Universidade Federal do Rio Grande do Sul - Instituto de Informática

Trabalho Final - aplicativo baseado no jogo Dig Dug II

Tiago Mazzarollo de Paula - 00210271

tmpaula@inf.ufrgs.br

Porto Alegre, 26 de Junho de 2016

1. Introdução

Foi implementado um jogo baseado e Dig Dug II utilizando OpenGl. Foram aplicados conceitos aprendidos em aula, como posicionamento, utilização de texturas e posicionamento de câmera.

2. Instruções de jogo

Abaixo estão os comandos disponíveis e a funcionalidade que implementam:

- **w**: movimenta jogador para frente
- **s**: movimenta jogador para trás
- **a**: muda direção do jogador em 90º no sentido anti-horário
- **d**: muda direção do jogador em 90º no sentido horário
- **f**: utiliza arma "bomba de ar"
- **barra-de-espaço**: cria rachadura quando o jogador está em cima de um buraco.
- **v**: troca modo de visualização
- **t:** habilita modo estou com sorte

3. Modos de visualização

O jogo é carregado em uma janela principal e possui uma sub janela no canto esquerdo superior, onde é carregado o minimapa do jogo. Esta câmera está posiciona do ortogonalmente em relação ao plano do jogo e está visível em todos os modos de visualização principal.

Há três modos de visualização principal:

- Em 1º pessoa
- Em 3º pessoa
- Visão aérea (por padrão)

4. Cenário

O cenário é composto por um plano no ponto mais baixo em y, seguido de dois níveis de blocos de tamanho 20x20. No nível inferior estão os blocos de base do cenário os quais determinam as regiões que compõe o estágio carregado. No nível superior ocorrem as

interações do jogo. Esse nível é composto pelos objetos: jogador, inimigo, parede, buraco e rachadura. Os dois níveis do jogo são carregados a partir de *bitmaps*, conforme descrição.

A morte/destruição dos inimigos ou do jogador ocorre quando eles estão em queda e ultrapassam a região do plano base, a qual representa a água no jogo.

5. Colisão

Além da colisão com o plano base citada acima, há o controle de colisão entre blocos de acordo com o tipo. O jogador colide com os inimigos e as paredes, enquanto os inimigos colidem com todos os elementos do nível superior do cenário.

6. Movimentação dos inimigos

A movimentação ocorre de maneira aleatória exceto quando o jogador está próximo, havendo o controle de colisão e de queda. Como explicado anteriormente os inimigos colidem com todos os elementos de cenário, desta formam há o controle de colisão entre dois inimigos. Esse controle foi implementado da seguinte maneira, quando um inimigo tenta se mover é verificado se o novo ponto, de acordo com sua direção, está dentro do espaço de outro inimigo. Caso isso ocorra, o inimigo tendo o seu movimento avaliado para e muda de direção.

A troca de direção dos inimigos ocorre também quando: ou há um buraco na frente, ou é o fim do mapa 20x20, ou há um bloco a frente, ou há outro inimigo no seu caminho ou de acordo com um fator aleatório especificado no código.

7. Corte do cenário

Ocorre conforme determinado na especificação.

Para controle de divisão ao criar um nova rachadura foi utilizado o algoritmo de *Flood Fill*. Para decidir quais as sementes iniciais de teste no caso de existir uma divisão do cenário em duas partes, foi utilizada uma matriz de inteiros representando o tipo de bloco. Percorre-se a matriz buscando a primeira posição com bloco no primeiro plano e vazia no segundo plano. A partir da primeira posição vazia encontrado o algoritmo de *Flood Fill* é aplicado, no processo é verificado se todas as posições da matriz foram atingidas, caso isso não tenho ocorrido existe uma separação do cenário, sendo assim busca-se o primeiro ponto

não atingido e *Flood Fill* é novamente aplicado. Ao final as dias regiões avaliadas têm sua área comparada e a menor é removida.

8. Funcionalidades extras implementadas

As seguintes funcionalidades extras foram implementadas

- Eliminação de buracos e rachaduras cercados por água
- Função “estou com sorte”, eliminando partes aleatórias do cenário, podendo levar ao fim da partida. Com a morte do jogador ou de todos os inimigos.
- Animação de final de jogo.

9. Implementação

Código e histórico de desenvolvimento disponíveis em
https://github.com/tmazza/UFRGS-FCG

10. Imagens