Relatório do Projeto: Jogo da Cobra em Python

Introdução

Este projeto consiste na implementação de um jogo da cobra (Snake) utilizando a linguagem Python e a biblioteca Pygame. O jogo segue paradigmas de programação funcional, incorporando monads e outras técnicas funcionais, e apresenta diversas funcionalidades típicas de jogos, como pontuação, temporizador e obstáculos.

Funcionalidades do Jogo

Requisitos Funcionais

Movimento da Cobra:

- **Funcionalidade:** A cobra se move automaticamente em uma direção. O jogador pode alterar a direção da cobra usando as teclas direcionais do teclado.
- **Implementação:** A movimentação é controlada através de eventos de teclado capturados pelo Pygame, que atualizam a direção da cobra em cada frame.

Crescimento da Cobra:

- **Funcionalidade:** Quando a cobra come a comida, seu comprimento aumenta em uma unidade. A pontuação do jogador também aumenta a cada comida consumida.
- **Implementação:** A detecção de colisão entre a cobra e a comida é feita verificando a coincidência das posições. Quando a colisão é detectada, a cobra cresce e a pontuação é incrementada.

Geração de Obstáculos:

- **Funcionalidade:** Obstáculos são gerados aleatoriamente no mapa a cada início de jogo. A posição dos obstáculos é garantida para não coincidir com a posição inicial da cobra ou da comida.
- **Implementação:** A geração de obstáculos utiliza funções aleatórias para determinar posições válidas, assegurando que não coincidam com a cobra ou a comida.

Colisões:

- **Funcionalidade:** O jogo verifica colisões da cobra com as bordas da janela, com ela mesma e com os obstáculos. Se uma colisão ocorrer, o jogo termina.

- **Implementação:** A verificação é feita a cada frame, comparando a posição da cabeça da cobra com os limites da janela e com as posições de seu corpo e obstáculos.

Temporizador:

- **Funcionalidade:** O jogo tem um temporizador decrescente que começa em 60 segundos. Quando o tempo chega a zero, o jogo termina.
- **Implementação:** O temporizador é decrementado a cada segundo usando a função `pygame.time.get_ticks()` para controlar o tempo decorrido.

Pontuação e Temporizador na Tela:

- **Funcionalidade:** A pontuação atual do jogador e o tempo restante são exibidos na tela.
- **Implementação:** A função `pygame.font.Font` é utilizada para renderizar o texto da pontuação e do temporizador na tela.

Menu Inicial:

- **Funcionalidade:** Uma tela de menu é exibida no início do jogo, apresentando o nome do jogo.
- **Implementação:** O menu inicial é desenhado utilizando funções de desenho do Pygame e controlado por eventos de teclado para iniciar o jogo.

Requisitos Não Funcionais

Programação Funcional Utilizada:

- **Funções Lambda:** Usadas para definir operações curtas e concisas, como o movimento da cobra. Exemplo: `move_snake_lambda`.
- **Funções de Alta Ordem:** Funções que recebem outras funções como argumentos ou retornam outras funções. Exemplo: `move_snake_lambda`.
- **List Comprehensions:** Usadas para gerar listas de maneira concisa e eficiente. Exemplo: geração de listas de posições válidas para a comida.
- **Map, Filter e Reduce:**
  - `map`: Utilizado na geração da posição da comida.
  - `filter`: Utilizado na verificação de colisões da cobra com ela mesma.
  - `reduce`: Utilizado na verificação de colisões da cobra com obstáculos e bordas.
- **Funções Puras:** Funções que não têm efeitos colaterais e sempre retornam o mesmo resultado para os mesmos argumentos.

- **Currying:** Transformação de uma função que toma múltiplos argumentos em uma cadeia de funções que tomam um único argumento.

Monads:

- **Monad Maybe:** Classe Maybe com subclasses Just e Nothing, permitindo encadear operações que podem falhar sem usar exceções.
- **Aplicação do Maybe:** A função `generate_food` retorna um monad Maybe, encapsulando a posição da comida ou Nothing se não for possível gerar uma posição válida.

Regras do Jogo

Movimento:

- A cobra se move automaticamente em intervalos regulares, determinados pelo FPS (frames por segundo). O jogador pode mudar a direção da cobra usando as teclas direcionais.

Crescimento e Pontuação:

- A cobra cresce e a pontuação aumenta quando a cabeça da cobra coincide com a posição da comida.

Colisões:

- Colisões com a borda da janela, com o próprio corpo ou com obstáculos resultam no fim do jogo.

Temporizador:

- O tempo restante é decrementado a cada segundo. O jogo termina quando o tempo chega a zero.

Conclusão

Este projeto demonstra como técnicas de programação funcional e monads podem ser aplicadas para desenvolver um jogo clássico como o Snake. A utilização de funções puras, funções de alta ordem, lambdas, e o monad Maybe contribuem para um código mais modular, robusto e fácil de manter. A abordagem funcional permite a manipulação eficiente de estados e a composição de funções de forma clara e previsível. O jogo implementa várias funcionalidades essenciais, proporcionando uma experiência completa ao jogador.

Exemplos de Implementação Funcional

Função lambda de alta ordem:

- `move_snake_lambda` é uma função lambda que retorna a nova posição da cobra.

Função lambda recursiva:

- Não diretamente possível em Python sem um nome de variável, mas a ideia de repetição está presente na função `generate_food` que utiliza `map` e `iter`.

Função lambda utilizando currying:

- Implicitamente utilizado em funções lambda dentro de comprehensions e outras funções.

List Comprehension dentro do escopo de uma Lambda:

- Utilizado na função `generate_food`.

Dicionário dentro do escopo de uma função lambda:

- A função `move_snake_lambda` retorna um dicionário.

Uso de map, filter, e reduce:

- `map`: Utilizado em `generate_food` para gerar possíveis posições da comida.
- `filter`: Utilizado em `check_collision` para verificar colisões da cobra com seu próprio corpo.
- `reduce`: Utilizado em `check_collision` para determinar se houve uma colisão.