ROTEIRO

ROTEIRO

Considerações gerais:

- Struct global: praticamente todas funções acabam recebendo um struct gigante com todas variáveis.
- Sempre plotar as coisas em uma imagem, para então enviar imagem a tela. Se plotar na tela direto fica extremamente lento (inviável). So' usei uma imagem.
- Usei sempre o flag do f-sanitize para compilar, não deixa passar nenhum memory leak.
- Usar coordenadas como double ou float e so' arredondar / converter para inteiro no ultimo instante. Arredondamentos antes da hora destroem todos algoritmos.
- Fazer free sempre após usar uma variável, minimizando o numero de *frees* necessários no final do programa.
- Ao usar arquivos XPM que editei, causava memory leak … usar arquivos .xpm prontos, ou aprender a editar direito os arquivos.
- Funções uteis da libft: ft_strlen, ft_strtrim (limpa as strings de espacos etc), ft_split (para pegar os argumentos das configurações), ft_strncmp, ft_strdup (corrigi 3 funções da libft que estavam com leaks / bug, mesmo tendo sido aprovadas pela Moulinete ...).
- A função de key_hook da miniblix acaba sendo o loop principal do programa. Tem uma outra função para fazer loop caso nao haja nenhum evento que usei apenas para enviar imagem a tela (se nao houve evento, nao precisa calcular nada … se calcular vai ficar mega-lento …)

Fazer parse e check dos argumentos

Checa se tem 1 ou 2 argumentos

Se tiver o 2o tem que ser "—save"

Checa se o 1o termina com ".cub"

Depois quando tentar abrir o arquivo automaticamente checa se o caminho / arquivo eh valido

Fazer parse e check das configurações

Uma função para cada opção (R,C,F,NO,SO,EA,WE)

Para as texturas o check acontece na hora de tentar abrir os arquivos

R: checar se não começa com espaço, se existem exatamente 2 argumentos separados por espaço, se os argumentos sao numeros.

C e F, idem R porem separados por virgula. Check adicional se estão no intervalo 0 a 255

Fazer parse e check do mapa

Primeiro le a parte do mapa e armazena em um array bidimensional, onde cada linha tem um comprimento diferente (strings)

Depois normaliza o mapa, colocando-o em um array do tamanho certo para ficar retangular (tem que saber num de linhas e comprimento da maior linha)

O mapa normalizado facilitar o algoritmo la na frente e facilita a checagem:\

Checar se nao ha uma linha vazia no meio do mapa
Checar se na 1a linha, ultima linha, 1a coluna e ultima coluna so existem espaços ou 1
Checar se existe exatamente 1 player (N, S ,E,W)
Checar se todos caracteres do mapa sao validos.
Checar para cada espaço se os 8 caracteres ao redor são “1” ou “espaço” (essa lógica garante que o mapa eh fechado por paredes)

Desenhar minimap

Blocos
Player
Raios

Se houver argumento “—save” salva como bitmap e sai, senao segue programa (agora pode criar uma janela. Enunciado diz que com flag “—save” não pode abrir janela …);

Fazer funções que atualizam dados do jogo a partir das teclas

Faz update da posição e angulo do jogador
Apos calcular nova posição, checar se ela e valida, isto eh, não entrou numa parede. Se entrou, não altera a posição do jogador.
Importante manter o angulo sempre entre 0 e 2*PI;

Calcular todos raios

Loop com o num de raios, calculando um por um e guardando num array
Distancia ate parede
Calcular distancia horizontal
Calcular distancia vertical
Pegar a menor delas (cuidado para o caso das distancias serem iguais !!!)
Guardar se o raio chegou em uma intersecção vertical ou horizontal (importante para saber qual coluna da textura precisa pegar para desenhar)
Guardar qual quadrante o raio esta: para cima e esquerda, para cima e direita, para baixo e esquerda, para baixo e direita (importante para saber qual textura aplicar)

Desenhar minimap

Escolher um fator de escala adequado. Quando a resolução especificada eh muito pequena as vezes o mínimap não cabe. Coloquei uma condição que nao desenha o mínimap se ele nao couber na janela …

Desenha 3d (chão, teto, colunas com texturas, sprites)

Desenha 2 retângulos para serem o teto e o chão (metade de cima e metade de baixo da janela). A linha central eh o “horizonte” e será a referencia para o tamanho das colunas a serem desenhadas

Loop para cada coluna / raio
Calcula altura da coluna proporcional a distancia do raio
Identifica quadrante do raio para saber qual textura pegar.
Identifica onde o raio bateu na parede (para saber qual coluna da textura pegar)
Pega coluna da textura e faz regra de três para desenhar no tamanho correto

Desenha sprites

Faz um loop varrendo o mapa e montando um array de sprites contendo as seguintes informações:
Coordenadas da sprite (assumi com sendo sempre o centro do bloco onde esta)
Distancia ate o player
Angulo para o player (usando a função atan2)
Ordena o vetor da sprite mais distante para a mais próxima (eh a ordem que tem que

desenha-las)
Faz um loop do array de sprites para (eventualmente) desenhá-los
Checa se sprite eh visível (se não for, nao desenha)
Checa se tem algum raio na mesma posição mais próximo que a sprite (se tiver, não plota sprite porque ela esta atras da parede)
Faz calculo semelhante as texturas para pegar coluna correta do Sprite.
Calcula regra de três para plotar a coluna no tamanho correto
Check adicional para não plotar a cor 0 da sprite (para dar o efeito de transparência)
Coloca conteúdo da imagem na tela

Função que sai (via ESC ou via hook do botão "X")
Destruir ponteiros da MLX, da(s) imagem(s) e da janela. A MiniLibX tem funções especificas para isso.
Função que sai quando encontra um erro (bom ter um array de msgs de erro e plotar msg adequada)

Utilidades:
Conversor de imagens para xpm:
https://www.online-utility.org/image/convert/to/XPM

Site para criar cor (rgb):
https://www.w3schools.com/colors/colors_rgb.asp?color=rgb(0,%20191,%20255)

Links diversos:
https://github.com/taelee42/mlx_example
https://github.com/keuhdall/images_example
https://qst0.github.io/ft_libgfx/man_mlx_new_image.html
https://harm-smits.github.io/42docs/libs/minilibx/sync.html
https://github.com/humblEgo/cub3D_map_tester