**Laboratório de Programação**

Bruno Guilherme Ricci Lucas |nºUSP 4460596|

André Luiz Abdalla Silveira |nºUSP 8030353|

Matheus Takeshi Yabiku |nºUSP 7629949|

**Relatório Unificado**

**Parte 3:**

Para a parte 3 do projeto, faltava o nosso grupo implementar os controles (rotação, aceleração e disparos), a navegação, os projéteis e as colisões.

A aprendizagem da utilização da captação de teclas não foi muito demorada, porém tivemos que fazer algumas mudanças no entendimento do programa quando cada tecla era pressionada, pois do modo que existe no xwc, existe uma competição para o acionamento de cada tecla. Com isso, fizemos com que a aceleração fosse binária e constante. Foi feita, também, a direção da nave em graus de 0 a 360, com cada tecla de rotação mudando a direção original em 11graus e 15 minutos.

Quanto à navegação não houveram grande dificuldades, dado que ela já estava pronta em grande parte. Houveram apenas algumas modificações nas funções *moving\_eixo* e *desloca\_eixo* (basicamente, o uso da aceleração, que foi implementada nesta fase).

A parte gráfica já estava pronta desde a parte 2 do projeto, faltando apenas de fato usá-la na função main e na update, o que foi feito sem maiores problemas. No geral, essa parte do projeto transcorreu tranquilamente.

Os problemas apareceram com os projéteis e as colisões. O membro do grupo responsável por essas implementações teve uma virose bem séria na semana de entrega, e não pode terminar a implementação. No domingo ele, já (s: um pouco) melhor, tentou terminar sua parte, mas mesmo com a ajuda dos colegas não foi terminada em tempo hábil. Sendo assim o grupo achou por bem enviar o trabalho como estava e finalizá-lo na parte 4.

As implementações incompletas de projéteis e colisões foram removidas para não afetar o que já estava pronto e o trabalho foi enviado às pressas, e assim o grupo esqueceu (ss: se encontrou impossibilitado) de enviar o relatório e de fazer um makefile, fazendo da entrega 3 uma entrega muito abaixo do que o grupo gostaria.

**Parte 4**

Com os problemas apresentados na parte 3, o grupo precisava não só polir o jogo, mas também implementar os projéteis e as colisões. Além disso, foi identificada a necessidade de se implementar uma escala, para que o grid em que a tela seria impressa não apresentasse problemas.

Como o tamanho do nosso planeta era o da terra, o grupo fez uma escala com a janela do jogo, onde o planeta tinha 200x200p, e a tela 800x800p, logo tivemos que a posição das naves seria entre -25,6e6 e 25,6e6, e fizemos uma função para conversão em um grid de 0 a 800. Foram adicionadas as variáveis posxGraph e posyGraph no Object e Projectile, sendo que essas variáveis guardam a posição convertida para o grid.

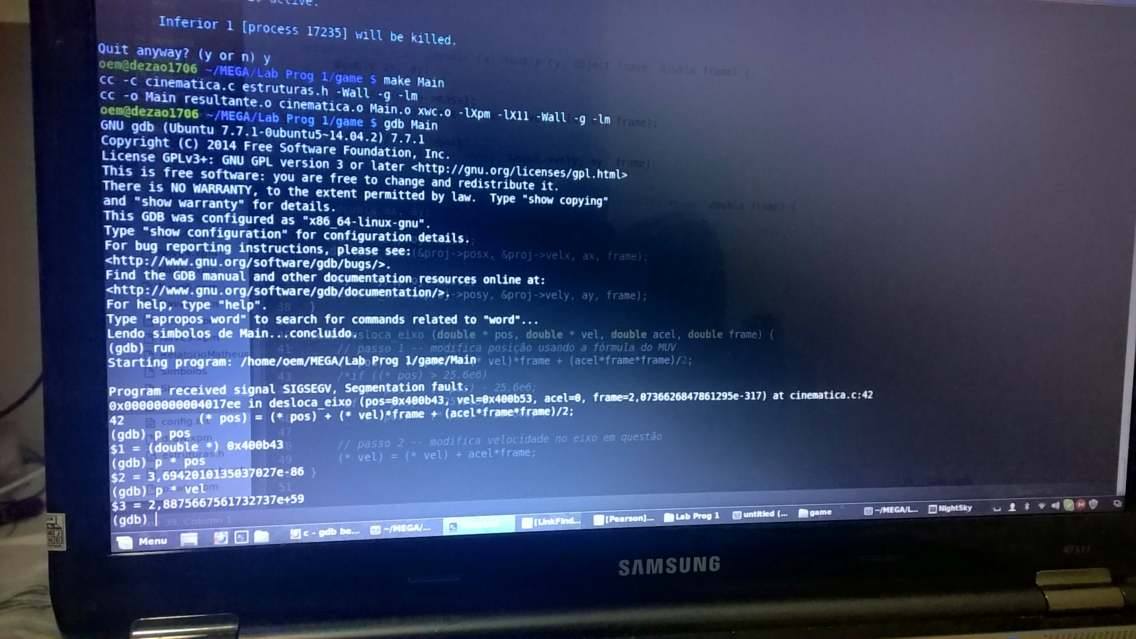
As colisões foram implementadas usando retângulos, no caso, as imagens em si. Sendo que o xwc considera o vértice esquedo superior de uma imagem como sendo a posição (0,0) da imagem, se soubermos suas medidas fica fácil de saber a posição de seus vértices. Sendo assim, a colisão foi implementada verificando se um dos quatro vértices de uma imagem se encontrava em uma posição entre os quatro vértices de outra imagem. Foram feitas quatro funções para tanto, uma que verificava colisão entre as naves, uma que verificava a colisão entre uma nave e o planeta, uma que verificava a colisão entre uma nave e um projétil e a última que verificava a colisão entre um projétil e o planeta.

Os projéteis foram implementados em uma lista ligada, já que o grupo decidiu que não haveria limite para disparos. Os projéteis, além de possuírem massa, velocidade no eixo x e eixo y, posição no eixo x e eixo y, também possuem *time*, que guarda o tempo em que o projétil foi disparado. Sua função é marcar o tempo para, 3 segundos depois, apagar o projétil caso ele ainda exista. Fora isso foi necessária a inclusão de um int id, que serve como identificador de projétil. Ele serviu para a função *apaga* (que está na biblioteca graph) poder fazer a comparação entre os projéteis, e assim encontrar e apagar o projétil correto.

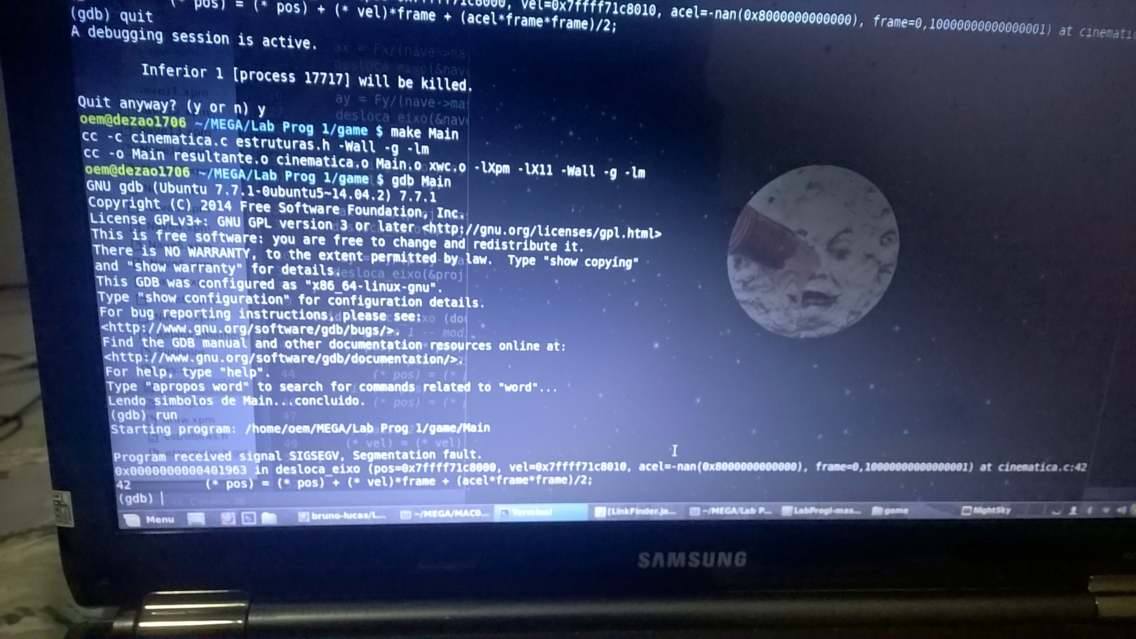
O grupo então resolveu modularizar melhor o código, e criou a graph.h, que guarda as funções relacionadas a controles e à parte gráfica, além das funções insere e apaga, relacionadas ao gerenciamento da lista ligada de projéteis. Foi então criado um novo makefile para esta entrega. Para a criação deste Makefile, da mesma forma como foi feito em outras ocasiões, consistiu em criar arquivos “.o” que convergem no executável **Main.** A maior dificuldade nessa parte foi lembrar onde incluir as flags (“-lXpm -lX11”).

Por fim incluímos imagens criadas por nós para os possíveis finais de jogo (empate, vitória do jogador 1 e vitória do jogador 2) e um limite de duração para a rodada (50 segundos). Feito isso, decidimos testar o jogo, e foram quando os problemas passaram mais sérios e tenebrosos começaram a ocorrer. Falando dessa forma, eu posso até compreender que pode soar como um exagero bobo. Antes fosse. Disse que eram problemas tenebrosos, pois até mesmo as saídas de programas de debugação como o **gdb** não se mostravam esclarecedoras o bastante.

Não houveram grandes dificuldades para se corrigir os erros de sintaxe, mas assim que todos eles foram devidamente corrigidos, o programa começou a dar Segmentation Fault, como vemos na imagem:



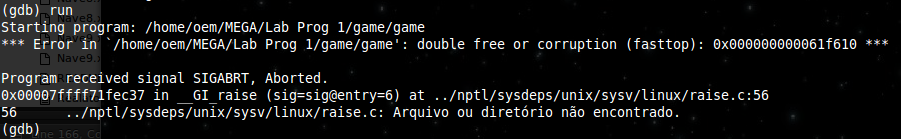
Esse foi o menor dos males, e ainda sim levamos cerca de 1 hora para descobrir que o erro era causado pelo frame, que havia sido inicializado corretamente (s: não seria “incorretamente” a palavra certa?). Então o inicializamos com o valor fixo de 0.1 e o erro sumiu. No entanto o programa ainda continuava dando segfault. A nova mensagem mostrava:



Dessa vez o erro era mais sutil. Muito mais. Alguma coisa estava fazendo a acel assumir um valor não numérico. O grupo levou mais de 3 horas para descobrir a razão. Primeiro foi teorizado que o erro era na função *desloca\_eixo*, mas após muito esforço descobrimos que a função era chamada por duas vezes antes de dar o erro, e sem problemas. Procuramos mais e descobrimos que a causa era uma má inicialização da lista de projéteis. Levou cerca de 20 minutos a mais para que esse erro fosse corrigido.

Testamos novamente o código, e desta vez a imagem de fundo, o planeta e as naves apareceram corretamente na tela. As naves giravam sem problemas, mas a gravidade aparentemente não fazia efeito. Aumentamos ela no arquivo config e ela passou a funcionar corretamente. Mas houve mais um problema. Ao se apertar a tecla ESPAÇO ou ZERO, que seriam as teclas que disparam os projéteis, o jogo voltou a dar segfault. O problema com certeza está na função *insere*, mas depois de mais de 4 horas buscando o grupo, infelizmente, não conseguiu encontrar o erro.

Mas o grupo foi insistente e não se abalou perante as dificuldades que se assomavam assombrosamente. Ainda que soubéssemos que o problema estava ali, foi um tanto quanto desafiador “puxar o fio até chegar ao novelo” para encontrar o coração do problema. Levamos mais algumas (+4h) horas até conseguir rastrear o problema até a linha que nos causava tal dor de cabeça, e descobrir que um outro ponto de implementação também referente aos projéteis estava implementado de forma que receber um segfault era uma questão de apertar ou não o botão de disparo. Sendo assim, decidimos por mudar a implementação de modo a alterar a inserção e remoção de projéteis da lista. Nessa última implementação, as células eram inseridas sempre no começo da estrutura, e a remoção acontecia na ordem que o jogo decorria. Assim, o último problema a ser encarado foi um double free, que foi rapidamente solucionado. Pensávamos que fosse o último dos meus problemas. Mas logo após resolvermos essa última questão, foi retornado um outro tipo de problema mais sombrio que um segfault, uma vez que nem o GDB foi capaz de nos auxiliar (vide imagem abaixo).



Desta forma e considerando que há pouco tempo para solucionarmos esta questão, decidimos por deixar da melhor forma que pudemos.

Está longe da perfeição, mas temos a certeza mais que absoluta que tivemos em mão a oportunidade única de aprender coisas valiosíssimas, nem que estas sejam como evitar ou consertar alguns tipos de erros.

**Considerações Finais:**

- Devemos dizer que o uso da ferramenta GBD foi essencial para corrigirmos boa parte dos erros, apesar de não conseguir resolver todos, para nossa tristeza.

- O grupo chegou ao consenso que usar ponteiros tem suas vantagens, mas para a construção de um jogo eles mais atrapalham que ajudam. Se o jogo fosse feito usando Python, por exemplo, com certeza não teríamos metade dos problemas que tivemos.

- Alocação de memória causa mais dores de cabeça do que resolve. Às vezes é melhor lidar com uma linguagem mais lenta e menos otimizada do que passar pelo estresse causado por um erro de alocação na fase de programação.

- Foi muito interessante usar xwc. Apesar de levar algum tempo para se entender como funciona é uma forma bem simples de se usar OpenGL.

- Apesar do estresse causado pelas nossas falhas e os erros de ponteiros e alocação de memória, sentimos que foi um projeto bem produtivo de um ponto de vista de aprendizagem.