Desenvolvimento do jogo “Quatro” utilizando a ISA RISCV32IM

Development of video game “Quatro” using ISA RISCV32IM

Fernando de Almeida Mendes Dias
Departamento de Ciências da Computação – Universidade de Brasília (UNB)
Brasília – DF – Brasil

Resumo. *Esse relatório apresenta o desenvolvimento de um jogo em arquitetura RISCV32IM, além do desenvolvimento de sua engine, baseado em Bomberman (1985), utilizando o Bitmap Display, KDMMIO Simulator e interface de áudio MIDI.*

1. Introdução

Bomberman é um jogo de arcade desenvolvido em 1983 pela Hudson Soft e lançado para o Nintendo Entertainment System em 1985. O jogo consiste em vários labirintos em que o jogador deve derrotar uma variedade de inimigos utilizando o posicionamento de explosivos para avançar antes que o tempo limite da fase acabe.

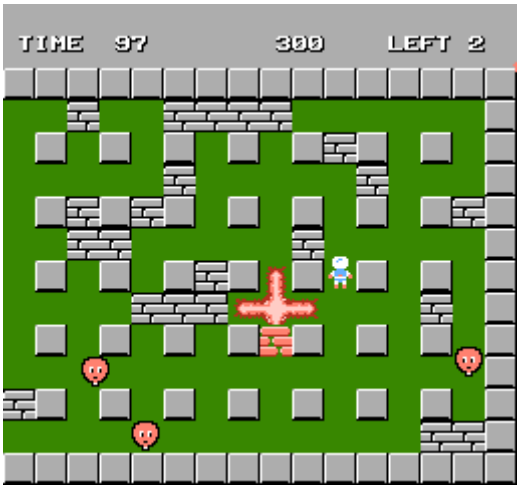

Figura 1. Bomberman (1985)

Este projeto tem como objetivo criar uma releitura do jogo, com recursos similares, junto a uma *engine* flexível e modular na arquitetura RISCV32IM, utilizando-se das interfaces pré-requeridas: Bitmap Display 320x240 pixels (8 bits por pixel); Teclado (Keyboard and Display MMIO Simulator) e áudio MIDI.

Também eram pré-requisitos a implementação de: a) no mínimo duas fases com layouts diferentes; b) animação e movimentação do jogador; c) habilidade do jogador ser capaz de destruir partes do mapa; d) colisão do jogador com os obstáculos e com os limites da área jogável; e) Mecânica de power-ups; f) Condição de vitória; g) Pelo menos 2 tipos diferentes de inimigos; h) Animação e movimentação dos inimigos; i)

HUD (heads-up display) que mostra as informações atualizadas do jogo; j) Música e efeitos sonoros

Para a implementação do jogo e da *engine*, foi originalmente planejado o uso do RARS; Problemas de desempenho, todavia, tornaram necessária o uso exclusivo do simulador FPGRARS, versão 1.13 (8 bits).

2. Metodologia

A *engine* e o jogo em si foram criados com o objetivo de atingir máxima modularização e manutenibilidade, apesar do custo de *overhead* com a quantidade aumentada de chamadas de funções (referidos no projeto exclusivamente como procedimentos ou apenas rotinas). Os módulos foram desenvolvidos para serem altamente reutilizáveis por todo o projeto, agilizando o desenvolvimento.

O desafio inicial foi arquitetar o sistema de fases, de display de texturas, cálculo de posições, hitboxes, e afins. Decidiu-se seguir um sistema baseado nas coordenadas absolutas da tela, isto é, utilizando o canto superior esquerdo como ponto de origem. Separou-se um espaço na memória onde um *tilemap* contendo uma matriz com cada *tile* ocupando uma posição, e criou-se um procedimento para manipular a *tilemap* de acordo com a posição absoluta, que o próprio procedimento convertia para uma posição no *tilemap*, considerando a posição relativa do canto superior esquerdo do mapa e supondo um mapa centrado na tela.

MAPA

O próximo passo foi criar o procedimento de impressão de fase, onde se calcula a posição absoluta do mapa necessária para ele estar centrando, levando em consideração uma constante arbitrária de tamanho de um *tile*. Utilizando constantes similares, foi possível criar um procedimento cujo funcionamento independe do tamanho de um tile e do tamanho do mapa.

O valor escolhido como tamanho de um tile dentro do jogo foi 20 pixels por *tile*. Como a *engine* foi feita para mapas estáticos, considerando a quantidade relativamente pequena de tempo e a grande demanda de esforço que o código em *Assembly*. Assim, derivou-se um tamanho máximo de um mapa, que cobriria a tela toda, como um mapa de 16x12 *tiles*.

Um programa em C foi criado para converter arquivos textos, contendo designs de fases em ASCII, em um pequeno *tilemap* em formato DATA que poderia ser carregado no programa. Isso permitiu a rápida criação de várias disposições que poderiam ser usada em um estilo *plug-and-play*, necessitando apenas a mudança de um parâmetro na chamada do procedimento de impressão de fase para carregar uma fase diferente.

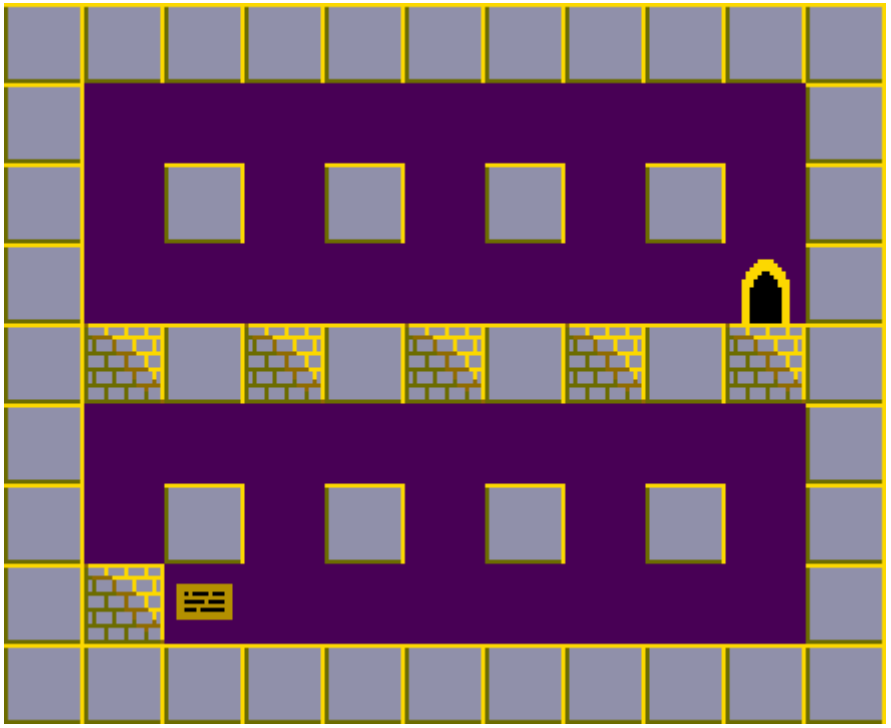

Figura 2: Mapa do Capítulo 1, Fase 4 depois de impressa na tela

COLISAO

Utilizando-se do *tilemap*, foi possível dinamicamente determinar, antes de um movimento, se qualquer um dos quatro cantos do jogador estaria dentro de um *tile* de parede, assim abortando qualquer movimento do jogador que estivesse fora da região caminhável.

Contudo, tal implementação incorreu algumas limitações: 1) O jogador não poderia ultrapassar o tamanho de um *tile*, ou estes poderiam passar por entre os cantos do jogador; 2) Todo e qualquer *tile* teria, na prática, uma hitbox quadrada e inflexível.

Apesar disso, esse modelo de colisão foi adotado pela sua simplicidade e manutenção, além de não necessitar de arquivos separados, facilitando o desenvolvimento.

INIMIGOS

Para calcular a posição de inimigos, foi escolhido um valor (9) que, no arquivo .data original, representaria o último verdadeiro *tile*. Valores a partir de 10 representam inimigos, que são colocados no procedimento de imprimir fase e administrados pelo procedimento PROC_INIMIGOS_MANAGER.

Para simplificar a lógica de movimentação de inimigos, foi escolhido que inimigos sempre teriam o tamanho de um *tile*. Isso permitiu que a detecção de obstáculos e *tiles* caminháveis se desse apenas por checar 8 pixels (2 para cada canto) utilizando a leitura do *tilemap*. Entre as direções disponíveis, cada inimigo aleatoriamente escolhia entre elas, com ir para frente com o maior peso (2), ir para a

esquerda e ir para a direita com 1, e, caso todo o resto falhe, a possibilidade de se mover para trás.

MAIN

Considerando que o jogo está disposto em 4 capítulos principais + 1 capítulo tutorial + 1 capítulo bônus, tornou-se necessário separar a main em rotinas como ROTINA_CAPITULO_0 e ROTINA_CAPITULO_1. Cada capítulo recebeu uma macro de fase, em que cada fase poderia chamar argumentos diferentes em cada macro, e assim personalizar a experiência sem muito esforço.

Para esse projeto, atualmente encontram-se terminados apenas os capítulos 0 (tutorial) e 1 (início).

MÚSICA E EFEITOS SONOROS

Considerando a abilidade do FPGRARS de tocar mais de uma nota ao mesmo tempo, foi criado um algoritmo em Python para converter arquivos midi em arquivos.data em que as notas estariam armazenadas em um struct, ordenadas de acordo com o tempo desde o começo da música que deveriam ser tocadas.

Aliado a um algoritmo em Assembly assíncrono com três faixas de reprodução, foi possível tocar músicas e efeitos sonoros concomitantemente de complexidade maior que o que costumava ser feito, utilizando structs de notas de duração e momento de reprodução flexíveis em vez de notas estritamente sequenciais.

Para criar a música-tema do menu do jogo, e para criar todos os efeitos sonoros, foi utilizado o programa Bosca Ceoil: The Blue Edition, equipado com instrumentos MIDI. O tema do capítulo 0 (A Internacional – instrumental) e o tema do capítulo 1 (Hino Hurrita a Nikkal №6) são de domínio público.

HUD

Para a HUD, se adaptou o sistema para RARS criado pelos alunos do componente de Organização Arquitetura de Computadores da Universidade de Brasília, o *SYSTEMv2.4*, de onde se retirou os procedimentos de impressão de *String* e impressão de inteiros. Assim, foi possível imprimir variáveis do jogo na tela antes de cada evento de desenho no final do ciclo no *gameloop*.

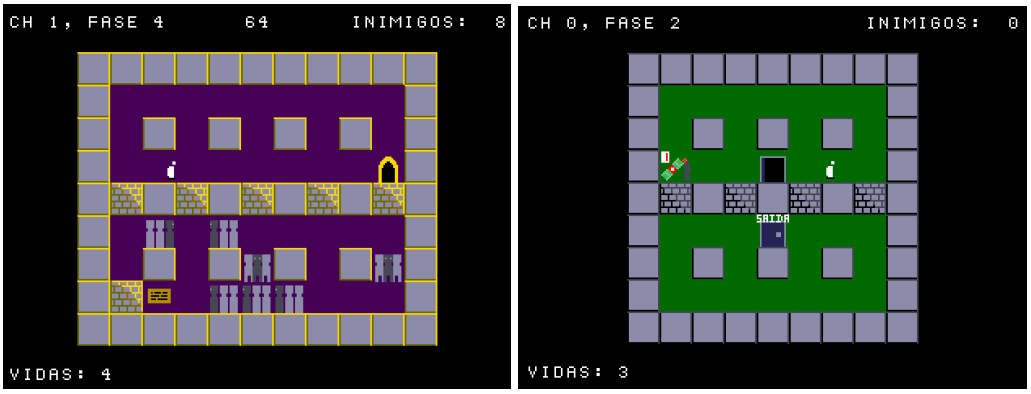

Figura 3: Fases do jogo

3. Resultados Obtidos

Essa versão, embora menos desenvolvida que aquele em que foi baseado, demonstra inovações que o próprio jogo original não possuía (como, por exemplo, a habilidade de reproduzir acordes, ou as texturas diferentes por capítulo). Nota-se o profundo lamento do único membro da equipe de não ter nem ao menos dois dias a mais para entregar esse documento, visto que seria tempo o suficiente para concluir as partes restantes do jogo.

O jogo em seu estado atual consiste de dois capítulos: O capítulo 0, sem tema específico, que entrega as fases de tutorial 1 a 8; e o capítulo 1, com tema babilônico, que entrega 5 fases adicionais, cada capítulo com uma versão cosmética diferente da mesma inteligência de inimigo.

4. Conclusão

O projeto foi bem sucedido. Ele cumpre todos os pré-requisitos, e demonstra a habilidade do único integrante de não só passar uma noite inteira antes da escrita deste documento em claro, sem dormir por um segundo e sem ingerir qualquer tipo de energético, mas também quão longe a determinação de uma pessoa a pode levar.

O desenvolvimento do jogo “Quatro” certamente demonstrou a presença de uma curva de dificuldade incrivelmente íngreme no início do processo, que vai se planeando conforme a *engine* é desenvolvida. Ao final do desenvolvimento da *engine*, o design visual passou a ser a parte com maior demanda de tempo, assim que adicionar fases novas se tornou trivial.

Certamente se lançou uma sólida fundação sobre lógica e linguagens de baixo nível na mente do solitário membro do grupo.

5. Referências Bibliográficas

Bomberman - Disponível em <https://www.retrogames.cz/play_085-NES.php>
Bosca Ceoil - Disponível em <https://yurisizov.itch.io/boscaceoil-blue>
Quatro - Disponível em <https://github.com/fer-amdias/quatro>
FPGRARS - Disponível em <https://github.com/LeoRiether/FPGRARS>

RARS - Disponível em <https://github.com/fer-amdias/RISCV-midiconverter>