nossa implementação do projeto. Para cada função, faremos uma breve explicação do seu propósito e, em seguida, exibiremos o código comentado linha por linha para facilitar a compreensão.

Display()

A função *display()* é responsável por desenhar e atualizar a tela do jogo, garantindo que todos os elementos sejam corretamente renderizados e atualizados a cada quadro. Ela também lida com a lógica de final de jogo e a exibição do menu. Inicialmente, são configuradas as variáveis de tempo que serão utilizadas. Em seguida, a função *draw_background()* é chamada para assegurar que o fundo do jogo seja desenhado desde o início. Uma variável auxiliar chamada *menuAtivado* facilita a transição entre o menu inicial e o jogo propriamente dito, especialmente no caso de o jogador perder. Nesse momento, a lógica de *game over* é acionada, exibindo a pontuação total e uma tela de fim de jogo. Código comentado linha por linha:

```
1 def display():
2     global TempoInicial, TempoTotal, TempoAnterior, PersonagemAtual, nInstancias, jogo, menuAtivado, Personagens
3
4     TempoAtual = time.time() # pega o tempo atual
5     TempoTotal = TempoAtual - TempoInicial # calcula o tempo total
6     DiferencaDeTempo = TempoAtual - TempoAnterior # calcula a diferença de tempo
7
8     # desenha o background
9     draw_background()
10
11     # Desenha o menu
12     if menuAtivado: # se o menu estiver ativo
13         menu.menuPrincipal() # desenha o menu principal
14
15     if not menuAtivado: # se o menu não estiver ativo
16
17         # zera o tempo de execução
18         if jogo:
19             TempoInicial = time.time() # zerando o tempo de execução
20
21         hud.mostraHud(int(TempoTotal)) # mostra o hud do jogo
22         DesenhaPersonagens() # desenha os personagens
23         AtualizaPersonagens(DiferencaDeTempo) # atualiza os personagens
24
25         # se vida <= 0, fim de jogo
26         if hud.vida <= 0:
27
28             # limpa a tela
29             glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
30             draw_background() # desenha o background
31
32             Personagens[0].Visivel = False # esconde o disparador
33             glColor3f(0.75, 0.75, 0.75) # cor cinza para o texto
34
35             texto = "G A M E   O V E R" # texto de fim de jogo
36             glRasterPos2f(-20, 50) # posição do texto
37             for i in range(len(texto)): # loop para desenhar o texto
38                 glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(texto[i]))
39
40             texto = "S C O R E : " + str(hud.pontos) # texto de pontuação
41             glRasterPos2f(-20, 40) # posição do texto
42             for i in range(len(texto)): # loop para desenhar o texto
43                 glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(texto[i]))
44
45             jogo = False # fim de jogo
46
47         glutSwapBuffers() # troca os buffers
48         TempoAnterior = TempoAtual # atualiza o tempo anterior
```


GeraVetorTiros()

A função *GeraVetorTiros()* realiza a distribuição dos tiros entre as instâncias de personagens, registrando essa informação em um dicionário chamado *dict*, onde cada chave é o identificador da instância e o valor é uma lista que armazena os tiros disponíveis para essa instância em específico. Como o disparador pode ter um número máximo de tiros diferente dos inimigos, há uma variável de controle de *loop* chamada *max_tiros*, à qual se atribui, primeiramente, o número máximo de tiros do disparador e, em seguida dos inimigos. É importante mencionar que a lista não armazena um objeto em si, e sim um inteiro que representa a posição do tiro no vetor *Personagens*, que armazena todas as instâncias do jogo.

```
1 def GeraVetorTiros():
2     global dict, NUM_MAX_TIROS
3     tirosDisponiveis = nInstancias - nTiros           # tiros disponiveis
4     instanciasNTiros = nInstancias - nTiros          # instancias com tiros
5     for i in range(0, instanciasNTiros):              # para cada instancia com tiros
6         max_tiros = NUM_MAX_TIROS_DISPARADOR if i == 0 else NUM_MAX_TIROS # quantidade de tiros
7         for x in range(0, max_tiros):                 # para cada tiro
8             dict[Personagens[i].Id].append(tirosDisponiveis) # adiciona o tiro ao vetor
9             tirosDisponiveis += 1                     # incrementa o tiro disponivel
10
```

Atira(instancia)

A função *Atira(instancia)* é responsável por gerenciar o disparo de tiro dos personagens. Ela recebe a instância que deseja atirar e, então, acessa a lista de tiros associada a esta instância a partir do dicionário *dict* gerado pela função *GeraVetorTiros()*. Em seguida, percorre essa lista e, no caso de encontrar um tiro disponível (um tiro está disponível se é positivo; se negativo, ele ainda está posicionado na tela), ocorre seu disparo, ele é negativado e, então, a iteração é interrompida.

```
1 def Atira(instancia):
2
3     id_instancia = instancia.Id # id da instancia
4     tiros = dict[id_instancia] # vetor de tiros
5
6     for tiro in tiros:          # para cada tiro
7         if tiro > 0:            # se o tiro estiver disponivel
8             idx = tiros.index(tiro) # pega o indice do tiro
9             dict[id_instancia][idx] = -tiro # marca o tiro como indisponivel
10            Personagens[tiro].Visivel = True # torna o tiro visivel
11            Personagens[tiro].Posicao = (instancia.Envelope[1] + instancia.Envelope[2])*0.5 # posicao do tiro
12            Personagens[tiro].Velocidade = 160 # velocidade do tiro
13            Personagens[tiro].Rotacao = instancia.Rotacao # rotacao do tiro
14            Personagens[tiro].Direcao = instancia.Direcao.getPonto() # direcao do tiro
15            break # sai do loop
```

AtualizaJogo()

Na função *AtualizaJogo()*, é realizada a verificação de colisão entre os personagens através da função *TestaColisao()* previamente disponibilizada. Há um laço de repetição responsável por verificar se os tiros do disparador atingiram cada um dos inimigos. Além disso, há outro laço para verificar se há colisão entre o disparador e os inimigos, assim como entre o disparador e os tiros dos inimigos. Para cada tiro e para cada inimigo, a função *TestaColisao()* é chamada.

Na função *AtualizaJogo()*, também é verificado quando as instâncias chegam no limite do espaço do jogo. O disparador “atravessa” a tela, reaparecendo do outro lado; os inimigos retornam como uma bola de bilhar ao bater no lado de uma mesa; e os tiros “desaparecem”, isto é, sua propriedade de visibilidade é desabilitada, permitindo que sejam reutilizáveis por quem os disparou.

Uma observação importante é que se tentou implementar a colisão e a verificação de limite do espaço como funções; ao serem utilizadas, porém, o jogo começou a apresentar problemas de desempenho, sendo observada uma grande lentidão. Por isso, então, essas funcionalidades foram implementadas através de laços de repetição.

A função *AtualizaJogo()* também é utilizada para verificar se o disparador foi atingido por um inimigo. Em caso afirmativo, é chamada a função *PersonagemAtingidoPisca()*, que inverte o estado de sua visibilidade. Além disso, como o jogo permite definir a quantidade inicial de inimigos na tela, a *AtualizaJogo()* verifica se há inimigos ocultos e, se sim, é chamada a função *CarregaInimigosOcultos()*. A *AtualizaJogo()* também é responsável por fazer a coleta de “lixo” da tela, isto é, desabilitar a visibilidade dos inimigos mortos. Por fim, ela é responsável por chamar a função *AtiraInimigos()*, para solicitar o disparo de tiro dos inimigos.

```

1 def AtualizaJogo():
2     global imprimeEnvelope, nInstancias, Personagens, atirou, t, inimigos_ocultos, dif, inimigoAtingiuDisparador, cont, tempo, segundoCompleto
3     TempoAtual = time.time()
4
5     if segundoCompleto != int(TempoTotal):
6         # a cada 2 segundos, é carregado um inimigo que estava oculto
7         if inimigos_ocultos and segundoCompleto % 2 == 0 and segundoCompleto != 0:
8             CarregaInimigosOcultos()
9             segundoCompleto = int(TempoTotal)
10
11     if contadorMeioSegundo(TempoAtual):
12         AtiraInimigos()
13         # a cada meio segundo, são limpados da tela os inimigos mortos (que se transformam em explosão)
14         if not inimigos_mortos.empty():
15             inimigo_morto = inimigos_mortos.get()
16             Personagens[inimigo_morto].Visivel = False
17
18     if inimigoAtingiuDisparador:
19         if cont < 3:
20             PersonagemAtingidoPisca(0)
21             cont += TempoAtual - TempoAnterior
22         else:
23             inimigoAtingiuDisparador = False
24             Personagens[0].Visivel = True
25             cont = 0
26
27     for i in range(0, nInstancias):
28         posx = Personagens[i].Posicao.getX()
29         posy = Personagens[i].Posicao.getY()
30         if (Personagens[i].Tipo == TipoInstancia.DISPARADOR):
31             if (posx > LarguraDoUniverso):
32                 Personagens[i].Posicao.set(-LarguraDoUniverso, posy)
33
34             if (posx < -LarguraDoUniverso):
35                 Personagens[i].Posicao.set(LarguraDoUniverso, posy)
36
37             if (posy > LarguraDoUniverso):
38                 Personagens[i].Posicao.set(posx, -LarguraDoUniverso)
39
40             if (posy < -LarguraDoUniverso):
41                 Personagens[i].Posicao.set(posx, LarguraDoUniverso)
42
43             elif (Personagens[i].Tipo == TipoInstancia.INIMIGO):
44                 if (posx > LarguraDoUniverso-11 or posx < (-LarguraDoUniverso+11)
45                     or posy > LarguraDoUniverso-11 or posy < (-LarguraDoUniverso+11)):
46                     r = random.randint(1, 20)
47                     ang = math.degrees(math.atan2(posy, posx)) + 90
48                     Personagens[i].Rotacao = ang + r
49                     Personagens[i].Direcao = Ponto(0, 1)
50                     Personagens[i].Direcao.rotacionaZ(ang + r)
51
52             elif (Personagens[i].Tipo == TipoInstancia.TIRO):
53                 atirador = Personagens[i].Id
54                 if (posx > LarguraDoUniverso or posx < -LarguraDoUniverso or posy > LarguraDoUniverso or posy < -LarguraDoUniverso):
55                     tiros_atirador = dict[Personagens[atirador].Id]
56                     idx = tiros_atirador.index(-i)
57                     tiros_atirador[idx] = tiros_atirador[idx] * -1
58                     Personagens[i].Posicao = Ponto(LarguraDoUniverso, LarguraDoUniverso)
59                     Personagens[i].Visivel = False
60                     Personagens[i].Velocidade = 0
61
62     # verifica se há colisão entre o disparador e os tiros inimigos e entre o disparador e os inimigos
63     for i in range(1, nInstancias):
64         if Personagens[i].Visivel and not inimigoAtingiuDisparador and TestaColisao(0, i):
65             if Personagens[i].Tipo == TipoInstancia.TIRO and -i not in dict[0]:
66                 Personagens[i].Visivel = False
67                 hud.perdeVida(5)
68
69             elif Personagens[i].Tipo == TipoInstancia.INIMIGO:
70                 hud.perdeVida(20)
71                 inimigoAtingiuDisparador = True
72
73     # atualiza envelope
74     for i in range(0, nInstancias):
75         AtualizaEnvelope(i)
76
77     # tiros do disparador contra inimigos
78     nInstanciasTiros = NUM_INIMIGOS + 1
79     idx_final_tiros_disparador = nInstanciasTiros + NUM_MAX_TIROS_DISPARADOR
80     for i in range(nInstanciasTiros, idx_final_tiros_disparador):
81         if Personagens[i].Visivel:
82             for idx_inimigo in idx_inimigos:
83                 if Personagens[idx_inimigo].Visivel and TestaColisao(i, idx_inimigo):
84                     Personagens[i].Visivel = False
85                     Personagens[idx_inimigo].IdDoModelo = 7
86                     idx_inimigos.remove(idx_inimigo)
87                     inimigos_mortos.put(idx_inimigo)
88                     Personagens[idx_inimigo].Velocidade = 0
89                     Personagens[idx_inimigo].Tipo = TipoInstancia.EXPLOSAO
90                     hud.ganhaPontos(10)
91

```


AtiraInimigos()

A função *AtiraInimigos()* é responsável por controlar as chamadas da função *Atira()* por parte dos personagens inimigos. Ela percorre a lista de índices de inimigos (índices relativos à lista *Personagens*) e, caso o personagem esteja visível na tela, ele está apto a fazer uma requisição de disparo de tiro – quem determinará se o tiro ocorrerá é a função *Atira()*.

```
1 def AtiraInimigos():
2     for idx in idx_inimigos:           # para cada inimigo
3         if Personagens[idx].Visivel:   # se o inimigo estiver visivel
4             Atira(Personagens[idx])    # atira
```

GeraPosicaoAleatoria()

A função *GeraPosicaoAleatoria()* é utilizada para gerar uma posição aleatória com coordenadas *x* e *y* variando entre $-LarguraDoUniverso$ e $+LarguraDoUniverso$. Ela retorna um objeto do tipo *Ponto* instanciado a partir destas coordenadas.

```
1 def GeraPosicaoAleatoria():
2     x = random.uniform(-LarguraDoUniverso, LarguraDoUniverso) # gera um valor aleatório para x
3     y = random.uniform(-LarguraDoUniverso, LarguraDoUniverso) # gera um valor aleatório para y
4     return Ponto(x, y) # retorna um ponto com as coordenadas x e y
5
```


CarregaInimigosOcultos()

A função *CarregaInimigosOcultos()* ativa um inimigo oculto da tela. Ela remove o primeiro inimigo de uma fila onde estão armazenados os inimigos ocultos, tornando-o visível e ativando sua velocidade.

```
1 def CarregaInimigosOcultos():
2     inimigo = inimigos_ocultos.pop() # pega um inimigo oculto
3     Personagens[inimigo].Visivel = True # torna o inimigo visivel
4     Personagens[inimigo].Velocidade = 30 # define a velocidade do inimigo
```

PersonagemAtingidoPisca()

A função *PersonagemAtingidoPisca()* é responsável por inverter o estado de visibilidade da instância. Quando chamada muitas vezes em intervalos curtos de tempo, cria um efeito de piscar.



```
1  # Função que faz o personagem piscar quando atingido
2  # efeito de dano na colisao do disparador com o proprio inimigo
3  def PersonagemAtingidoPisca(id):
4      Personagens[id].Visivel = not Personagens[id].Visivel
5
```

6. Configurações do jogo

A quantidade de instâncias do jogo é facilmente ajustável, permitindo personalizar o número de inimigos na tela, o limite máximo de tiros consecutivos dos inimigos, a capacidade máxima de disparos consecutivos do jogador e o número inicial de inimigos. Após uma série de testes, recomendamos os valores pré-definidos para uma experiência mais otimizada.

7. Conclusão

Ficamos muito felizes com a implementação do nosso projeto e acreditamos que conseguimos extrair bastante do que o OpenGL tem a oferecer de acordo com o que aprendemos em aula. Sentimos que aplicamos nossos conhecimentos de maneira eficaz e que evoluímos tanto nosso pensamento matemático quanto computacional ao longo do processo.