Descrição

Matilda é uma desenvolvedora de jogos muito habilidosa, e recentemente iniciou o maior de seus projetos: a criação de um motor de jogos. Uma ideia ambiciosa, certamente, mas muito possível com seus talentos de programação. Os jogos favoritos de Matilda são jogos com cenários destrutivos e cheios de interações dinâmicas com o mundo. Para isso ela percebe que precisará implementar alguns sistemas independetes: um módulo de renderização que mostra coisas na tela, e outro de física que seja capaz de calcular as interações entre os diferentes objetos.

Apesar de sua empolgação, Matilda nunca realmente fez um motor de jogos com as próprias mãos e resolve começar pequeno. Talvez ela possa aprender algo a partir dali e expandir cada vez mais até chegar no nível que ela quer. Foi assim que ela pediu a sua ajuda para implementar essa primeira versão. Essa versão, consiste de um sistema no qual todos os objetos são conjuntos de partículas. Existem 4 tipos diferentes de partículas e elas interagem entre si para formar um ambiente dinâmico.

O motor de jogos consiste de um loop que itera por frames. Um frame é uma representação visual do estado do jogo naquele momento e é essa representação que deve ser impressa na tela. Depois de imprimir o frame, o estado do jogo deve ser atualizado de acordo com as regras da física desse mundo (descritas abaixo).

Faça um programa em C que rode uma simulação com as regras descritas abaixo. Para isso, uma matriz de 32 linhas e 64 colunas deve ser criada, onde cada elemento é do tipo `char`. A cada frame essa matriz deve ser imprimida na tela, isso é, cada linha imprimida como uma string e ao final dela um caractere `'\n'` para indicar o final da linha. Após isso, uma função de aplicar física será executada. Essa função atualizará a matriz para ser impressa no próximo frame.

Cada caractere dessa matriz representará uma partícula que possui como posição o seu índice na matriz. Por exemplo, a partícula na posição `(10, 20)` é representada pelo caractere em `mat[20][10]` (repare que usamos `mat[y][x]`), onde `mat` é a matriz mencionada.

Existem 4 tipos de partículas representadas por caracteres: Areia (`#`), Água (`~`), Cimento (`@`) e Ar (`<espaço>`). Cada uma dessas partículas possui um comportamento diferente que poderá mudar sua posição na matriz na hora de aplicar a física.

Além disso, na entrada padrão serão fornecidos alguns valores. Na primeira linha será fornecido o número de frames que sua simulação deve calcular e imprimir. As linhas subsequentes seguirão o seguinte formato: `<frame>: <x> <y> <char>`. Onde `<frame>` é o frame no qual o caractere `<char>` deve ser posto na posição `(<x>, <y>)` da matriz. Tanto `<x>` quanto `<y>` são números inteiros. Os valores na posição `<frame>` são sempre crescentes.

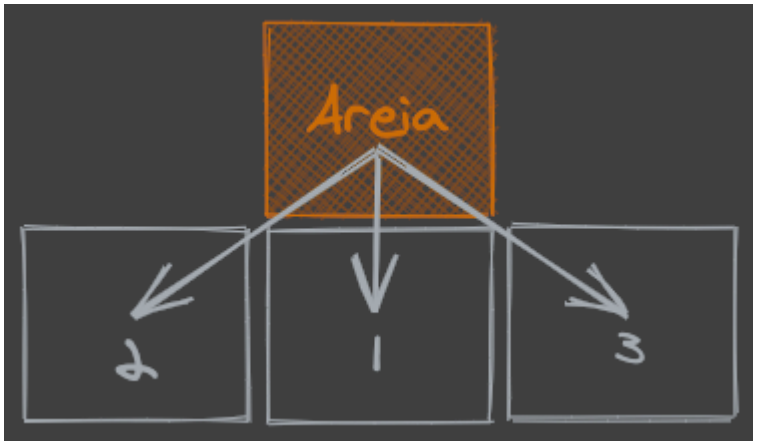
As partículas

A **partícula de cimento** é a mais simples. Ela nunca se move. Uma vez que ela for posicionada na matriz, ela nunca sairá daquela posição.

A **partícula de ar** também não faz nada. Entretanto, ela pode ser movida de lugar por outras partículas como veremos abaixo.

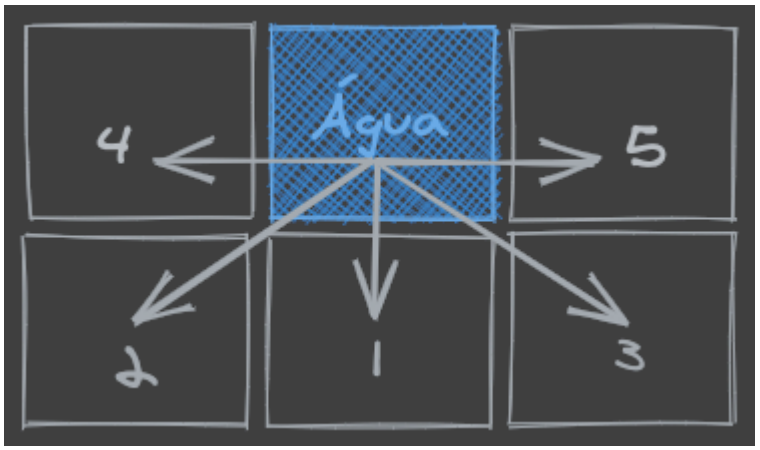
Já a **partícula de areia** sofre os efeitos da gravidade e deve seguir a algumas regras na hora de aplicar a física. Algumas posições adjacentes a essa partícula serão verificadas e, se essa posição estiver sendo preenchida por água ou ar, a areia troca de lugar com essa partícula adjacente. Isso é como dizer que a areia é mais pesada que água e ar. As posições adjacentes a serem verificadas em ordem são:

1. Posição imediatamente abaixo.
2. Posição à esquerda abaixo.
3. Posição à direita abaixo.



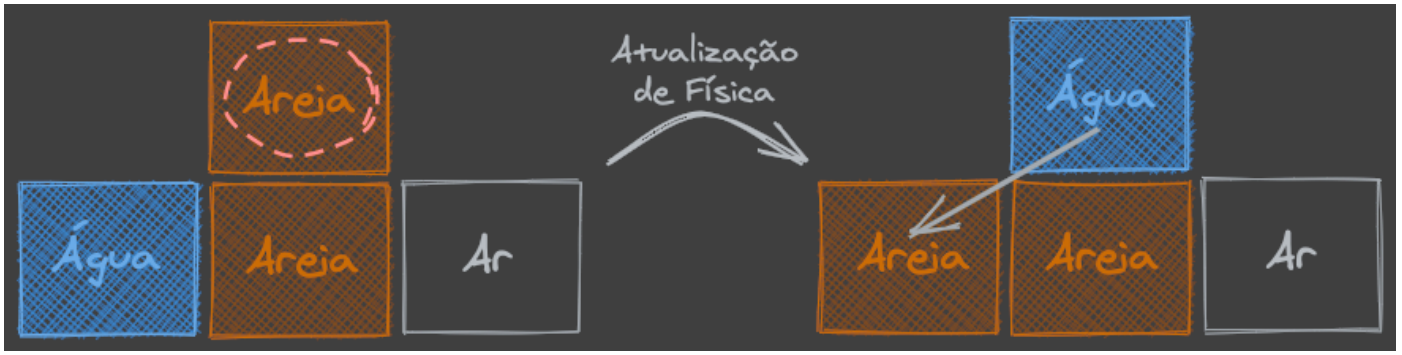
Se nenhuma das posições verificadas tiver água ou ar, a partícula não muda de lugar.

Por fim, a **partícula de água** funciona de forma muito parecida com a partícula de areia, com a diferença de que ela só pode trocar de lugar com ar. Além disso ela também verifica a lateral esquerda e depois direita para possíveis lugares para ir.

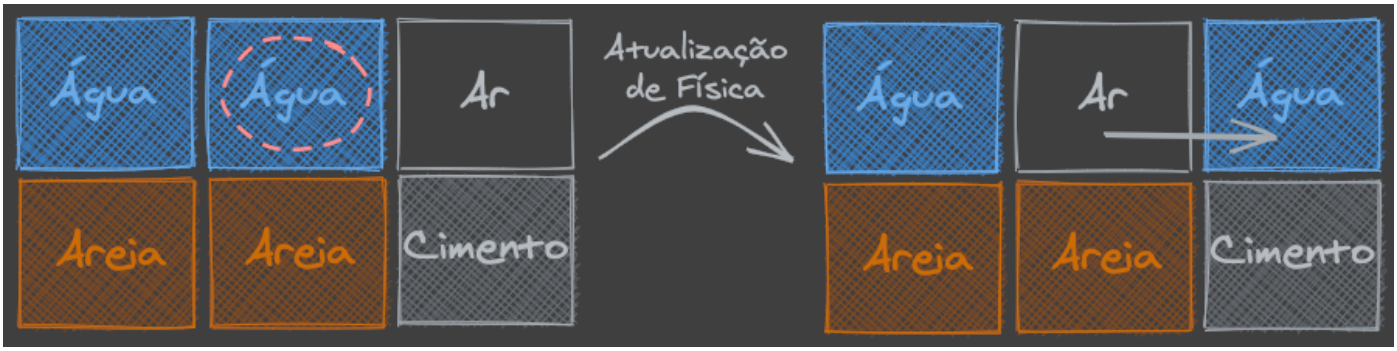


Se nenhuma das posições verificadas tiver ar, a partícula não muda de lugar.

Exemplos das regras da areia/água



Explicação: A partícula de areia ao topo está sendo atualizada agora. Primeiro, verifica a posição imediatamente abaixo que contém areia. Como areia não é água nem ar, verifica a próxima posição que é a esquerda abaixo e, como ela contém água trocamos as partículas de posição e o processo para essa partícula está terminado.



Explicação: A partícula de água circulada é a que será atualizada. Os locais adjacentes são verificados na ordem especificada, entretanto o único lugar para o qual a água pode se mover é para a direita.

Detalhes de implementação

Representando o estado da simulação através de uma matriz

O estado atual da simulação é completamente armazenado na mesma matriz que contém os dados que serão imprimidos na tela. Cada posição da matriz tem um caractere que representa a partícula naquela posição e essa é toda a informação que precisa ser armazenada.

A matriz deve ser iniciada com partículas de ar em todas as posições. E pode ser representada como uma variável global.

A função de atualização de física

No início da função uma cópia da matriz de estado deve ser criada. Durante a simulação, as atualizações devem ser feitas nessa matriz, enquanto as verificações de qual partícula se encontra em qual posição devem ser feitas com a matriz original. Isso é necessário para que não verifiquemos posições que já foram atualizadas.

Atenção: Para atualizar a matriz de cópia durante a execução da função de atualização da física, faça uma **troca** direta e simples dentro da própria matriz cópia. Não use valores diretos já que pode não haver relação entre a matriz original e a matriz de cópia em algum momento. Ou seja, é sim possível que alguns comportamentos estranhos aconteçam. Esses comportamentos, entretanto são previsíveis e não geram problemas sérios. Aqui segue um exemplo de como fazer essa troca de valores. **Utilize esse tipo de troca, caso contrário seu programa não irá funcionar corretamente**. O código abaixo é um exemplo onde consideramos que `copia` é o nome da variável com a matriz de cópia e gostaríamos de trocar a partícula da posição `(j, i)` com `(j, i + 1)`.

```
char tmp = copia[i][j];
copia[i][j] = copia[i + 1][j];
copia[i + 1][j] = tmp;
```

Isso é uma troca direta, ou seja, trocamos valores apenas utilizando os dados da própria matriz de cópia. **Esse tipo de troca é necessária para fazer seu programa funcionar corretamente**.

Percorra a matriz do topo para baixo **apenas uma vez**, da esquerda para a direita e atualize a partícula em cada posição de acordo com as regras. Por fim, coloque a atualização na matriz cópia.

Por último, ao final da função, a matriz com as atualizações (matriz cópia) substitui a matriz original, efetivamente atualizando o estado da simulação.

A função main

A função main deve consistir essencialmente de um loop que imprime a matriz de estado e em seguida chama a função de aplicar a física. Esse loop deve rodar por um determinado número de iterações fornecido na entrada padrão.

Espaços fora da matriz

Quando estiver rodando a simulação, pode ser que alguma partícula tente verificar um vizinho que não é um índice válido na matriz. Nesse caso, considere como se a partícula nessa posição fora da matriz fosse de cimento.

Criando partículas

Para criar partículas, usamos da entrada padrão. Por exemplo, digamos que queremos adicionar uma partícula de areia na posição `(10, 10)` antes do primeiro frame ser desenhado. Então enviamos na entrada padrão: `0: 10 10 #`.

Na hora de criar o programa, devemos primeiro ler uma linha da entrada padrão e armazenar os valores fornecidos. Então, podemos rodar nossa simulação até o frame desejado. Chegando lá, criamos a partícula pedida e lemos mais uma linha da entrada padrão e assim segue. Aqui vai um código de exemplo de como isso pode ser feito. Sua implementação **não precisa ser igual**, basta seguir às mesmas regras.

```
int n_frames, frame, x, y;
char nova_particula;

scanf("%d", &n_frames);

int contador = 0;
while (contador < n_frames) {

        // `scanf` retorna EOF quando chega ao fim da entrada.
        int n_lido = scanf(" %d: %d %d %c", &frame, &x, &y, &nova_particula);

        // Se não há mais partículas a serem criadas, continue até o final
        // da simulação.
        if (n_lido == EOF) {
                frame = n_frames;
        }

        // Calcula todos os frames até o próximo frame onde queremos criar
        // uma partícula.
        while (contador < frame) {
                printf("frame: %d\n", contador + 1);
                /* Seu código de imprimir a matriz */
                /* Seu código de calcular a física */
                contador++;
        }

        // Se há uma particula a ser criada, crie ela.
        if (n_lido != EOF) {
                matriz[y][x] = nova_particula;
        }
}
```

Desafio adicional

Nota: Isso se trata apenas de algo **opcional** que não deve ser entregue.

Com algumas adições no código, é possível tornar as partículas coloridas e renderizar a simulação no terminal de maneira mais fácil de visualizar. Aqui vai um exemplo: