Fundação Getúlio Vargas
Escola de Matemática Aplicada
Linguagens de Programação

A Fuga de Peixonauta e Careca de Bangu

Alunos:
- Gabriela Barbosa Souza
- Helora Kelly Tavares Matias
- Matheus Mendes de Assunção

Professor:
- Matheus Werner

Dezembro
2024

Conteúdo

1 Introdução

O objetivo deste trabalho foi desenvolver um jogo utilizando a biblioteca Pygame em Python. Esse jogo contém movimentação do jogador, interação com inimigos e pontuação. O projeto foi realizado de forma colaborativa, com a aplicação de conceitos de orientação a objetos, documentação, testes unitários e controle de versão usando GitHub.

2 Como Jogar?

A Fuga de Careca e Peixonalta de Bangu é um jogo cooperativo, isto é, os jogadores trabalham em equipe para vencer o jogo. O objetivo é guiar os personagens Careca e Peixonalta em uma fuga emocionante de uma prisão, enfrentando desafios estratégicos e inimigos ao longo do caminho.

O jogo foi projetado para dois jogadores, de modo que cada jogador controle um personagem.

Careca: Movimentação com as teclas **W,A,S,D**
Peixonauta: Movimentação com as teclas de seta $\uparrow$, $\leftarrow$, $\downarrow$, $\rightarrow$.

Ao iniciar o jogo, cada personagem aparece em uma área diferente do mapa. Os jogadores devem explorar o mapa, evitando seus inimigos e planejando estratégias de caminhos para coletar as chaves.
O jogo termina quando os dois escapam da prisão e, vencendo o jogo, ou quando um dos personagens colide com um inimigo; nesse caso, perdendo o jogo.

3 Divisão de Tarefas

A equipe dividiu as tarefas da seguinte maneira:

- **Gabriela:** Produção das telas de menu e créditos, adição de música às telas e testes unitários.
- **Helora:** Produção do cenário e das sprites do jogo. Além disso, fez as animações e movimentos, fez o arquivo `utils.py` e fez ajustes finais ao trabalho.
- **Matheus:** Produção da estrutura do mapa, implementação dos inimigos, movimentos e colisões. Fez os arquivos *classes.py* e *game.py*.

4 Aplicação da Orientação a Objetos

No desenvolvimento do jogo, foi utilizada a Programação Orientada a Objetos para organizar o código de forma legível e eficiente, organizando o código em módulos, o que facilitou a reutilização de componentes.

Cada elemento do jogo, como botões, telas e música, foi modelado como uma classe. As classes definem características e ações, que representam as propriedades e comportamentos dos objetos do jogo. Por exemplo, a classe Button foi criada para representar um botão na interface do usuário. Ela define atributos como posição, tamanho, texto, cor e verifica se ele foi clicado. Isso permite que cada botão seja tratado de forma independente, facilitando a criação de novos botões ou a modificação dos existentes sem afetar outras partes do código. A classe Display gerencia a exibição das telas do jogo e troca entre elas conforme o estado do jogo muda (de menu para créditos, por exemplo). Cada tela é tratada como um objeto distinto, o que ajuda a manter o código modular.
Essa estrutura modular contribui para a clareza do código, facilita a detecção e correção de erros, e permite que novos recursos sejam implementados de forma simples.

5 Funções Implementadas no Jogo

Abaixo estão detalhadas as funções das classes implementadas:

5.1 Classe `Player`

- **função** `__init__`: Define a posição inicial do jogador, o tamanho da imagem e outras variáveis importantes, como a velocidade de movimento e a gravidade
- **função** `jump`: Esta função chamada quando o jogador aperta a tecla para pular. Ela altera a velocidade vertical do jogador, fazendo com que o personagem suba na tela. A função também define a variável `is_jumping` como `True`, indicando que o jogador está no ar e não pode pular novamente até tocar o chão.
- **função** `apply_gravity`: Adiciona a gravidade ao jogo. Ela verifica se o jogador está no chão ou colidiu com algum obstáculo, para garantir que ele não continue caindo indefinidamente.
- **função** `horizontal_movement_collision`: Trata da colisão do jogador com obstáculos enquanto ele se move horizontalmente para evitar que ele passe por cima deles, corrigindo sua movimentação e evitando que o jogador "atravesse" as paredes.
- **função** `move`: Controla o movimento horizontal do jogador. Essa função chama a `horizontal_movement_collision` para garantir que o jogador não passe por obstáculos. O movimento é calculado com base na direção que o jogador escolheu.

- **função** `update`: Essa função atualiza o estado do jogador a cada iteração do jogo. Ela chama a função `vertical_movement_collision` para lidar com as colisões verticais.
- **função** `check_death`: Verifica se o jogador morreu. Isso acontece caso ele entre em contato com um inimigo fatal para ele. Se o jogador morrer, a função retorna `True`, sinalizando que o jogo precisa reiniciar.

5.2 Classe `Inimigo`

- **função** `mode`: Controla o movimento horizontal dos inimigos. Ela verifica se o inimigo colidiu com algo e, se isso acontecer, ajusta a direção do movimento para que ele não atravesse as paredes. O inimigo pode mudar de direção caso colida com algo, permitindo que ele "caminhe"ao redor do mapa de forma fluida.
- **função** `gravidade`: Essa função aplica a gravidade aos inimigos, assim como a função `apply_gravity` no jogador.
- **função** `pular`: A função `pular` permite que o inimigo execute saltos. Ela funciona de maneira semelhante ao pulo do jogador.
- **função** `update`: Atualiza o comportamento do inimigo a cada quadro do jogo. Ela verifica a posição do inimigo em relação ao jogador (Careca ou Peixonalta) e, com base nisso, faz o inimigo se mover em direção ao jogador mais próximo.
- **função** `draw`: A função draw é responsável por desenhar o inimigo na tela. Ela garante que ele apareça nas coordenadas corretas.

5.3 Classe `Chave`

- **função** `update`: Verifica se o jogador pegou a chave. Quando o jogador colide com a chave, a função marca a chave como "pegada"e a remove da tela.
- **função** `draw`: Desenha a chave na tela nas coordenadas corretas, se ela ainda não tiver sido pega.

6 Testes Unitários

6.1 O Que São Testes Unitários

Testes unitários são uma prática fundamental em desenvolvimento de software que ajudam a garantir que pequenas partes do código, conhecidas como unidades (funções ou métodos), funcionem corretamente. Eles verificam se uma função ou método específico retorna os resultados esperados para dados de entrada pré-definidos.

6.2 Para Que Servem

Verificação de Funcionalidade: Asseguram que as funções e métodos funcionam como esperado.

Detecção de Erros: Ajudam a detectar e corrigir erros no código antes que o software seja entregue ou implantado.

Manutenção do Código: Facilitam a manutenção e refatoração do código, garantindo que as mudanças não quebrem a funcionalidade existente.

Documentação: Servem como uma forma de documentação ao descrever como as funções devem se comportar.

6.3 Como Eles Funcionam

Testes unitários são geralmente automatizados e escritos usando frameworks de teste específicos, como o unittest no Python. Aqui está um exemplo de como um teste unitário funciona:

Configuração Inicial (setUp): Inicializa as condições de teste. Por exemplo, criar instâncias de objetos necessários.

Execução do Teste: Chama a função ou método a ser testado com entradas específicas.

Verificação (assert): Verifica se o resultado retornado é o esperado.

Finalização (tearDown): Limpa ou reseta qualquer estado alterado pelo teste.

6.4 Como Rodar os Testes

Para rodar os testes unitários que criamos, siga os passos abaixo:

Instale as Dependências: Certifique-se de que o pygame está instalado.

Salve os Arquivos de Teste: Cada script de teste (por exemplo, `test_chave.py`, `test_player.py`) deve estar salvo em seu diretório de projeto.

Execute os Testes: Abra o terminal, navegue até o diretório onde os scripts de teste estão salvos e execute o comando:

```sh
python -m unittest discover
```
Isso vai procurar e executar todos os arquivos de teste no diretório atual.

6.5 Explicação dos Testes Específicos

Teste da Classe Chave

python class TestChave(unittest.TestCase):

Testa a inicialização, atualização, detecção de colisão e desenho na tela.
Este teste verifica se a chave é criada corretamente, se ela é atualizada, se detecta quando um jogador a pega e se é desenhada na tela.

Teste da Classe Player

python class TestPlayer(unittest.TestCase):
Testa a inicialização, movimento, pulo, aplicação de gravidade e verificação de morte.
Este teste garante que o player é inicializado, se move corretamente, pula, aplica a gravidade e verifica se morre ao colidir com obstáculos.

Teste da Classe Inimigo

python class TestInimigo(unittest.TestCase):

Testa a inicialização, movimento, aplicação de gravidade, pulo e verificação de morte.

Este teste assegura que o inimigo é inicializado corretamente, move-se, aplica a gravidade, pula e verifica se mata um jogador ao colidir.

Teste das Funções animar e carregar_sprites

python class TestGameFunctions(unittest.TestCase):

Testa a animação dos sprites e o carregamento das spritesheets.

Este teste verifica se a função de animação atualiza corretamente os frames e se a função de carregamento de sprites divide a spritesheet corretamente.

6.6 Benefícios dos Testes

Confiabilidade: Aumentam a confiança de que o código funciona como esperado.

Facilidade de Manutenção: Ajudam a manter e refatorar o código sem medo de quebrar funcionalidades existentes.

Automatização: Permitem a execução automatizada, economizando tempo e esforço.

Incluir testes unitários é uma prática de desenvolvimento ágil e robusta, proporcionando uma base sólida para o desenvolvimento contínuo do projeto. Esses testes funcionam como um escudo protetor contra bugs e regressões.

7 Desafios Encontrados

Durante o desenvolvimento do jogo, o desafio encontrado foi o ajuste de colisões e detecção de interações no jogo.

8 Conclusão

O trabalho foi bem-sucedido no desenvolvimento do jogo, atendendo a todos os requisitos propostos no enunciado. A aplicação dos conceitos de orientação a objetos e testes unitários contribuiu para a criação de um código robusto e funcional. O controle de versão no GitHub garantiu uma boa colaboração entre os membros do grupo, e o jogo foi finalizado com sucesso.