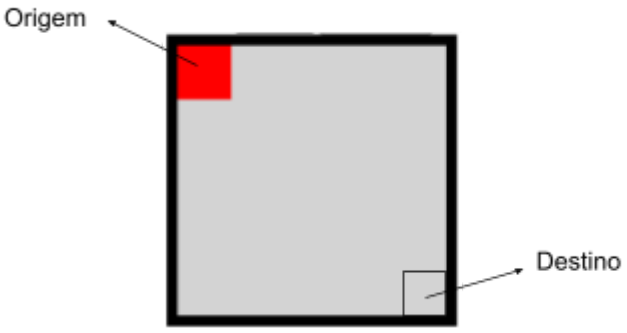
Hokama's Tower Defense (v1)

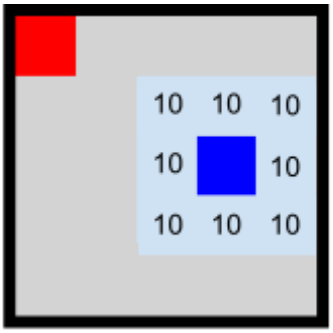
Importante:

- Em caso de plágio, fraude ou tentativa de burlar o sistema será aplicado nota 0 na disciplina aos envolvidos.
- Os alunos deverão ser capazes de explicar com detalhes os algoritmos e as implementações.
- Passar em todos os testes não é garantia de tirar a nota máxima. Sua nota ainda depende do cumprimento das especificações do trabalho, qualidade do código, clareza dos comentários, desempenho do algoritmo, boas práticas de programação e entendimento da matéria.
- Este trabalho deverá ser implementado em linguagem C. Pode ser feito em duplas.

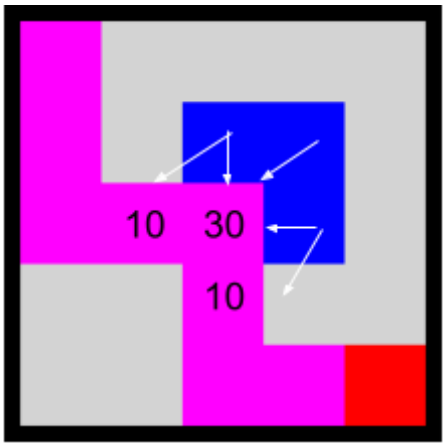
Tower Defense é uma modalidade de jogo onde um defensor posiciona torres que machucam os atacantes que passarem por perto, os atacantes por sua vez seguem caminhos de uma fonte até o destino.
No Hokama's Tower Defense o jogo se dá em um tabuleiro de n x n, o atacante sai do quadrado superior esquerdo na coordenada (0, 0) e precisa chegar no quadrado (n-1, n-1).



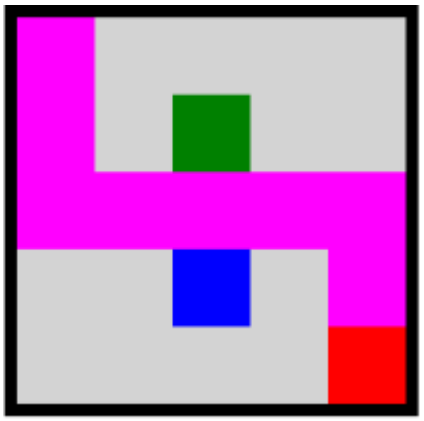
O Defensor pode colocar quantas torres quiser em qualquer posição do tabuleiro (exceções serão explicadas posteriormente) e cada torre causa 10 de dano em cada quadrado que fica adjacente à torre. No desenho abaixo, o quadrado azul representa uma torre colocada e a área azul claro representa a área de alcance da torre. Note que a torre ataca na diagonal.



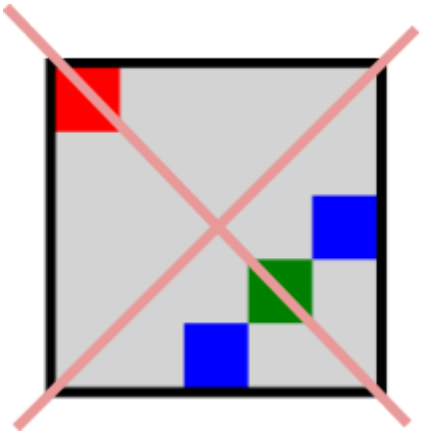
O dano das torres acumula, ou seja, em um quadrado que tiver 3 torres em quadrados adjacentes, caso o atacante passe por ele, sofrerá 30 de dano. O atacante também pode sofrer danos múltiplos de uma mesma torre, caso passe por mais de um quadrado defendido por ela. Na figura a seguir o caminho realizado pelo atacante está marcado em magenta, note que o atacante sofre 50 de dano.



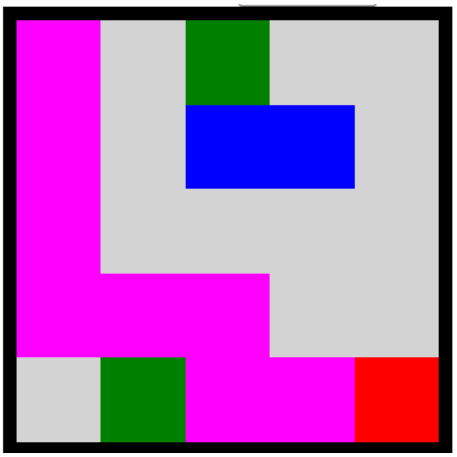
Existem Fontes de Cura que nascem no tabuleiro no início do jogo. As Fontes de Cura, funcionam como a torre, porém curam o atacante em 10 de vida por quadrado (você pode entender como um dano negativo de -10). Na figura a seguir a fonte é representada pelo quadrado verde. Note que o atacante passa sem tomar dano, pois todo o dano da torre, é anulado pela cura da Fonte.



O defensor não pode construir uma torre no mesmo quadrado de uma Fonte, mas pode construir adjacente a ela. O atacante só pode andar para cima, para baixo, para a esquerda e para a direita, não podendo ir pela diagonal. O defensor **não pode bloquear completamente a passagem do atacante**. Por exemplo, a configuração abaixo é proibida.



O atacante obviamente gostaria de tomar a menor quantidade de dano possível (se possível até tomar um dano negativo), com a condição de que ele não pode passar duas vezes pelo mesmo quadrado. No exemplo abaixo, o atacante tomou -40 de dano.



Serão 4 trabalhos envolvendo o Hokama's tower defence.

Trabalho 01: Você deverá desenvolver uma heurística rápida para o atacante encontrar um caminho em que ele tome pouco dano.

Trabalho 02: Você deverá desenvolver um algoritmo exato em que o atacante toma o menor dano possível.

Trabalho 03: Você deverá desenvolver uma heurística para o defensor escolher onde colocar suas torres. Fazendo com que o atacante tome bastante dano.

Trabalho 04: Você deverá desenvolver um algoritmo exato para o defensor posicionar suas torres de forma que o atacante tome o maior dano possível.

Você receberá uma implementação inicial que faz a leitura do tabuleiro e você deverá implementar as funções solicitadas. O programa inicial lê da entrada do sistema o tabuleiro no seguinte formato, primeiro o tamanho n do tabuleiro, e depois n linhas cada uma com n caracteres, em que 0 (zero) significa campo aberto, F significa Fonte de Cura, e T significa Torre (nos trabalhos 3 e 4 não haverá torres). Não há nenhuma marcação especial na origem e no destino do atacante, mas é esperado que essas posições estejam livres. Por exemplo a configuração da última imagem seria:

```
5
00F00
00TT0
00000
00000
0F000
```

Para o trabalho 01:
Uma solução para o trabalho 01 é um caminho formado pelas direções que o atacante deve se mover. As direções são dadas por caracteres ‘N’ norte, ‘S’ sul, ‘O’ oeste e ‘L’ leste.
Você receberá um código parcial, que faz a leitura do tabuleiro em uma matriz bidimensional de caracteres e cria um array com n * n caracteres, suficiente para qualquer caminho pelo tabuleiro. Você deverá implementar a função

```
int encontra_caminho_heuristico(char ** T, int n, char * caminho)
```

que se encontra no arquivo solver.c. Essa função recebe o tabuleiro, a dimensão dele, e um array de caracteres caminho (já pré alocado), a função devolve o tamanho do caminho encontrado. No exemplo da entrada acima, uma solução (ruim) seria o caminho com 8 movimentos:

‘S’, ‘S’, ‘S’, ‘L’, ‘L’, ‘S’, ‘L’, ‘L’

Um exemplo de solver bem fraco é fornecido em “exemplo_solver_burro.txt” simplesmente para teste. Note que é esperado que a sua solução seja MUITO melhor do que essa.

- Você poderá, se quiser, buscar literatura e códigos na internet. Entretanto, caso deseje se basear nessa fonte, você deverá compreendê-la completamente e realizar a sua própria implementação, além de explicá-la com detalhes em apresentação.
- Você pode usar bibliotecas elaboradas, conhecidas e confiáveis, com a condição de que você explique com antecedência como é feita a instalação no Ubuntu 20.04.2 LTS.
- Se você não tiver certeza se alguma coisa é permitida ou não no trabalho, não hesite em perguntar ao professor!

Para o trabalho 02:
Uma solução para o trabalho 02 é um caminho DE DANO MÍNIMO formado pelas direções que o atacante deve se mover. As direções são dadas por caracteres ‘N’ norte, ‘S’ sul, ‘O’ oeste e ‘L’ leste.
Você receberá um código parcial, que faz a leitura do tabuleiro em uma matriz bidimensional de caracteres e cria um array com n * n caracteres, suficiente para qualquer caminho pelo tabuleiro. Você deverá implementar a função

```
int encontra_caminho_exato(char ** T, int n, char * caminho)
```

que se encontra no arquivo solver.c. Essa função recebe o tabuleiro, a dimensão dele, e um array de caracteres caminho (já pré alocado), a função devolve o tamanho do caminho encontrado. No exemplo da entrada acima, uma solução ótima toma -40 de dano.

- Você poderá, se quiser, buscar literatura e códigos na internet. Entretanto, caso deseje se basear nessa fonte, você deverá compreendê-la completamente e realizar a sua própria implementação, além de explicá-la com detalhes em apresentação.

- Você pode usar bibliotecas elaboradas, conhecidas e confiáveis, com a condição de que você explique com antecedência como é feita a instalação no Ubuntu 20.04.2 LTS.
- Se você não tiver certeza se alguma coisa é permitida ou não no trabalho, não hesite em perguntar ao professor!