INF01047 - Fundamentos de Computação Gráfica - 2016/1

Especificação do Trabalho Final

Professor Marcelo Walter (marcelo.walter@inf.ufrgs.br)
Dennis Giovani Balreira (dgbalreira@inf.ufrgs.br)

1. Objetivo

Consolidar o conhecimento sobre a representação de objetos 2D e 3D e sua visualização através do desenvolvimento de uma aplicação prática. Exercitar conceitos básicos de Computação Gráfica, como visualização em ambientes 3D, interação, detecção de colisão e utilização de texturas.

2. Especificação

2.1. Descrição

O trabalho consiste em desenvolver um aplicativo baseado no jogo **Dig Dug II** (https://en.wikipedia.org/wiki/Dig_Dug_II). O jogador é carregado em um ambiente tridimensional com cenários na forma de *grids* retangulares, onde cada célula deve ser preenchida com diferentes objetos. O protagonista deve eliminar todos os inimigos do cenário com o auxílio de uma britadeira, usada para rachar terrenos de uma ilha para posterior desmoronamento quando um ciclo for detectado. Os inimigos que estiverem nos terrenos serão eliminados quando ocorrer o desmoronamento.

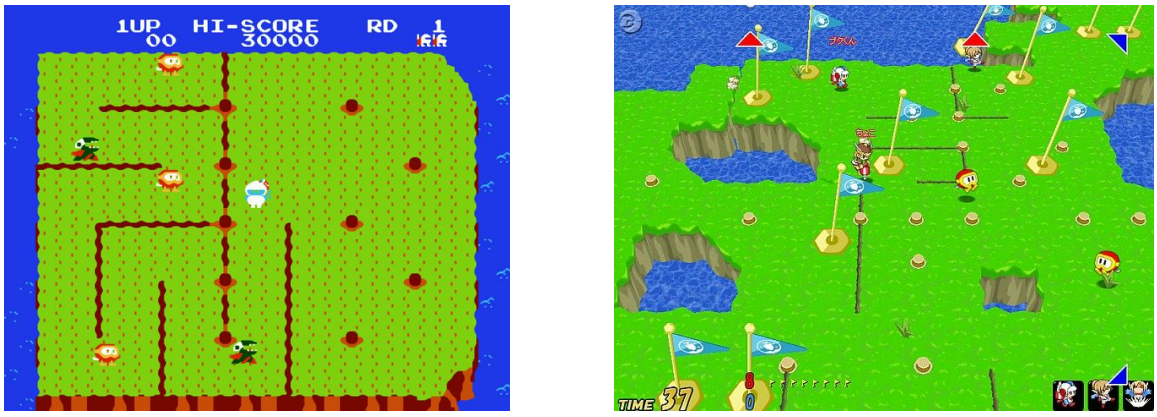

Figura 1: Imagens dos jogos Dig Dug II (NES) (esquerda) e Dig Dug Island (direita).

Vídeo: https://www.youtube.com/watch?v=Ah1h9ROwG6U

O jogo termina com **vitória** quando todos os inimigos são **eliminados** ou **derrota** quando o jogador **colide** com um inimigo ou **cai** na água.

2.2. Requisitos Gerais

2.2.1. Cenário

- O jogo é composto por dois **níveis** (**inferior** e **superior**) fechados com formato retangular e com tamanho de pelo menos 20x20, dispostos um sobre o outro. Os *grids* devem ser inicialmente carregados a partir de arquivos **bitmap** com as mesmas dimensões, onde cada cor representa um tipo de objeto e a posição no pixel representa o seu respectivo local de início no mundo tridimensional;

- Cada posição do *grid* pode ser ocupada por diferentes tipos de objetos:
    - Jogador;
    - Inimigo;
    - Bloco;
    - Buraco;
    - Rachadura;
    - Vazio.

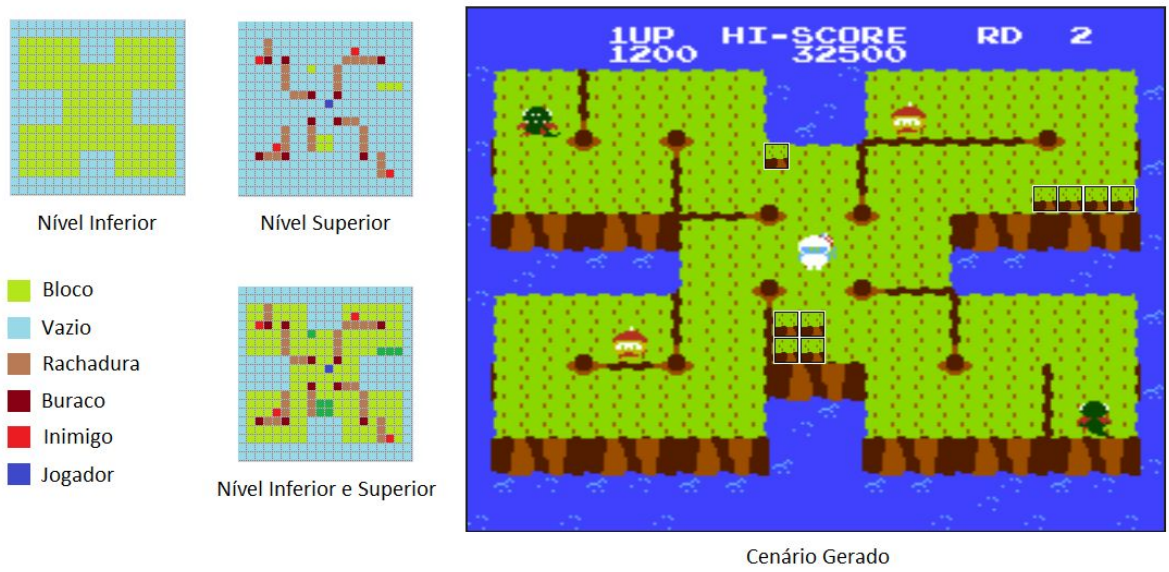

Figura 2: Exemplo de construção de cenários. A partir de duas imagens bitmap representando a posição e os tipos de elementos dos níveis inferior e superior, é gerado o cenário do jogo.

- O **número** e a **alocação** inicial de todos os elementos do jogo devem ser especificados manualmente, editando diretamente o **bitmap** do cenário antes do início do jogo;

- O nível **inferior**, representando o chão, pode ser formado por bloco ou vazio, enquanto o nível **superior** por qualquer um dos objetos;

- Ao colocar **objetos** no **nível superior** que não seja o vazio, deve ser verificado se há blocos imediatamente abaixo no nível inferior.

- Deve haver um **plano** visível representando a água, localizado abaixo do nível inferior, o qual deve prever detecção de colisão dos inimigos e do jogador;

- As **bordas** do mapa contornando o *grid* especificado devem ser vazias por padrão, possibilitando tanto aos inimigos quanto ao jogador caírem na água;

2.2.2. Jogador

- Deve ser controlado pelo teclado, de acordo com as teclas listadas:
    - Tecla **W** (ou seta para cima): andar para frente;
    - Tecla **S** (ou seta para baixo): andar para trás;
    - Tecla **A** (ou seta para esquerda): gira o jogador 90 graus no sentido anti-horário;
    - Tecla **D** (ou seta para direita): gira o jogador 90 graus no sentido horário;
    - Barra de **espaço**: cria rachadura;
    - Tecla **F**: empurra o inimigo;
    - Tecla **V**: alterna entre os diferentes tipos de câmera;

- Pode **criar rachaduras** livremente estando em cima de um buraco. A rachadura irá começar no buraco onde está o jogador até i) um buraco análogo na direção em que estiver olhando; ii) uma outra rachadura; iii) chegue ao final do mapa. Em todos os casos o percurso deve conter células vazias;

- Pode **empurrar** o inimigo duas células utilizando uma bomba de ar. A direção do empurrão será diametralmente oposta à do jogador. O inimigo deve se encontrar a uma distância de até duas células à sua frente e não pode haver blocos no percurso do empurrão. Após usar a bomba de ar e acertar em um inimigo, o jogador só poderá utilizar a bomba de ar novamente passados 2 segundos;

- Pode **andar livremente** no nível superior**, exceto onde houver inimigos** e blocos no mesmo nível;

- É **eliminado** quando colidir com um inimigo (ambos compartilhando a mesma célula) ou quando cair na água.

2.2.3. Inimigos

- Deve ser prevista a alocação no jogo de até **4 inimigos** contendo uma Inteligência Artificial (IA) simples que se comporte da seguinte forma:
    - Caso o inimigo esteja a 4 células de distância do jogador, o inimigo deve se **movimentar em direção ao jogador**;
    - Caso contrário o inimigo deve ficar **andando aleatoriamente** pelo cenário, desde que não se jogue na água e observe as colisões.

- Podem **andar** em células vazias e sobre blocos. Haverá colisão com o jogador e com eventuais blocos, rachaduras e buracos no mesmo nível;

- Devem levar em consideração a posição dos demais para o ***path finding***, ou seja, eles não podem colidir uns com os outros;

- Serão **eliminados** quando caírem na água, seja por estarem em uma região de desmoronamento, ou por serem empurrados para fora pelo jogador.

2.2.4. Bloco

- Caso seja colocado no nível **superior** representa uma **barreira**, ocupando uma célula inteira e impedindo que o jogador e os inimigos atravessem esse espaço. Se for colocado no nível **inferior**, representa o **chão** onde o jogador e os inimigos podem andar.

- Deve ser representado por um cubo que se limite ao tamanho da célula.

2.2.5. Buraco

- Quando o **jogador** se encontra **sobre** ele pode ser possível gerar uma **rachadura,** caso os requisitos de criação de rachadura sejam atendidos;

- Deve haver **colisão** dos **buracos** apenas com os **inimigos**;

- Deve haver um **indicativo** texturizado no chão que indique que aquela célula é um buraco.

2.2.6. Rachadura

- Pode ser gerada a partir de **dois buracos**, entre **um buraco** e a **borda do mapa**, ou ainda entre um buraco e uma rachadura, em uma **mesma linha ou coluna** que não contenha blocos entre eles, desde que o jogador esteja sobre um dos buracos. Nesse caso a rachadura preenche todos os espaços, caso existam, entre o jogador e o elemento que define o final da rachadura;

- Deve haver **colisão** das **rachaduras** apenas com os **inimigos**;

- Deve haver um **indicativo** texturizado no chão ou próximo a ele que indique que aquela célula é uma rachadura.

2.2.7. Desmoronamento

- Ocorre quando um **ciclo** formado por rachaduras, buracos ou o fim do cenário é **fechado** no nível **superior**, dividindo em duas áreas os blocos do nível inferior: internos ao ciclo e externos ao ciclo. Neste caso, a área formada pelo **menor número de blocos** dentre as duas deve ser desmoronada;

- Deve **destruir** (tornar vazio) todos os blocos do nível **inferior** de onde a menor área do ciclo foi detectado. As rachaduras e os buracos não devem ser destruídos. Dessa forma, todos os inimigos que lá estiverem desaparecem (caem na água) e, portanto, são eliminados.

- Sugere-se que a **detecção da menor área do desmoronamento** seja feita utilizando o algoritmo ***flood fill*** (https://en.wikipedia.org/wiki/Flood_fill);

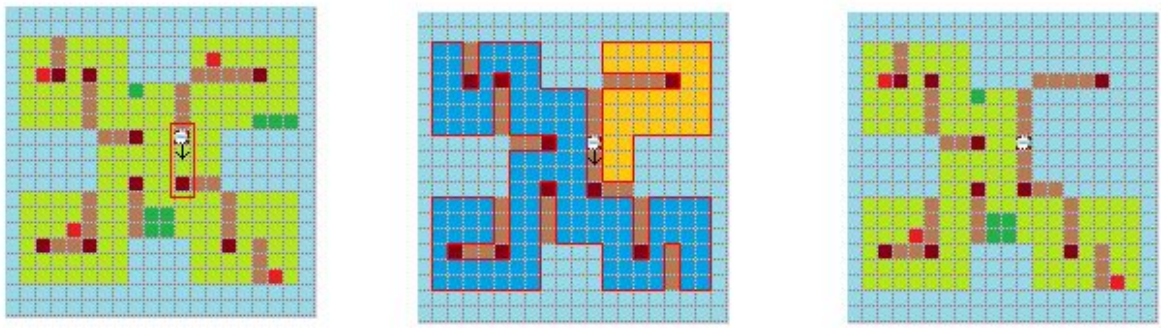

Figura 3: Exemplo de desmoronamento. O jogador utiliza a britadeira sobre um buraco em direção à outro, ativando o algoritmo de verificação de desmoronamento. Duas áreas são geradas a partir da rachadura criada, representadas por azul forte e laranja. A menor área encontrada é a laranja, removendo todos os tipos de objetos que ali se encontravam.

2.2.8. Jogabilidade

- Devem ser implementados **3 tipos** diferentes de câmera:
    - Em primeira pessoa;
    - Em terceira pessoa fixada atrás e acima do jogador;
    - Vista de cima em 2D.

- Deve ser implementado um método de **colisão** entre os objetos do cenário;

- Deve ser implementado um **mini mapa** mostrando uma vista de cima em 2D;

- A posição do personagem e dos inimigos deve ser **contínua** (não discreta), ou seja, a movimentação dentro do cenário deve ser suave.

2.3. Requisitos Técnicos

O jogo deve ser desenvolvido preferencialmente em C/C++ utilizando OpenGL em Windows ou Linux. A opção padrão é utilizar a OpenGL clássica abordada na disciplina, entretanto o uso de Modern OpenGL ou WebGL é encorajado.

Os objetos utilizados no jogo podem ser modelados à mão ou lidos de um arquivo gerado por algum software de modelagem tridimensional, procurando ser condizentes com o contexto do jogo. Além disso, devem ser utilizadas técnicas simples de iluminação e textura para os modelos.

2.4. Desafios Extras

- Implementar um algoritmo que elimine rachaduras e buracos cercados por água;
- Implementar uma nova arma para o jogador capaz de criar e remover buracos;
- Implementar a arma de inflamento do jogo original para eliminação do inimigo;
- Implementar um algoritmo sofisticado de IA para os inimigos;
- Implementar interfaces gráficas (menu inicial, opções, etc.);
- Implementar animações para os modelos;
- Modo para 2 jogadores cooperativo;
- Modo para 2 jogadores competitivo;
- Implementar um cronômetro que mostre o tempo restante para o final da partida, incorporando o tempo como condição de derrota;
- Incluir efeitos sonoros.

3. Avaliação

O trabalho deve ser desenvolvido preferencialmente em **dupla** ou, excepcionalmente, individualmente. **Deve ser utilizado no máximo 30% de código pronto para este trabalho** (por exemplo, leitor de arquivos obj). Qualquer utilização de código além desse limite será considerada plágio e o trabalho correspondente receberá zero de nota.

Os responsáveis pelo trabalho também tem que montar uma **página web** que deverá estar pronta no dia da apresentação final. Nesta página deverá constar um **relatório** simples sobre o desenvolvimento do trabalho, os **fontes** e **executáveis** para download, no mínimo **3 imagens** do jogo funcionando e um **manual** de utilização do jogo (comandos disponíveis). Esta página web é considerada parte integrante do trabalho. Todo o material do jogo (relatório, fontes e executável) deve ser zipado num arquivo e feito **upload no moodle** até às 23:55 do dia **26 de junho**.

Os alunos devem apresentar o trabalho final em laboratório no dia **27 de junho**, quando serão observados os seguintes pontos:

- Criação da *viewport* principal;
- Modelagem/carregamento da cena;
- Aplicação de textura e iluminação da cena;
- Lógica do jogo (condições de término, etc.);
- Aspectos interativos (controle do jogador, mapa, inimigos, etc);
- Detecção de colisão dos personagens entre si e com os objetos e limites do mundo;
- Jogabilidade (tem que ser em tempo real).

Pontos extras serão atribuídos a quem desenvolver soluções para os desafios propostos e valerão até 10% a mais da nota. Somente recebe nota extra máxima (11 sobre 10) quem implementar no mínimo 2 itens de desafios propostos. Outros desafios poderão ser considerados conforme sugestão dos alunos e avaliação pelo professor.

4. Proposta de Jogo Diferente

Os alunos que assim desejarem poderão escolher seu próprio trabalho final. Para tanto devem enviar ao professor responsável, até o dia **8 de maio**, uma mensagem contendo o seguinte:

- Título do trabalho.
- Aluno(s) responsável(veis), no máximo 2 alunos.
- Breve descrição do trabalho, incluindo os conceitos de computação gráfica que pretende utilizar.

Para o trabalho ser válido para a disciplina ele deve incluir os diversos conceitos que serão abordados no projeto final definido pelo professor. No mínimo o aplicativo deverá prever:

- Carregamento de objetos complexos;
- Interação com o usuário;
- Iluminação;
- Texturas;
- Movimentos de câmera;
- Detecção de colisão.