



PYKOMBAT

Relatório Final

CES22 – Programação Orientada a Objeto

Matheus Vidal de Menezes

Abril, 2019

Turma 1COMP

Prof.º Yano

Instituto Tecnológico de Aeronáutica (ITA)

São José dos Campos, São Paulo, Brasil.

matheusvidaldemenezes@gmail.com

git: vidalmatheus

I. Introdução¹

O grupo Adriano Soares Rodrigues, Matheus Vidal de Menezes e Pedro Alves de Souza Neto decidiu fazer um jogo inspirado no Mortal Kombat II 1993 (plataforma: SNES). Com essa ideia, o grupo começou a se dividir em parte gráfica (menus, animações, *lifebars*...), organização das classes a serem utilizadas (*set* de funções úteis), bem como estudar sistema de colisões em *pygame*. A partir das primeiras movimentações e colisões bem executadas, houve mais confluência dos integrantes para codificar todo o tipo de movimentação, *special moves* e animação em geral.

II. Codificação Individual: Matheus Vidal

Comecei minha codificação com a implementação do *menu.py* que cuida de todos os menus do jogo, vide Figura 1.

```
class MainMenu(Menu):
    def __init__(self, game=engine.Game()):
        clock = pygame.time.Clock()
        screen = "start"
        mainmenu = pygame.image.load('../res/Background/MainMenu01.png')
        game.getDisplay().blit(mainmenu, (0, 0))
        pygame.display.update()
        while True:
            clock.tick(10)
            for event in pygame.event.get():
                if event.type == QUIT:
                    pygame.quit()
                if event.type == pygame.KEYDOWN:
                    # carregar áudio
                    sound = engine.Sound()
                    if event.key == pygame.K_BACKSPACE:
                        pygame.quit()
                    if event.key == 13: # 13 == ENTER
                        # coloca áudio "in"
                        if screen == "start":
                            sound.setSound("start")
                            sound.play()
                            ScenarioMenu()
                        if screen == "options":
                            sound.setSound("options")
                            sound.play()
                            OptionMenu()
```

Figura 1. Trecho de código da criação do menu principal.

Em seguida, fui responsável pelas animações de combate e colisões. A grande lógica implementada no loop principal do jogo foi a do método *judge(self, scenario)*, que, após a adição dos lutadores, ele fica verificando se houve colisão entre os personagens, inclusive se vivo ou morto, e que tipo de colisão foi, vide Figura 2, e é ele que chamará os métodos de *fight(clock(), nextframe)*¹. Este contém todas as animações de movimentação, combate e de hits, vide Figura 3.

¹ O método *clock()* pega o tempo do relógio em *ms*. Caso este tempo seja maior do que o frame anterior mais um *frame_step*, o frame é atualizado na tela. Veja a o trecho de código da Figura 3.

```

if not player1.isAlive() or not player2.isAlive():
    if not player1.isAlive(): # finish player1
        player1.takeHit("dizzy")
        if (collide(player1.currentSprite(),player2.currentSprite()) or collide(player1.getProjectile().getProjectileSprite(), player2.currentSprite())) or player2.isAttacking() or collide(player2.getProjectile().getProjectileSprite(), player1.currentSprite()):
            dizzyCounter = 100 # tempo de dizzy
        if dizzyCounter >= 100:
            player1.takeHit("dead") # player1 morreu
    if not player2.isAlive(): # finish player 2
        player2.takeHit("dizzy")
        if (collide(player2.currentSprite(),player1.currentSprite()) or collide(player2.getProjectile().getProjectileSprite(), player1.currentSprite())) or player1.isAttacking() or collide(player1.getProjectile().getProjectileSprite(), player2.currentSprite()):
            dizzyCounter = 100
        if dizzyCounter >= 100:
            player2.takeHit("dead") # player2 morreu
    dizzyCounter -= 1
    player1: Fighter
elif (collide(player1.currentSprite(),player2.currentSprite())):
    # caso só encostem
    if ( (player1.isWalking() or player1.isJumping()) and (player2.isDancing() or player2.isCrouching() or player2.isAttacking()) or (player2.isWalking() or player2.isJumping()) and (player1.isDancing() or player1.isCrouching() or player1.isAttacking())):
        player1.setX(x1-6)
        if not player2.isSpecialMove(): player2.setX(x2+6)
    # caso houve soco fraco:
    if ( player1.isApunching() and (player2.isWalking() or player2.isDancing() or player2.isApunching() or player2.isAttacking()) or (player2.isApunching() and (player1.isWalking() or player1.isDancing() or player1.isApunching() or player1.isAttacking()))):
        if player1.isApunching():
            player2.takeHit("Apunching")
            specialCounter = specialLimit
        if player2.isApunching():
            player1.takeHit("Apunching")
        engine.Sound("Hit0").play()
        if hitCounter == 0: engine.Sound().roundHit()
        hitCounter = (hitCounter+1) % 5
    # caso houve soco forte:

```

Figura 2. Trecho de código que mostra brevemente a lógica por trás do loop do método judge.

Note, na Figura 2, temos o *if* mais externo, que executa animações de morte de algum lutador e o *elif* correspondente que verifica, por exemplo se houve soco fraco: *playerX.isApunching()* retorna *true* se o lutador X está fazendo o soco fraco e caso, o lutador inimigo não esteja bloqueando, o inimigo recebe hit com o método *takeHit(self, by)*. Essa foi a lógica utilizada durante todo o combate.

```

# combatMoves = [{"j","n","k","m","l","u","f"},[1,4,2,5,3,0,6]] -> jab
elif ((keyPressed(self.combat[0]) and self.end_Apunch) or ( not self.end_Apunch ) and (not self.hit) ):
    self.curr_sprite = self.spriteList[self.Apunch]
    self.Apunching = self.setState()
    self.setEndState()
    self.attacking = True
    self.end_Apunch = False
    if time > nextFrame:
        moveSprite(self.spriteList[self.Apunch], self.x, self.y, True)
        self.setSprite(self.spriteList[self.Apunch])
        changeSpriteImage(self.spriteList[self.Apunch], self.frame_Apunching)
        self.frame_Apunching = (self.frame_Apunching+self.Apunch_step) % (self.punchLimit[0]+1)
        if (self.frame_Apunching == self.punchLimit[0]-1):
            self.Apunch_step = -1
        if (self.frame_Apunching == self.punchLimit[0]):
            self.frame_Apunching = 0
            self.Apunch_step = 1
            self.end_Apunch = True
        nextFrame += 1*frame_step

```

Figura 3.

Dessa forma, minhas maiores codificações corresponderam a 100% dos arquivos *pykombat.py*, *menu.py* e cerca de 80% dos arquivos de *fighterScene.py* e *_fighter.py*.

Segue o cronograma do que majoritariamente foi feito:

Tabela 1. Agenda de atividades feitas. Semana "s" é semaninha.

#	Descrição	Arquivos Usados	Semana
0	Criação do repositório <i>github</i> , com as imagens a serem utilizadas no jogo	-	1,2
1	Criação de telas de opções e de escolha de cenário	-	1,2,3
2	Criação dos vetores de frames para cada animação do sub-zero	-	4
3	Implementação de telas (principal, opções, escolha de cenários)	pykombat.py engine.py menu.py	5,6
4	Trilha sonora e sons de seleção nas telas do jogo	engine.py menu.py	5,6
5	Implementação da cena de batalha, após escolha de cenário direto ou aleatório	fighterScene.py	6,7
6	Movimentação (<i>up, down, left, right</i>) com <i>sprites</i> , e não só com os <i>.png</i>	_fighter.py	8,s
7	Implementação da colisão com máscaras de <i>sprites</i>	fighterScene.py	8,s
8	Implementação (animação/colisão) de soco fraco, soco forte, chute fraco e chute forte (em pé e agachado)	_fighter.py fighterScene.py	s
9	Implementações das animações de <i>hits</i>	fighter.py fighterScene.py	s
10	Implementação do sistema de defesa (em pé e agachado)	_fighter.py fighterScene.py	s
11	Implementação da animação de <i>dizzy</i>	fighter.py fighterScene.py	s
12	Implementação da animação de morte normal	fighter.py fighterScene.py	s

III. Reflexão

IV. Conclusão

Dado o tempo disponível, pouco conhecimento prévio e dificuldades enfrentadas na utilização de métodos do *pygame* para se fazer o que realmente queríamos, considero um resultado muito satisfatório, tanto para a qualidade do jogo: vide a mecânica e sistema de colisões respondendo bem às teclas, mesmo com dois jogadores apertando várias teclas ao mesmo tempo; quanto para o maior aprendizado em *python* e POO.