C--ampo-Minado

Feito por Marcus Siqueira Coutinho como um teste para a Manifesto Games

Índice:

O teste:

Campo Minado em C++

Para este teste, deve-se seguir as regras padrão do jogo: https://pt.wikipedia.org/wiki/Campo_minado

O que se espera:

- Um grid com posições de bombas geradas aleatoriamente
- Ao tocar num local, se houver uma bomba, a partida termina
- Se ao tocar num local, não existir uma bomba, um de dois eventos devem ocorrer: 1 - Um número aparece, indicando a quantidade de quadrados adjacentes que contêm minas; 2 - Nenhum número aparece. Neste caso, o jogo revela automaticamente os quadrados que se encontram adjacentes ao quadrado vazio, já que não podem conter minas;

Opcional:

- Fluxo de telas, com tela de início de partida, tela de jogo, tela de game over, etc
- Possibilidade de o jogador adicionar/remover manualmente uma bandeira em uma posição, que indica um local onde não há bomba
- Possibilidade de o jogador escolher as configurações de sua partida (Quantidade de bombas, tamanho do grid, etc)
- Um powerup que ajuda o jogador (fique à vontade para criar seu comportamento)

Apesar de se tratar de um jogo simples, tente estruturá-lo de forma organizada, documentando o código e fazendo uso de classes quando necessário. Fique à vontade para usar qualquer framework que desejar, como SFML por exemplo, ou até para fazer apenas via terminal, o que está sendo avaliado não é a questão visual do jogo, então não se preocupe muito com isso.

Para a entrega você deve apresentar:

- Um link de acesso ao repositório Git com as suas alterações
- Um link com a build Windows

Links:

Link de acesso ao repositório Git:

https://github.com/ArcaXbcXde/C--ampo-Minado

Link com build Windows (x64):

https://github.com/ArcaXbcXde/C--ampo-Minado/releases/tag/1.0

Link para esse documento:

https://docs.google.com/document/d/11jVmla50TEFOLO5_mZgby9Rz1h34lElK/edit?usp=sharing&ouid=112505681553041733733&rtpof=true&sd=true/

Link para o Paletton com as cores utilizadas:

https://paletton.com/#uid=73x2c0kxlK3k4NXpYLvIXvXILoA

O que foi feito:

Campo:

- Uma matriz com 9 espaços verticais e 10 espaços horizontais;
- Há um total de 12 bombas;
- Cada espaço dessa matriz, quando revelado, pode conter uma bomba, um número que indica quantas bombas estão ao redor, ou ser apenas um espaço vazio quando não há bombas ao redor;
- Fundo possui uma cor aleatória, ajuda na variação visual entre cada partida;

Objetivos:

- Revelar todos os espaços onde não há bomba;
- Se o jogador revelar uma bomba, ela aparece como explodida e é considerado que o jogador perdeu;

Clique esquerdo do mouse:

- O espaço é revelado e:
    - Se for uma bomba, o espaço será revelado como uma bomba explodida e todas as bombas são reveladas, não sendo possível revelar mais espaços;
    - Se não, revela o espaço e:
        - Se o espaço não houver bomba por perto, revela todos os espaços ao seu redor que ainda não foram revelados, e se os próximos espaços que forem revelados também não tiverem bomba por perto, irão revelar todos os espaços ao seu redor que ainda não foram revelados, e assim por diante;
        - Se o espaço houver uma bomba por perto, revela apenas ele;
        - Se for o último espaço faltando para ser revelado, revela ele, todas as bombas e o jogo termina;

- Ao segurar o clique, o espaço aparece marcado e só é revelado se o jogador soltar o clique em cima do mesmo, caso contrário o espaço volta ao padrão e nada é revelado;

Clique direito do mouse:

- Marca o espaço clicado com uma bandeira ou remove-a, mas não interfere no comportamento do jogo de forma alguma;
    - Serve para o jogador usar como quiser, mas seu principal uso é para marcar locais onde ele sabe que há uma bomba;
        * Existem versões de campo minado que também há o símbolo de “?”, pensei em substituir a bandeira pelo “?” no segundo clique e remover no terceiro clique, mas não gostei de usá-lo, embora seja possível dar um uso interessante para ele.

Clique central do mouse (apertando a rodinha do mouse):

- O espaço é revelado e:
    - Se for uma bomba, revela ela e não explode (espaços revelados dessa forma não contam para o total necessário para a vitória);
    - Se não, revela o espaço mas ao invés de aparecer um número ou um espaço vazio, aparece um alicate no local, indicando que o alicate foi usado e não havia uma bomba lá (embora perca um pouco de informação, acredito que sendo bem usado, essa informação não é algo que irá fazer muita falta para um jogador experiente, fora que precisa de algum ponto negativo com uma ferramenta tão boa assim).
- No total o jogador tem 3 desarmes;
- Ao segurar o clique, o espaço aparece marcado e só é revelado se o jogador soltar o clique em cima do mesmo, caso contrário o espaço volta ao padrão e nada é revelado ou gasto;

Teclado:

- “Esc” sai do jogo;
- “R” reseta o jogo;
- Barra de espaço mostra no terminal quantos espaços faltam ser revelados para o fim do jogo;
- “Z” mostra uma matriz no terminal com todos os números e bombas do campo, com bombas sendo sinalizadas com “xx”, facilita o teste e avaliação;
- “X” mostra uma matriz no terminal indicando quais espaços foram revelados se estiverem marcados com “1” ao invés de “0”, facilita o teste de algumas características;
- “C” mostra uma matriz no terminal indicando as coordenadas de cada posição no campo, facilita na lógica do desenvolvimento;

* Inclusive no desenvolvimento inicial inverti a lógica de ordenação de cada sprite no campo, e como para reverter isso eu iria gastar muito tempo e esforço resolvi manter e utilizar como aprendizado, funcionou muito bem. Não influencia de forma alguma na jogabilidade, apenas na lógica;

O código:

Definições e padrões usados:

- Organizei as variáveis em grupos de ordem de uso > constantes mais simples > constantes mais complexas > variáveis mais simples > variáveis mais complexas;
    * O mais simples usado seria o bool e o mais complexo seriam objetos;

- Usei uma quantidade excessiva de comentários para tentar demonstrar o melhor possível o que cada trecho no código faz;

- Busquei escrever variáveis que são para ter valores constantes com o nome inteiramente em maiúsculo e separado por "_", e valores variáveis com nome minúsculo com palavras separadas por letra maiúscula;

- Nos comentários, "space" é dito como um espaço no campo, como por exemplo o sprite na posição (3, 5);

- Busquei manter o código o mais versátil possível, substituindo números ou strings fixas por variáveis com valor constante

- Busquei manter cada função o mais granular possível para aumentar a possibilidade de reuso de código

- Defini e separei as etapas de execução do jogo em classes. Por exemplo, o jogo começa com o código se configurando, ao se configurar, mostra a tela inicial que no caso é a do próprio jogo, e ao ter a tela pronta manuseia os controles do jogador. Se o jogo tivesse mais telas implementadas, seriam novas classes que estariam controlando essas telas

- Chaves começam na mesma linha que a declaração do grupo, mas possuem uma linha vazia abaixo;

- Estruturas lógicas e de repetição possuem um espaço entre seu nome e sua condição

- No caso de múltiplas condições em uma estrutura lógica, de repetição, ou vários argumentos em um cálculo, separei por parêntesis;

- Declarei todas as variáveis fora do método que a utiliza, mesmo se é usado apenas por um método;

- Else fica na mesma linha do fechamento do if;

Foi deixado de lado:

- Separar o arquivo main em vários arquivos menores;

- Arquivos header (diretamente ligado com o anterior, gastei mais tempo do que deveria tentando fazer tudo funcionar com ele);

- A classe “Game” poderia ser subdividida em outras classes;

- Vetores, cada dia que passou tentei implementar vetor em alguma parte do código, e sempre tive algum problema diferente. Se eu voltar a trabalhar nesse projeto, essa será a primeira coisa que irei atrás de implementar;

Recursos utilizados:

- Segui sua sugestão de utilizar o SFML;
- O Paletton foi utilizado para encontrar cores que se adequam melhor ao visual que eu estava buscando;
- Fiz minha própria spritesheet em um programa chamado Piskel para todos os estados que um espaço no campo pode ter;
- Fora a spritesheet, tudo foi feito utilizando Microsoft Visual Studio, a build, os commits, etc.;

Configuração da janela:

- WINDOW_X e WINDOW_Y: definem o tamanho padrão da janela;
- WINDOW_TITLE: um título para a janela;
- window: envia as configurações da janela para o SFML;
- backgroundColoration: Cor de fundo que será aplicado no fundo da janela atrás do sprite transparente usado como fundo

Configuração do jogo:

- gameSet: se o jogo foi configurado, usado para resetar o jogo;
- fieldX: largura do campo;
- fieldY: altura do campo;
- bombAmount: quantidade de bombas;

- safeSpacesAmount: Quantidade de espaços que o jogador precisa revelar, usado para condição de vitória;
- pliersAmount: quantidade de desarmamentos que o jogador pode efetuar;

Configuração de sprites e texturas:

- SPRITESHEET_x e SPRITESHEET_Y: definem quantos ladrilhos existem na minha spritesheet (automatizável com vetor);
- RECT_SIZE: tamanho {x, y} de cada ladrilho na spritesheet;
- SPRITESHEET_FILE: localização da minha spritesheet no explorer;
- spacesRevealed: Marcações para o campo inteiro para saber se o campo foi revelado ou não;
- flagged: Marcações para dizer quais espaços foram marcados com bandeira pelo jogador;
- fieldStatus: Grava qual o número de bombas ao redor de cada ladrilho, e se o ladrilho é uma bomba com valores de 10 para cima;
- textures: matriz com cada ladrilho separado como textura do meu spritesheet;
- spacesSpr: Define a textura usada por cada sprite no campo;

Classe GameSettings:

* responsável por inicializar as configurações do jogo

private:

- spacesTex: Textura usada como a textura padrão de um espaço não-revelado

public:

método getSpacesTextures: Traz cada ladrilho da minha spritesheet presente no local da constante “SPRITESHEET_FILE” para a variável “textures” utilizando as constantes “SPRITESHEET_X” e “SPRITESHEET_Y” para dizer quantos ladrilhos a spritesheet tem horizontalmente e verticalmente, com o apoio da constante “RECT_SIZE” para dizer o tamanho dos ladrilhos;

método setSpacesSprites: diz qual posição na matriz “textures” está o ladrilho padrão para a variável “spacesTex”, aplica como textura em todos os sprites presentes na matriz “spacesSpr”, e com um cálculo usando metade o tamanho da tela, subtraindo metade do tamanho do ladrilho, somando sua posição na matriz e subtraindo um deslocamento de quantos sprites estão na matriz, define onde cada sprite deve ficar posicionado na tela;

método zeroFieldStatus: Zera todos os valores das matrizes “fieldStatus” e “spacesRevealed”

Classe Screen:

* responsável por controlar o que aparece na tela
private:

- bgTex: ladrilho usado como imagem de fundo;
- bgSpr: sprite usado como imagem de fundo;

método randomColor: simplesmente retorna uma cor aleatória, usado para aleatorizar a cor do fundo e dar um pouco mais de variação visual a cada partida;

método drawField: desenha todos os sprites presentes na matriz “spacesSpr” na tela;
public:
método setBackground: define o que é o fundo com uma cor aleatória, uma textura da matriz “textures” e configura a variável “bgSpr” para ser desenhado na tela

método drawScreen: limpa a tela anterior, desenha o fundo, o campo e mostra para o jogador, nessa ordem;

Classe Game:
* Poderia ser subdividido em outras classes, é usado para manusear todas as interações do jogador com o jogo e o estado atual do jogo;
private:

- breakLoop: garantia de que todos os loops dentro dos controladores do mouse serão quebrados quando necessário;
- holding: uma marcação para garantir que apenas um sprite será marcado como segurado;
- heldSpace: posição {x, y} que o jogador começou a segurar o clique do mouse;
- spaces: apenas uma string para uma visualização melhor no terminal;
- spacesHitbox: A hitbox que cada sprite no campo ocupa;

método changeSprite: muda um sprite para a textura de valor respectivo;

método revealSpace: verifica se o jogador revelou uma bomba, um número ou um espaço vazio, e se revelou um espaço vazio revela todos os outros não-revelados ao seu redor, e os espaços vazios também revelam os espaços não-revelados ao redor, e assim encadeado até revelar todos os espaços vazios agrupados, e por fim se restou mais nenhum espaço sem revelar que não seja revelado, o jogo considera que o jogador ganhou;

método revealAllBombs: revela todas as bombas não-reveladas presentes na matriz “spacesSpr”;

método gameWon: chamado quando não há mais espaços que não sejam bomba e chama o método de revelar todas as bombas;

método gameLost: chamado quando o jogador explode uma bomba e chama o método de revelar todas as bombas;

método coutFieldStatus: revela no terminal todos os valores da matriz “fieldStatus”;

método coutSpacesRevealed: revela no terminal todos os valores da matriz “spacesRevealed”;

método coutPositions: revela no terminal todas as coordenadas do campo;

método coutSpacesLeft: revela no terminal o valor da variável “safeSpacesAmount”;

public:

método setBombs: Limita a quantidade de bombas no campo para sempre ter 1 espaço vazio e adiciona os valores de “fieldStatus” para 10 onde cada bomba foi sorteada e adiciona 1 a todos os espaços adjacentes de cada bomba na matriz, se a bomba cair em um local repetido não coloca a bomba e tenta denovo;
* seria extremamente mais eficiente se houvesse uma matriz para dizer onde cada bomba irá ficar;

método getSpacesHitbox: Pega as bordas de cada valor na matriz “spacesSpr” e salva na matriz “spacesHitbox”;

método handleKeyboard: manuseia inputs de teclado do usuário, sendo R para resetar o jogo, barra de espaço para mostrar quantos espaços faltam do campo no terminal, Z para mostrar uma matriz de todos os valores do campo no terminal, X para mostrar o que foi revelado do campo no terminal e C para mostrar quantos espaços faltam para o jogador revelar do campo no terminal;

método leftMousePressed: Verifica onde o jogador começou o clique do mouse e o que acertou;

método leftMouseReleased: Verifica onde o jogador largou o clique-esquerdo do mouse e se for um espaço não-revelado o revela;

método rightMouseClick: Verifica onde o jogador clicou com o botão direito do mouse e marca ou desmarca com um ladrilho de bandeira;

método middleMouseClick: Verifica onde o jogador largou o clique-central (clique da rodinha do mouse) e, se o jogador possui desarmador (ou alicate) sobrando, revela o espaço ou com uma bomba sem explodi-la caso o espaço seja uma bomba ou com um ladrilho de alicate indicando que um alicate foi usado alí;

Main:

Ele primeiro instancia um objeto para cada classe, em seguida chama todos os métodos necessários em ordem para definir como o jogo vai acontecer, e em seguida entre em um loop onde todos os eventos do jogo ocorrem, e dentro desse evento ele verifica se a janela deve ser fechada (que só aí ele sai desse loop), se o jogador ofereceu algum input do mouse ou do teclado, e por fim desenha o atual frame na tela;
* para resetar o jogo basta definir a variável “gameSet” como false;

Comentários de última hora:

- Eu deveria ter pensado em incomodar mais a Luiza da Manifesto Games com perguntas, agora que estou para entregar penso em várias perguntas, como:
    - Estou em uma competição com outros? Provavelmente sim mas se eu tivesse pensado nisso antes eu não teria feito com tanta calma e quebrado a cabeça por tanto tempo com coisa que nem cheguei a usar;
    - Posso enviar antes e ir melhorando até o prazo final? o "esperado" está pronto desde o começo/meio do 3º dia de produção;
    - O que pesa mais na avaliação do meu código? Será melhor avaliado a questão do que e como estou usando ou o quão bom é minha lógica e organização?;
    - Documentar seriam apenas comentários ou criar um documento explicando cada parte do processo para que vejam melhor todo meu processo de criação? Não importa, fiz tudo;
    - Quando falou que a entrega é até "Meia-noite de domingo 10/10", seria a meia-noite de sábado para domingo ou a meia-noite de domingo para segunda? Nitidamente arrisquei nessa mas deu a entender o segundo caso;
    - Onde preferem que o link da build esteja hospedado?

- Não sei se vão se dar esse trabalho, mas curiosamente espero que leiam meus commits no Github, pelo menos a primeira parte que fala meu processo de pensamento de cada push;

- Para quase todas as matrizes que criei no código, juro que foi uma tentativa de criar um vetor, que é bem mais simples e torna o código extremamente mais versátil, mas todos os dias esse detalhe foi deixado para última hora e acabei não deixando vetores no código final;

Eu sendo aceito ou não, realmente agradeço essa oportunidade que me deram de ter uma chance de trabalhar com vocês, aprendi bastante com o teste que me passaram e agora tenho mais uma opção de framework que posso investir tempo para conhecer melhor e poder ter mais versatilidade para futuros projetos

- Marcus Siqueira Coutinho