**1. Regras do Jogo**

O jogo "Caça ao Monstro" envolve um jogador tentando capturar um monstro que se move por uma grade. O jogador faz tentativas adivinhando as coordenadas do monstro e o monstro se move aleatoriamente após cada tentativa. O objetivo do jogador é capturar o monstro antes que ele "escape" ou alcance o limite de rodadas permitidas.

**Regras detalhadas:**

* O jogo acontece em uma **grade de tamanho fixo**, como uma 5x5.
* O jogador e o monstro começam em posições aleatórias na grade.
* O jogador não sabe onde o monstro está no início, mas faz **tentativas de capturá-lo** adivinhando as coordenadas.
* Após cada tentativa incorreta, o monstro **se move aleatoriamente** para uma nova posição válida na grade.
* O jogador tem um **número limitado de tentativas** antes que o monstro "escape". O limite pode ser configurado, por exemplo, como 10 turnos.
* O monstro também pode "escapar" se atingir o **limite de tentativas** sem ser capturado, ou se houver uma condição especial, como alcançar um canto ou borda da grade (opcional).
* **Condições de vitória:**
  + O jogador vence se adivinhar corretamente as coordenadas e capturar o monstro.
* **Condições de derrota:**
  + O jogador perde se o número de tentativas esgotar e o monstro não for capturado.
  + Se o monstro "escapar" (se implementarmos essa mecânica).

**2. Mecânicas do Jogo**

Agora que temos as regras estabelecidas, vamos detalhar as mecânicas que regem o funcionamento do jogo.

**2.1 Movimentação do Jogador**

* O jogador faz uma tentativa a cada turno, **escolhendo coordenadas** na grade (ex.: (2, 3)).
* A entrada do jogador será validada para garantir que ele forneça **coordenadas válidas** dentro dos limites da grade (5x5 ou outro tamanho definido).
* Se o jogador adivinhar corretamente a posição do monstro, ele vence.
* Após cada tentativa, o jogo dará **feedback** sobre o resultado:
  + Se o jogador errou, pode informar algo como "O monstro não está nesta posição".
  + Opcionalmente, podemos dar **dicas de proximidade** (ex.: "Você está perto", "O monstro está longe").

**2.2 Movimentação do Monstro**

* O monstro se move **aleatoriamente** para uma nova posição a cada turno, mas nunca sai dos limites da grade.
* A movimentação do monstro será baseada em **coordenadas aleatórias** geradas pelo jogo após a tentativa do jogador.
* O monstro não pode se mover para a posição atual do jogador, mas pode voltar para uma posição que ele já ocupou antes.
* O movimento do monstro é oculto do jogador (não mostraremos onde ele está após se mover).

**2.3 Loop de Jogo**

* O jogo segue um loop de turnos:
  1. O jogador escolhe uma coordenada para tentar capturar o monstro.
  2. O jogo verifica se a tentativa foi bem-sucedida.
  3. Se o jogador errar, o monstro se move aleatoriamente.
  4. O loop continua até que o jogador vença (capturando o monstro) ou perca (número de tentativas esgotadas ou o monstro escapar).

**2.4 Condições de Vitória/Derrota**

* **Vitória:** O jogador vence ao capturar o monstro (acertando suas coordenadas).
* **Derrota:** O jogador perde se:
  + O número de tentativas chega ao limite sem capturar o monstro.
  + O monstro escapa (opcional, se implementarmos uma mecânica de escape).

**3. Principais Funcionalidades a Implementar**

Vamos dividir as funcionalidades entre as classes que serão implementadas.

**3.1 Classe Grid**

* Responsável por:
  + **Gerar a grade** de tamanho fixo (ex.: 5x5).
  + Manter o controle das **posições** do jogador e do monstro.
  + (Opcional) Exibir uma visualização simples da grade (apenas para desenvolvimento e debugging).

**3.2 Classe Player**

* Armazenar as informações do jogador, como sua **posição atual**.
* Receber **entradas** do jogador (coordenadas para capturar o monstro).
* Validar se as coordenadas fornecidas são **válidas** (dentro da grade).

**3.3 Classe Monster**

* Armazenar a **posição atual** do monstro.
* Implementar a lógica de **movimentação aleatória** do monstro após cada tentativa do jogador.
* Garantir que o monstro sempre se mova para uma **posição válida** dentro dos limites da grade.

**3.4 Classe Game (arquivo game.py)**

* Coordenar a **execução do jogo**:
  + Inicializar o tabuleiro (classe Grid) e as posições do jogador e do monstro.
  + Controlar o **loop do jogo** (receber tentativas do jogador, mover o monstro, verificar condições de vitória ou derrota).
  + Gerenciar o **feedback** ao jogador (informar se ele está perto ou longe do monstro).
* Implementar as **condições de término do jogo** (vitória ou derrota).

**3.5 Utils (utils.py)**

* Funções auxiliares para o jogo, como:
  + **Gerar coordenadas aleatórias**.
  + Validar se as coordenadas estão **dentro da grade**.
  + (Opcional) Funções para calcular a **distância** entre o jogador e o monstro, para dicas de proximidade.

**4. Planejamento da Implementação**

Com base nas funcionalidades descritas, podemos seguir o seguinte plano para a implementação:

1. **Configuração inicial do projeto**:
   * Criar a estrutura de pastas e arquivos conforme descrito.
   * Configurar o ambiente com git, .gitignore, README.md, requirements.txt, e setup.py.
2. **Implementação da Lógica do Jogo**:
   * **Classe Grid**: Criação da grade e métodos para atualizá-la.
   * **Classe Player**: Implementação de entradas do jogador e validação de coordenadas.
   * **Classe Monster**: Movimentação aleatória e armazenamento da posição do monstro.
   * **Classe Game**: Integração de tudo no loop principal.
3. **Refinamentos e Testes**:
   * Adicionar mais funcionalidades conforme necessário, como contador de turnos, dicas de proximidade, e lógica de escape do monstro.

**5. Dependências**

* O projeto pode utilizar apenas bibliotecas nativas do Python como random para gerar números aleatórios e sys para capturar exceções de entrada do jogador.
* Se houver necessidade de adicionar bibliotecas externas, elas serão listadas em requirements.txt.