Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

MAC0211 - Laboratório de Programação I

EP3 - Batalha Espacial

Professor: Marco Dimas Gubitoso

Autores:
Bárbara de Castro Fernandes - 7577351
Duílio Henrique Haroldo Elias - 6799722
Marcos Vinicius do Carmo Sousa - 9298274

São Paulo - SP, 5 de Junho de 2016

Conteúdo

Conteúdo

1 Introdução

O projeto referente a este relatório faz parte da matéria de Laboratório de Programação I, lecionada pelo professor Marcos Dimas Gubitoso no primeiro semestre de 2016. Sua proposta foi o desenvolvimento em quatro fases de um *game* de batalha espacial semelhante ao jogo *Asteroids*.

Nesta terceira fase do desenvolvimento temos como objetivo criar a parte de detecção de colisões e criar a mecânica de navegação das naves, ou seja, o controle das naves pelo teclado. Para tal, usamos o que já foi desenvolvido na primeira e segunda fase, com algumas alterações básicas nas estruturas de dados. Para melhor organização dos módulos foi acrescentado um *Design Pattern* [8], modelo MVC (*Model View Control*) [7], com o qual os membros do grupo já tinha um certo envolvimento.

2 Desenvolvimento

2.1 Model View Control (MVC)

MVC nada mais é que um padrão de arquitetura de software, separando sua aplicação em três camadas. A camada de interação do usuário **(view)**, a camada de manipulação dos dados **(model)** e a camada de controle **(controller)** [7]. Escolhemos este padrão pela afinidades do membros. Mais a frente iremos esclarecer como foram organizados nossos módulos do jogo.

2.2 Teclado

O controle das naves foram implementados conforme as especificações do enunciado, como pode ser visto em 1.

Tabela 1: Teclas para o controle das naves

Função	Nave 1	Nave 2
Rotação anti-horária	a	←
Rotação horária	d	→
Propulsão	w	↑
Freio	s	↓
Disparo	q	/

2.3 Decisões tomadas

Algumas modificações na estrutura do código foram realizadas nesta fase. Dentre elas, podemos citar:

– Usar um *Design Pattern* para melhor organização código. No caso usamos o MVC como, foi descrito anteriormente 2.1.

2.4 Bibliotecas

As bibliotecas utilizadas nesta fase foram as mesmas que na segunda fase. Não foi necessário acrescentar mais nenhuma.

3 Entrada

Nesta terceira fase o formato de entrada não foi alterado, mantendo-se igual ao da segunda fase:

```
$ ./batalha-espacial [arquivo de entrada] [intervalo
   de atualização]
```

Onde *[arquivo de entrada]* é o nome do arquivo com as especificações dos corpos e com o tempo total de simulação, em segundos, e *[intervalo de atualização]* é o intervalo de tempo, também em segundos, em que as posições dos corpos serão recalculadas.

O intervalo de iteração representa o intervalo em que acontece a atualização de um quadro. Isto é, se o valor dado como entrada for 0.017, isso significa que a cada 0.017 segundos ocorre a atualização de um quadro, o que dá aproximadamente 60 quadros por segundo. O valor recomendado de entrada é de 0.017.

4 Estruturas de dados

4.1 Nave

Em **Nave** foram adicionados os campos: *raio*, que auxilia no cálculo de colisões; *energia*, que representa a quantidade de vida que a nave ainda possui; *angulo*, que auxilia no desenho da nave; *propulsaoAtivada* e *freioAtivado*, que ajudam no correto deslocamento da nave e na sua animação.

```
typedef struct nave {
        char nome[TAM_MAX_NOME];
        float massa;
        float raio;
        float energia;
        float angulo;
        Vetor posicao;
        Vetor velocidade;
        Vetor aceleracao;
        Vetor forca;
        Boolean propulsaoAtivada;
        Boolean freioAtivado;
} Nave;
```

4.2 Planeta

A estrutura **Planeta** foi mantida como se encontrava na fase anterior.

```
typedef struct planeta {
        float raio;
        float massa;
        Vetor posicao;
} Planeta;
```

4.3 Projetil

Em **Projetil** também houve a adição dos campos: *raio*, que auxilia no cálculo de colisões; *angulo*, que ajuda a desenhar o projétil; *tempo* e *projetilAtivo*, que ajudam a controlar o tempo de vida do projétil.

```
typedef struct projetil {
        float massa;
        float raio;
        float tempo;
        float angulo;
        Vetor posicao;
        Vetor velocidade;
        Vetor aceleracao;
        Vetor forca;
        Boolean projetilAtivo;
} Projetil;
```

4.4 Vetor

Vetor é uma estrutura genérica que permite a construção de um vetor com posições x e y. Ele não sofreu modificações.

```
typedef struct vetor {
    float x;
    float y;
} Vetor;
```

5 Conjunto de módulos

5.1 Organização dos módulos

Nesta terceira fase foi implementado o modelo MVC. Assim todos os módulos foram divididos em *views*, *models* e os *controllers*.

Exemplo: para as naves, temos a *view* (**view_nave.c** e **view_nave.h**), o *model* (**model_nave.c** e **model_nave.h**) e o *controller*, que está num arquivo

controller.c e **controller.h** (por enquanto um arquivo geral para todos os *controllers*).

Todos os outros módulos foram organizados da mesma forma, sendo os arquivos principais o **batalha-espacial.c** e sua *header* **batalha-espacial.h**.

6 Testes

Um arquivo foi adicionado ao repositório do projeto para servir de sugestão de entrada e também como forma de fornecer referenciais de quais grandezas funcionam melhor na execução do programa. Ele está localizado na pasta **tests**. Recomenda-se a sua execução com o intervalo de iteração supracitado de 0.017.

7 Exibição dos resultados

O programa agora abrirá uma janela, onde a simulação dos movimentos dos corpos irá acontecer. Neste processo são utilizadas as funções gráficas responsáveis pelo desenho de cada objeto da tela (naves, planeta, projéteis e plano de fundo), que são redesenhados a cada iteração.

O *GLUT* leva em conta o tamanho da tela do usuário para decidir qual vai ser o tamanho da tela a ser aberta. Esta, por escolha nossa, terá 75% da altura e 75% da largura da tela do usuário. A imagem da janela também se ajusta a redimensionamentos desta.

A exibição dos resultados é feita com a iteração acontecendo em tempo real e depende do espaço de tempo dado no arquivo de entrada como sendo o tempo total de simulação e o intervalo de iteração dado como parâmetro na execução do programa.

As texturas das naves [3][1], projéteis[4], fundo[6] e planeta[2] e a fonte utilizada no jogo[5] foram retiradas de diferentes *sites*.

8 Dificuldades

A maior dificuldade dessa fase foi a realização do cálculo de colisões, principalmente devido ao formato das texturas escolhidas e das funções que desenham o *grid* no GLUT. A função **glOrtho** com os parâmetros internos definidos realiza uma espécie de esticamento do *grid*, fazendo com que as dimensões da tela fiquem distorcidas para janelas de diferentes formatos. Isso resolveu o problema de redimensionamento na primeira fase, mas nesta fase esta mesma solução ocasionou uma dificuldade no cálculo de colisões. Janelas em *widescreen*, faziam do círculos tornarem-se elipses, e isso não correspondia com a textura implementada, deixando a colisão inconsistente.

Para solucionar esse problema, na fórmula do cálculo das colisões, multiplicamos a coordenada x pela largura/altura da tela, isso fez diminuir o problema ao abrir o jogo numa tela *widescreen*.

9 Conclusões

Durante o desenvolvimento dessa etapa, desenvolvemos novas habilidades como o aprendizado das funções que captam eventos de tecla, e aprofundamos o estudo do *grid* para realizarmos os cálculos das colisões corretamente. Fizemos também o desenvolvimento de pequenas animações e movimentos que servirão de base para última fase. Com o término dessa etapa temos um jogo em situação inicial, onde dois jogadores podem disputar uma batalha espacial.

Referências

[1] Gamedevtuts. *Textura da segunda nave*. 30 de abr. de 2016. URL: http://opengameart.org/content/simple-shoot-em-up-sprites-spaceship-starscape-ufo-0.

[2] MillionthVector. *Textura do planeta*. 30 de abr. de 2016. URL: http://millionthvector.blogspot.com.br/p/free-sprites.html?m=1.

[3] Nickname: Phobi. *Textura da primeira nave*. 30 de abr. de 2016. URL: http://opengameart.org/users/phobi.

[4] Opengameart. *Textura dos projéteis*. 5 de jun. de 2016. URL: http://opengameart.org/content/side-blaster-gfx-m484-games.

[5] TutorialWiz. *Fonte utilizada no jogo*. 5 de jun. de 2016. URL: http://www.tutorialwiz.com/starcraft-2-logo.

[6] Wallpaperscraft. *Textura do universo*. 30 de abr. de 2016. URL: https://wallpaperscraft.com/download/stars_galaxies_rotation_universe_68893/2560x1440#.

[7] Wikipédia. *MVC — Wikipédia, a enciclopédia livre*. [Online; accessed 25-maio-2016]. 2016. URL: https://goo.gl/zk8j6m.

[8] Wikipédia. *Padrão de projeto de software — Wikipédia, a enciclopédia livre*. [Online; accessed 25-maio-2016]. 2015. URL: https://goo.gl/S6APjZ.