# MC102 - Algoritmos e Programação de Computadores

Lab 10

**Data da Primeira Chance:** 12 de junho de 2023
**Peso:** 3

Aloy é uma caçadora e arqueira, que utiliza a velocidade, esperteza e agilidade para permanecer viva e proteger sua tribo, em um mundo pós-apocalíptico dominado por criaturas mecanizadas como animais robôs colossais. Ela decide descobrir seu passado, explorando o mundo e lutando contra essas máquinas. Aloy possui um dispositivo chamado Foco, que lhe permite escanear objetos e criaturas para obter informações como quanto dano já levou, a fraqueza das máquinas e quanto de dano os monstros levam ao serem acertados. Os componentes destas criaturas são complexos e delicados. Tais componentes, ou partes, costumam receber mais dano quando acertados em locais específicos (pontos críticos), além de possuírem vulnerabilidades a determinados elementos.


Fonte: Horizon Zero Dawn: Eine atemberaubende Open-World-Erfahrung auf PS4 (marc.tv)

Para provar seu valor para tribo, da qual foi exilada, Aloy deverá derrotar uma quantia **N** de máquinas, sendo que pode enfrentar até **U** (1 ≤ U ≤ N) máquinas ao mesmo tempo. Como uma caçadora experiente e veloz, ela consegue **disparar 3 flechas seguidas antes de receber dano de todas as U máquinas inimigas**. Essas flechas podem acertar pontos diferentes em partes diferentes e ter máquinas diferentes como alvo.

Em relação às máquinas, cada uma de suas partes possui um valor de dano máximo (**M**) em relação aos pontos totais de vida da máquina (**V**). O cálculo de dano (**D**) funciona da seguinte maneira:

1. Caso a flecha seja do mesmo tipo da fraqueza e ela acerte um ponto qualquer $(f_x, f_y)$, o dano causado será igual à diferença entre o dano máximo e a Distância de

Manhattan entre o ponto de crítico $(\mathbf{C_x},\mathbf{C_y})$ e o ponto acertado, ou seja: $D = M - (|(c_x - f_x)| + |(c_y - f_y)|)$

2. Caso a flecha acerte um ponto qualquer e ela não for do mesmo tipo da fraqueza, o dano será igual a D (como definido acima) dividido por 2. Você deve utilizar a divisão inteira para não ocorrerem problemas de precisão.

O Foco faz com que Aloy esteja ciente das fraquezas das peças (podendo ser todas ou nenhuma), sendo assim, ela sempre usará os tipos de flechas compatíveis, caso estejam disponíveis. Fazer flechas diferentes gasta muitos recursos, então **após o final do combate** ela vai até as coordenadas de acerto de cada flecha e as **recolhe**, a fim de manter seu estoque. Além disso, **ao final de cada combate** ela toma um remédio, que **recupera seus pontos de vida** em **até 50% dos pontos de vida máximos**, isto é, os pontos de vida de Aloy após se curar serão acrescidos de floor(0,5 × vida máxima), desde que não ultrapassem os pontos de vida máximos.

# Entrada

A entrada do programa é constituída pelas seguintes informações, em ordem:

1. Um inteiro **A**, correspondente aos pontos de vida máximos de Aloy;
2. Uma linha contendo os tipos de flechas e suas respectivas quantidades separados por espaço. Ex: `fogo 10 gelo 5 normal 8`
3. Um inteiro **N** contendo o número de monstros que Aloy enfrentará.
4. Um inteiro **U** (1 ≤ U ≤ N), referente à quantidade de máquinas que enfrentará no combate. Se 2 ≤ U, serão repetidos os pontos 5 e 6 até a quantidade **U**.
5. Para cada máquina, será dada uma linha contendo 3 inteiros, separados por espaço, contendo informações da $U_i$ máquina. São essas informações: pontos de vida ($V_i$), pontos de ataque ($P_i$) e quantidade de partes ($Q_i$).
6. Em seguida, $Q_i$ linhas contendo o nome do componente (parte do corpo), a sua fraqueza, o dano máximo e as coordenadas $\mathbf{C_x}$ e $\mathbf{C_y}$, tudo separado por vírgulas.
7. Após isso, será dada uma linha contendo as seguintes informações separadas por vírgula: um inteiro **K** (0 ≤ K < U) representando a unidade alvo, uma string **C** ($C \in P_i$) com a parte do corpo que será o alvo, uma string referente ao tipo de flecha usado e dois inteiros, as coordenadas da flecha $\mathbf{f_x}$ e $\mathbf{f_y}$. Essa entrada será dada até que os **U** inimigos tenham seus pontos de vida zerados, voltando ao ponto 4, ou até que os pontos de vida ou todas as flechas de Aloy cheguem a zero, finalizando o programa.

# Saída

A cada combate, seu programa deve exibir o **índice** do mesmo e com **quantos pontos de vida** Aloy o iniciou. Quando uma **máquina for derrotada**, você deve exibir a mensagem "Máquina $U_i$ derrotada". **Ao final de cada combate**, você deve exibir os **pontos de vida de Aloy**, antes de se curar, a **quantidade total de flechas de cada tipo utilizadas no combate**, ordenadas pela ordem que foram declaradas (ponto 2 da entrada), uma por linha.

Você deverá exibir também **quais foram os pontos críticos acertados,** conforme a ordem de declaração das máquinas e das partes (pontos 4 a 6 da entrada)**, e quantas vezes**. **As informações de crítico e da quantidade de flechas são dadas pelo Foco, se Aloy for derrotada ou não possuir nenhuma flecha, essas informações não serão mostradas.** Ao final do programa deve ser exibida uma das três mensagens a seguir, a depender do desfecho:

- “Aloy provou seu valor e voltou para sua tribo.”, caso derrote todos os monstros;
- “Aloy ficou sem flechas e recomeçará sua missão mais preparada.”, caso fique sem flechas; e
- “Aloy foi derrotada em combate e não retornará a tribo.”, caso seus pontos de vida cheguem a zero.

# Dica:

Para solucionar esse laboratório você precisará de:

- Condicionais;
- Laços de repetições;
- Funções;
- Listas;
- Dicionários;
- Tuplas; e
- Conjuntos.

Atenção na hora de modelar suas estruturas e organizar seu código:

- Para armazenar a localização das flechas e dos pontos críticos procure usar tuplas. Lembre-se: você pode utilizar uma tupla como chave de um dicionário;
- Para recuperar as chaves de um dicionário na ordem que foram inseridas, você pode utilizar o método “*.keys()*”. Ex: flechas.keys() retornaria os tipos de flechas na ordem inserida no dicionário flechas;
- Você pode ter listas de dicionários, nelas a posição do dicionário pode significar algo;
- Você pode precisar de dicionários que contenham dicionários, ou seja, um dicionário pode ser o valor para a chave de outro dicionário.
- Não existe dano negativo.

# Exemplos

## Exemplo 1:

**Entrada**

```
500
confusão 20 fogo 10 gelo 15 normal 50 perfurante 20 veneno 15
1
1
90 5 2
corpo, nenhuma, 90, 0, 0
olho, todas, 180, 0, 10
0, olho, perfurante, 0, 10
```

**Saída**

```
Combate 0, vida = 500
Máquina 0 derrotada
Vida após o combate = 500
Flechas utilizadas:
- perfurante: 1/20
Críticos acertados:
Máquina 0:
- (0, 10): 1x
Aloy provou seu valor e voltou para sua tribo.
```

## Exemplo 2:

**Entrada**

```
500
confusão 20 fogo 10 gelo 15 normal 50 perfurante 20 veneno 15
2
1
90 5 2
corpo, nenhuma, 50, 0, 0
olho, todas, 180, 0, 10
0, olho, perfurante, 0, 10
1
90 5 2
corpo, nenhuma, 50, 0, 0
olho, todas, 180, 0, 10
0, corpo, normal, 0, 0
0, corpo, normal, 0, 0
0, corpo, normal, 0, 0
0, olho, perfurante, 0, 0
```

**Saída**

```
Combate 0, vida = 500
Máquina 0 derrotada
Vida após o combate = 500
Flechas utilizadas:
- perfurante: 1/20
Críticos acertados:
Máquina 0:
- (0, 10): 1x
Combate 1, vida = 500
Máquina 0 derrotada
Vida após o combate = 495
Flechas utilizadas:
- normal: 3/50
- perfurante: 1/20
Críticos acertados:
Máquina 0:
- (0, 0): 3x
Aloy provou seu valor e voltou para sua tribo.
```

### Exemplo 3:

**Entrada**

```
500
confusão 20 fogo 10 gelo 15 normal 50 perfurante 20 veneno 15
1
1
7500 250 5
corpo, gelo, 500, 0, 40
bolsa de fogo, todas, 200, -5, -15
unidade de fogo, todas, 160, 5, 10
sparker, perfurante, 100, 5, -20
bolsa de explosivos, nenhuma, 300, 0, -10
0, corpo, gelo, 9, 1
0, corpo, gelo, 40, -40
0, corpo, gelo, 1, -15
0, sparker, perfurante, -41, -35
0, unidade de fogo, normal, 40, 32
0, corpo, gelo, 47, 1
```

**Saída**

```
Combate 0, vida = 500
Vida após o combate = 0
Aloy foi derrotada em combate e não retornará a tribo.
```

# Regras e Avaliação

**Nesse laboratório, você não pode usar bibliotecas (isto é, o comando *import*). Exceto pela biblioteca *typing* para melhorar a clareza e escrita do código e pela biblioteca *math* para fazer cálculos.**

Seu código será avaliado não apenas pelos testes do CodePost, mas também pela qualidade. Dentre os critérios subjetivos de qualidade de código analisaremos nesse laboratório: o uso de listas, tuplas, dicionários e conjuntos; o uso apropriado de funções, e de documentação; a escolha de bons nomes de funções e variáveis; a ausência de diversos trechos de código repetidos desnecessariamente. Note, porém, que essa não é uma lista exaustiva, pois outros critérios podem ser analisados dependendo do código apresentado visando mostrar ao aluno como o código poderia ser melhor.

Os casos de testes estão disponíveis através do link.

# Submissão

Você deverá submeter no CodePost, na tarefa Lab 10, um arquivo com o nome `lab10.py`. Você pode enviar arquivos adicionais caso deseje para serem incluídos por `lab10.py`. Após a correção da primeira entrega, será aberta uma tarefa Lab 10 - Segunda Chance, com prazo de entrega apropriado.