**1. Principais Classes**

O jogo será dividido em várias classes, onde cada uma terá um papel específico. Aqui estão as classes que identificamos para o projeto:

**1.1 Classe Grid (grid.py)**

Essa classe será responsável por criar e gerenciar o **tabuleiro** onde o jogo acontece.

**Responsabilidades:**

* **Gerar a grade** de um determinado tamanho (ex.: 5x5).
* **Atualizar o tabuleiro** com as posições do jogador e do monstro.
* (Opcional) Visualizar ou depurar o estado atual da grade (por exemplo, exibir as posições do jogador e do monstro para debugging).

**Principais Funções**:

* \_\_init\_\_(self, size: int): Inicializa a grade de tamanho size x size.
* is\_valid\_position(self, x: int, y: int) -> bool: Verifica se as coordenadas fornecidas estão dentro dos limites da grade.
* update\_positions(self, player\_pos: tuple, monster\_pos: tuple): Atualiza as posições do jogador e do monstro na grade.
* (Opcional) display(self): Mostra a grade com as posições do jogador e do monstro para fins de debugging.

**1.2 Classe Player (player.py)**

A classe Player será responsável por gerenciar a interação do jogador com o jogo, principalmente **registrar as tentativas** do jogador.

**Responsabilidades:**

* **Armazenar a posição** atual do jogador (se necessário).
* **Receber a entrada** do jogador (coordenadas para capturar o monstro).
* **Validar as coordenadas** fornecidas pelo jogador (verificar se estão dentro dos limites da grade).

**Principais Funções**:

* \_\_init\_\_(self): Inicializa o jogador com uma posição ou estado padrão.
* guess\_position(self) -> tuple: Solicita ao jogador uma tentativa (entrada de coordenadas).
* validate\_guess(self, x: int, y: int, grid\_size: int) -> bool: Valida se a coordenada fornecida está dentro dos limites da grade.

**1.3 Classe Monster (monster.py)**

A classe Monster gerencia a **posição e o movimento** do monstro, que se move aleatoriamente pela grade.

**Responsabilidades:**

* Armazenar a **posição atual** do monstro.
* **Gerar movimentações aleatórias** após cada turno do jogador.

**Principais Funções**:

* \_\_init\_\_(self, grid\_size: int): Inicializa o monstro em uma posição aleatória dentro da grade.
* move\_random(self, grid\_size: int) -> tuple: Gera uma nova posição aleatória para o monstro dentro dos limites da grade.

**1.4 Classe Game (game.py)**

Essa é a classe principal que controlará o **loop de jogo** e a **lógica central**.

**Responsabilidades:**

* **Inicializar** a grade, o jogador e o monstro.
* Controlar o **fluxo do jogo**, incluindo o loop de tentativas do jogador e movimentação do monstro.
* Verificar **condições de vitória ou derrota**.
* **Atualizar** as posições na grade após cada turno.
* Dar **feedback** ao jogador sobre o estado do jogo.

**Principais Funções**:

* \_\_init\_\_(self): Inicializa os componentes do jogo (grade, jogador, monstro).
* play(self): Controla o loop do jogo.
* check\_victory(self, player\_pos: tuple, monster\_pos: tuple) -> bool: Verifica se o jogador capturou o monstro.
* is\_game\_over(self) -> bool: Determina se o jogo terminou (por vitória ou derrota).
* give\_feedback(self, player\_pos: tuple, monster\_pos: tuple): (Opcional) Dá dicas ao jogador sobre a proximidade do monstro.

**1.5 Funções Auxiliares (utils.py)**

Essas funções podem ser agrupadas no arquivo utils.py para ajudar nas operações secundárias do jogo.

**Responsabilidades:**

* **Validação de coordenadas** e outras tarefas auxiliares que não são responsabilidade direta das classes principais.

**Principais Funções**:

* generate\_random\_position(grid\_size: int) -> tuple: Gera uma coordenada (x, y) aleatória dentro da grade.
* calculate\_distance(player\_pos: tuple, monster\_pos: tuple) -> int: (Opcional) Calcula a distância entre o jogador e o monstro (pode ser usada para dicas de proximidade).

**2. Exemplo de Interação entre as Classes**

Aqui está um exemplo de como essas classes interagem no ciclo principal do jogo:

1. O jogo começa com a classe Game, que inicializa a **grade** (Grid), o **jogador** (Player) e o **monstro** (Monster).
2. O **loop principal** do jogo começa, onde:
   * O jogador faz uma **tentativa** fornecendo coordenadas.
   * O jogo verifica se a tentativa foi **correta** ou não.
   * Se a tentativa falhar, o monstro se **move aleatoriamente** para uma nova posição.
3. O loop continua até que o jogador capture o monstro (vitória) ou o número de tentativas se esgote (derrota).

**3. Estrutura Detalhada das Funções**

Aqui está uma visão detalhada das funções mais importantes:

**3.1 Classe Grid (grid.py)**

python

Copy code

class Grid:

def \_\_init\_\_(self, size: int):

"""Inicializa a grade de tamanho size x size."""

self.size = size

self.grid = [[None for \_ in range(size)] for \_ in range(size)]

def is\_valid\_position(self, x: int, y: int) -> bool:

"""Verifica se a posição (x, y) está dentro dos limites da grade."""

return 0 <= x < self.size and 0 <= y < self.size

def update\_positions(self, player\_pos: tuple, monster\_pos: tuple):

"""Atualiza as posições do jogador e do monstro na grade."""

# Essa função pode ser expandida se quisermos "desenhar" a grade

pass

def display(self):

"""Exibe a grade (opcional, para debugging)."""

pass

**3.2 Classe Player (player.py)**

python

Copy code

class Player:

def \_\_init\_\_(self):

"""Inicializa o jogador sem posição (vai adivinhar as posições)."""

pass

def guess\_position(self) -> tuple:

"""Recebe as coordenadas do jogador."""

x = int(input("Digite a linha: "))

y = int(input("Digite a coluna: "))

return (x, y)

def validate\_guess(self, x: int, y: int, grid\_size: int) -> bool:

"""Valida se a tentativa está dentro dos limites da grade."""

return 0 <= x < grid\_size and 0 <= y < grid\_size

**3.3 Classe Monster (monster.py)**

python

Copy code

import random

class Monster:

def \_\_init\_\_(self, grid\_size: int):

"""Inicializa o monstro em uma posição aleatória na grade."""

self.position = (random.randint(0, grid\_size-1), random.randint(0, grid\_size-1))

def move\_random(self, grid\_size: int) -> tuple:

"""Move o monstro para uma nova posição aleatória na grade."""

self.position = (random.randint(0, grid\_size-1), random.randint(0, grid\_size-1))

return self.position

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

python

Copy code

class Game:

def \_\_init\_\_(self):

"""Inicializa os componentes do jogo."""

self.grid\_size = 5

self.grid = Grid(self.grid\_size)

self.player = Player()

self.monster = Monster(self.grid\_size)

self.turns = 10 # Número máximo de turnos/tentativas

def play(self):

"""Controla o loop principal do jogo."""

while self.turns > 0:

player\_guess = self.player.guess\_position()

if not self.grid.is\_valid\_position(\*player\_guess):

print("Coordenadas inválidas. Tente novamente.")

continue

if self.check\_victory(player\_guess, self.monster.position):

print("Você capturou o monstro!")

break

# O monstro se move

self.monster.move\_random(self.grid\_size)

self.turns -= 1

print(f"Você tem {self.turns} tentativas restantes.")

if self.turns == 0:

print("Você perdeu! O monstro escapou.")

def check\_victory(self, player\_pos: tuple, monster\_pos: tuple) -> bool:

"""Verifica se o jogador capturou o monstro."""

return player\_pos == monster\_pos

**4. Resumo**

* A classe Grid lida com o tabuleiro e a validação de posições.
* A classe Player recebe e valida as tentativas do jogador.
* A classe Monster controla a movimentação aleatória do monstro.
* A classe Game coordena todo o ciclo do jogo, verificando as condições de vitória e derrota, e controlando o loop principal.