

RELATÓRIO MINI-PROJETO

SISTEMA INTERATIVO



EQUIPE:

CAIO BARRETO
GABRIEL QUEIROZ
ERNESTO GONÇALVES
EDVYN LUIZ

DIVISÃO DE TAREFAS

CAIO BARRETO

Mecânica do Player, inimigo, mudança de cenas e mísseis teleguiados

GABRIEL QUEIROZ

Refatoração do código, mecânica do high score e balanceamento do jogo


ERNESTO GONÇALVES

Design completo do jogo e armazenamento do high score

EDVYN LUIZ

Mecânica de temperatura, layout da parte jogável e sons

BIBLIOTECAS UTILIZADAS

1. Pygame: é a biblioteca principal do projeto, onde usamos módulos internos como o Vector2D e Mixer.
 2. Os: Manipulação de caminhos dos assets
 3. Random: foi usada para randomizar a geração de itens com passar do tempo
- 

ORGANIZAÇÃO DO CÓDIGO

- Main.py

Esse arquivo possui o loop principal do jogo, loop da música principal e faz as devidas mudanças de cenas com base no valor do escopo das funções do menu.py. Além disso, ele tem o papel de fazer com o jogo resete de uma forma mais eficiente e ocupando menos espaço na memória, pois tínhamos encontrado um bug onde a cada reset ocorria uma queda de frames.

- Game.py

O game.py é o arquivo onde o jogo acontece de fato, ele possui 8 funções e um loop que é interrompido se o jogador colidir com um avião inimigo ou tomar qualquer dano após estar com um nível ínfimo de gasolina ou quase máximo de temperatura. Nesse arquivo é também onde todas as partes do jogo são interligadas como: diminuição do nível da gasolina e aumento da temperatura, movimentação dos sprites na tela do jogo, geração dos itens de uma forma aleatória, etc.

- draw(win, sprite.Group, gas_bar, temp_bar, score):

A função draw recebe como parâmetro a janela onde o jogo está sendo exibido, grupo de todos os sprites do jogo, barra de temperatura e gasolina que pertencem a classe Bar e o score do jogo. Então, basicamente essa função desenha todos os objetos e a pontuação do jogador na tela.

- check_collision(sprite, Group):

Essa função checa se o sprite passado como parâmetro colidiu com o grupo também passado como parâmetro. Se sim, ela retorna **True**. Senão, retorna **False**.

- generate_item(class, asset, spawn_chance, Group):

Gera todos os coletáveis, aviões inimigos e mísseis por meio de um mecanismo simples: 1-Geramos um número aleatório entre 1 e 1000, se esse número for menor que a chance de spawn o item vai ser gerado.



- **move_sprites**(Group):

Move todos os sprites pertencentes ao grupo do parâmetro de acordo com a variável SCROLL_SPEED e, para deixar o jogo progressivamente mais difícil, é somado um valor (5×10^{-5}) que faz com que a velocidade aumente com o passar do tempo.

- **shoot**(enemies, bullets, *Groups):

É responsável pelos tiros dos inimigos, onde é usado o atributo self.tick da classe Enemy para fazer um cronômetro e, desse jeito, regularizar os tiros do inimigo.

- **animation**(sprite, ticks):

Ela funciona de uma forma bem parecida com a função shoot(), porém ela faz a movimentação do background do jogo onde, a cada frame, soma-se o valor 1 ao self.tick da sprite que foi passada como parâmetro e, se self.ticks for igual ao valor do parâmetro ticks, a imagem do background é rotacionada em 90°.

- **find_angle**(missiles, player):

Essa função é a atuadora principal para o funcionamento da mecânica de mísseis teleguiados. Em suma, essa função calcula o ângulo entre o vetor diretor do míssil e o vetor da distância entre o míssil e o player e adiciona ou subtrai uma certa velocidade angular a fim de tornar esse ângulo igual a 0, para assim o míssil sempre apontar para onde o avião do jogador estiver.

- **game_loop**(win):

É a função que interliga todas as outras funções já apresentadas dando efetivamente vida ao jogo.

- Objects.py

Esse arquivo contém todas as classes usadas para fazer o jogo, cada uma com suas características específicas

- Bar (cool overheating empty)

É a classe responsável por criar todas as barras do jogo, que são usadas para visualizar o nível de temperatura, gasolina, vida dos mísseis e inimigos.

- Objects (start game ...)

Gera a classe base para outras classes como Player, Enemy, Missiles e Bullets, pois possui elementos em comum que são fundamentais para a criação das outras classes.

- Plane ()

Classe do avião do jogador e suas funções particulares como: movement (movimentar o avião), shoot (tiros do avião) e move_bullets (movimenta as balas do avião)

- Enemy ()

Classe do inimigo e sua função de tiro.

- Bullets ()

Classe derivada da classe Objects, optamos por fazer uma classe a parte para ser possível movimentar a bala mais rapidamente do que os outros objetos.

- Missiles ()

Classe dos mísseis teleguiados com sua função de movimento (movimenta o míssil com uma certa velocidade e gira a sua imagem de acordo com a velocidade angular) e função de animação (basicamente fica alternando entre as imagens do míssil para dar uma sensação de movimento)

- Menus.py

Arquivo responsável por armazenar todas as cenas do jogo e efetivar a mudança entre elas

- menu(win, archive):

Desenha os assets do menu ou do high score de acordo com a escolha do jogador

- game_over(win, score):

Desenha os assets da tela de fim de jogo, mostra o score e dá as opções de sair do jogo ou resetar.

- highscore(win, archive):

Desenha na tela a pontuação máxima que o jogador conseguiu

- check_archive(names):

Checa se o arquivo highscore.txt existe na máquina do usuário. Se sim, retorna **True**. Senão, retorna **False**

- create_archive(name):

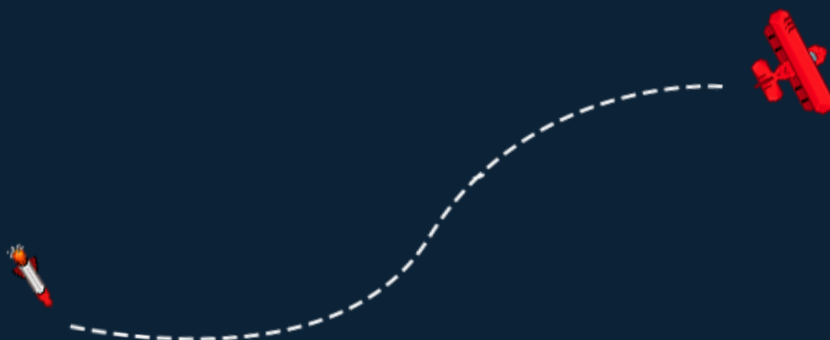
Com base na resposta da função check_archive ela cria ou não um novo arquivo chamado highscore.txt para armazenar a pontuação máxima do jogador

- update_archive(name):

Lê o arquivo highscore.txt e verifica se a pontuação atual do jogador é maior que a que está armazenada. Se sim, faz a atualização da pontuação.

- Settings.py

Armazena todas as constantes do jogo como: cores utilizadas, FPS, velocidade do jogo, chance de um item ser gerado, etc.



CONCEITOS APRENDIDOS NA DISCIPLINA

- Funções

Definitivamente esse foi o conceito aprendido na disciplina que mais foi usado na construção do jogo, sendo usada para definir movimentação dos itens, tiros, mecânica do míssil teleguiado, desenhar os sprites na tela e utilizando o seu escopo para tomada de decisões como: perder ou ganhar gasolina e temperatura, colisões entre o avião e os itens e mudança de cenas

- Loop

Esse aprendizado é fundamental para desenvolver o projeto, pois, basicamente, jogos consistem em um grande loop de eventos e nós usamos o while loop e o for loop para nos ajudar a criar algumas mecânicas e executar ações a partir dos eventos do usuário ao jogar

DESAFIOS ENFRENTADOS

No geral, não tivemos muitos problemas para a realização do projeto, seguimos uma linha de produção contínua até a finalização. Porém, alguns pontos que podem ser levados em consideração seriam: o tempo inicial para se familiarizar e aprender as funcionalidades do pygame e do github, fazer a mecânica do míssil teleguiado pois usamos conceitos aprendidos na matéria de álgebra vetorial linear para computação e isso exigiu um pensamento matemático mais elaborado do que as demais funções que tínhamos feito até então.

CONCEPÇÃO, DESIGN E EFEITOS SONOROS

A principal referência na elaboração do projeto foi o jogo arcade 1942, que também é de combate aéreo, concebido com rolagem vertical e com visão top-down. Imediatamente após a decisão de que o grupo elaboraria um sistema interativo, veio também a escolha por produzir as próprias imagens do jogo. Ou, ao menos, utilizar a menor quantidade de imagens externas possíveis.

O nome do jogo, Aero (com o “o” sendo formado por dois parênteses em alusão as funções em Python) foi escolhido por acaso como nome provisório dos arquivos dos primeiros esboços e acabou se tornando o nome do projeto. Foi definido, também sem maiores justificativas, que o jogo usaria a tela na proporção 600x750. Assim, todas as imagens foram pensadas respeitando as dimensões da tela, para que, quando fossem ampliadas, não sofressem distorções e perda de legibilidade.

Ao total, foram produzidas 28 peças gráficas autorais (menus, pequenas animações, coletáveis, etc.) Ademais, foram editadas outras 4 imagens de terceiros, nomeadamente: o ícone para gasolina sofreu alterações de textura; o foguete que persegue o avião foi animado; os projéteis que o avião dispara foram alterados e o avião inimigo foi bastante modificado. As referências para as imagens utilizadas constam no fim do relatório.

Para o desenvolvimento das imagens, foram utilizadas quatro diferentes ferramentas de edição. Os primeiros esboços da visualização do jogo em andamento, bem como o primeiro esboço do poster principal do menu inicial foram feitos no Paint e posteriormente editados com Pixlr Editor.

Passada a fase inicial, conforme a demanda de imagens aumentava, ficou inviável seguir utilizando o Paint. Passamos, então, a produzir as imagens através do Piskel e do GraphicsGale. Os efeitos finais, correções de saturação, granulação, bem como algumas tipografias foram também editados com o Pixlr Editor.

O tema principal do jogo foi extraído da música Abaddon's Bolero, do grupo Emerson, Lake and Palmer (BMG Rights Management (UK) Ltd). Como a música acontece quase que inteiramente em fade in, tivemos que normalizar o áudio para que não ficassem uma diferença de altura tão grande quando a música entrasse em loop.

Referências das imagens:

Avião inimigo: <https://br.pinterest.com/pin/355291858075108680/>

Foguete: <https://pixlr.com/stock/details/1001469285-pixel-art-rocket-missile/>

Projétil: <https://www.reddit.com/r/PixelArt/>

Gasolina: <http://pixelartmaker.com/art/ff3703de0f975f1>