Bad Ice-Cream

Ana Caroline Evangelista da Silva
Maria Eduarda Rocha Rodrigues
Maria Salgado Pinheiro

Universidade de Brasília, Dep. Ciência da Computação

Resumo

Este documento retrata o desenvolvimento da releitura do jogo "Bad Ice-Cream", pelas criadoras: Ana Caroline Evangelista Da Silva, Maria Rocha Rodrigues e Maria Salgado Pinheiro, estudantes da Universidade de Brasília na disciplina "Introdução aos Sistemas Computacionais" (ISC) presidida pelo professor Marcus Vinicius Lamar. A versão original do entretenimento foi lançada no ano de 2010 e foi o primeiro jogo Nitrome em Flash. Em suma, Flash é uma tecnologia de plug-in que permite a execução de jogos e animações diretamente no navegador da web, de forma a armazenar os dados na memória não volátil. No entanto, a versão recém-criada usa a arquitetura RISCV e o simulador RARS.

1 Introdução

Atualmente, a maior indústria de entretenimento do mundo é a indústria gamer, isso porque os jogos se tornaram muito populares com a globalização e com uso de tecnologias avançadas que possibilitam melhor resolução gráfica, maior disponibilidade de recursos e um ambiente extremamente propício à inovação e à criatividade, em que a finalidade gira em torno do usuário. Dessa forma, é perceptível que os jogos possuem a capacidade de estimular o desenvolvimento de habilidades cognitivas e sensoriais, como: o raciocínio lógico mais rápido, a coordenação motora mais aflorada e a capacidade de socialização através de canais de comunicação.

Figura 1: Logotipo do jogo

Bad Ice-Cream foi elaborado pela Nitrome Limited, uma empresa britânica com foco em desenvolvimento de jogos. A primeira versão do jogo foi lançada em 2010 e desde então várias versões e atualizações foram sendo implementadas no decorrer dos anos. Inicialmente, ele foi criado com o intuito de ser jogado em arcade multiplayer – os famosos fliperamas com suporte para mais de um jogador -, hoje, já está disponível para smartphones, tablets, laptops, dentre outros. Em síntese, o objetivo do jogo é comer todas as frutas presente no nível enquanto tem que evitar contato com os inimigos, desviar de obstáculos e criar ou destruir blocos de gelo, isso tudo com a finalidade de vencer a etapa presente. Ao contrário da original, a versão apresentada é mais simplificada, a qual porta apenas um jogador, dois vilões e três tipos de frutas, mecanismos usados de forma diferente a depender se da dificuldade de cada nível.

Figura 2: Fliperama

2 Metodologia

Para que o projeto se concretizasse, foi necessário usar o simulador RARS, na arquitetura RISCV, realizado em Assembly. Nesse contexto, as interfaces usadas foram: a gráfica (Bitmap Display de dimensões 320×240 pixels, 8 bits por pixel), o teclado (Keyboard and Display MMIO Simulator) e o áudio (MIDI). Por conseguinte, é válido relatar a ordem de passos realizados pelas estudantes, que foram, respectivamente: a representação do personagem, armazenar dados na memória, a movimentação e o apagamento dos rastros, implementação do tile, a elaboração da lógica de colisões, a animação do personagem, a implantação das fruta e a criação e destruição de blocos de gelo. Dando seguimento, depois, foi criado o inimigo, adicionada a música e o menu com a pontuação, para então implementar as fases seguintes

2.1 Assembly

Uma dificuldade apresentada desde o começo foi compreender a dinâmica mecanicista da linguagem assembly de baixo nível. Sabe-se que a linguagem é complexa, específica e detalhada, por isso foi uma quebra de paradigma trabalhar com ela. Consequentemente, uma grande quantidade de erros de sintaxe apareceram no decorrer da execução do código, os quais sem dúvida trouxeram valiosos ensinamentos, além de capacitá-las para programar de forma mais coesa, organizada e, principalmente, otimizada. Por fim, realizar o projeto em Assembly mostrou como o hardware funciona e como a manipulação de registradores e da memória acontecem.

2.2 Cinematografia

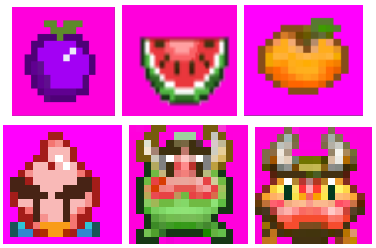


Figura 3: elementos

A arte gráfica foi desenvolvida no Paint.net, com os recursos voltados à computação, como o constante uso da cor magenta que, quando passa pela função “PRINT”, fica transparente na tela. Todas as imagens respeitam o limite estipulado de 16x16 pixels e são previamente convertidas no prompt de comando para “.data” com o arquivo “bmp2oac3”. São necessárias quatro imagens para cada personagem do jogo, que, de acordo com o comando executado no teclado, são acessadas e impressas na tela, o que gera a animação do jogo. O personagem apenas se move após analisar o tile da frente e ver como prosseguir, da mesma forma, a cada pixel que passa é printado o tile da cor de fundo do jogo, a fim de cobrir os rastros. O menu do jogo é atualizado usando os instrumentos “SYSTEMv21.s” e “MACROSv21.s”, os quais são incluídos no final e no início do código, respectivamente.

2.3 Memória

Manipular os registradores limitados, usar a memória e ligar ao armazenamento de posições sem que afetasse o desempenho do jogo não foi uma tarefa fácil.

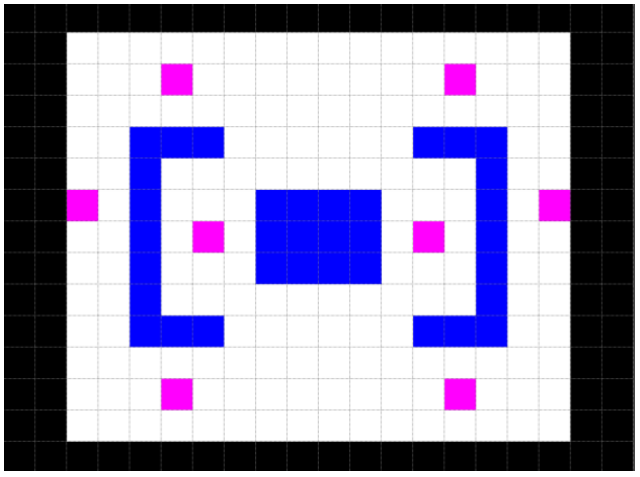


Figura 4: Mapa de colisões da fase 1

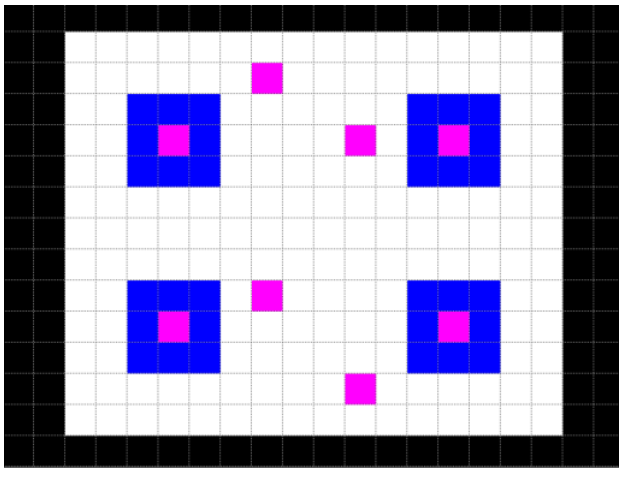


Figura 5: Mapa de colisões da fase 2

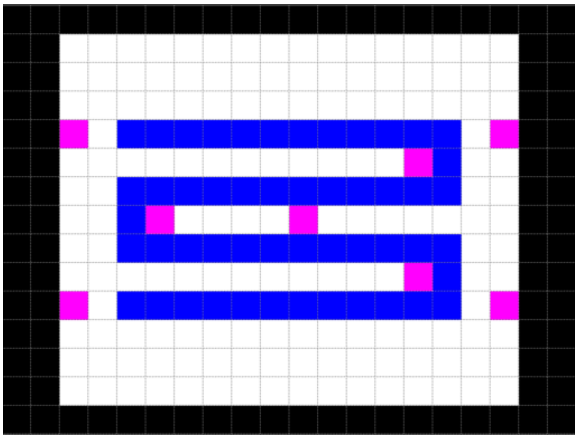


Figura 6: Mapa de colisões da fase 3

A melhor forma identificada foi a da utilização do mapa de colisões, que possibilitaram a movimentação do personagem e dos inimigos com exatidão. Neles, são correlacionados a posição dos objetos e o limite a eles estabelecidos com as cores aplicadas no mapa. Assim, vale esclarecer o que cada cor representa: a borda preta são os gelos que emolduram todas as fases do jogo, a parte azul são os gelos, os quais participam como obstáculos móveis, já que, ao decorrer das ações do personagem rosa, os blocos podem ser construídos ou destruído; a parte rosa refere-se às frutas que devem ser consumidas para vencer o nível presente e, por último, a parte branca é o espaço disponível para ser usado conforme comandos passados pelo dispositivo de entrada, o teclado. Em continuação a essa lógica, para utilizar a imagem, foi preciso convertê-la para o formato “.bmp” e então em arquivo “.data”, utilizando o “bmp2oac3”. Isso tornou possível salvá-la na memória de dados para que os bytes sejam ocupados por esse mapa de colisões. A posição sempre é calculada nesse grid 20x15 da memória, em tiles de tamanho 16x16.

2.4 Fases

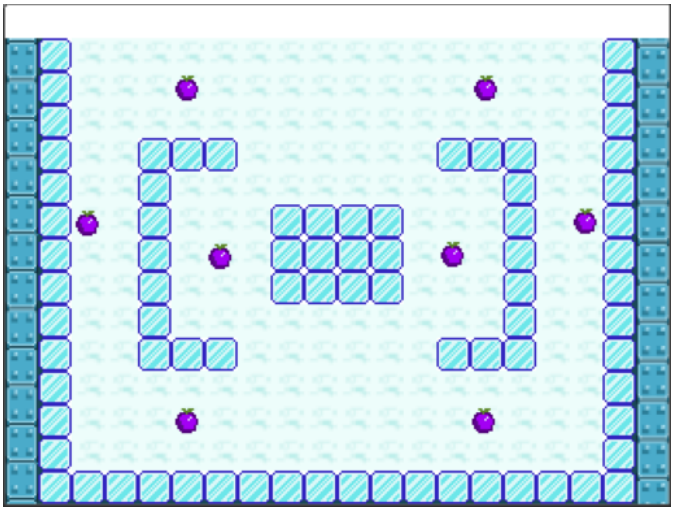


Figura 7: Mapa da fase 1

A fim de facilitar e seguir a dinâmica da versão original, a fase 1 é a mais fácil de ganhar. Nela foi colocada apenas uma fruta, a “blueberry”, e apenas um inimigo, o verde, o qual segue o mesmo percurso até o fim, a menos que seja barrado por um bloco de gelo. Nesse caso, ele retorna na direção oposta que veio. Apesar disso tudo, foi a fase mais trabalhosa de elaborar no jogo, tendo em vista que foi o ambiente que nos permitiu ter as primeiras experiências e tentativas.

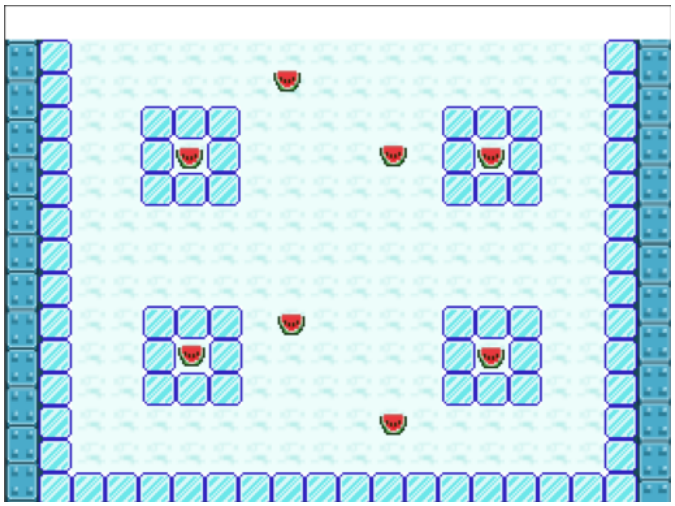
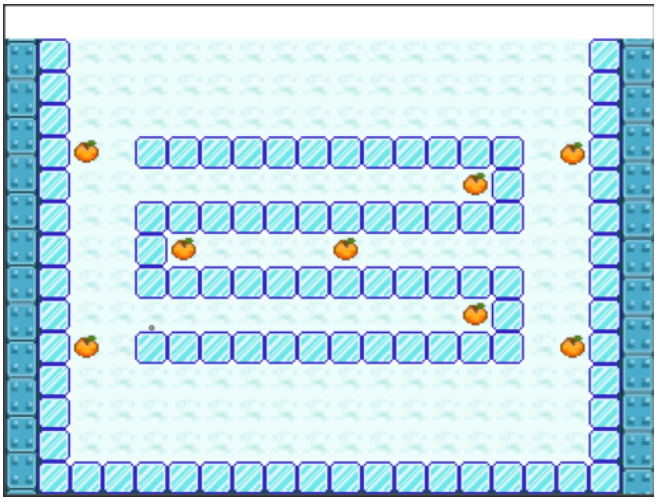


Figura 8: Mapa da fase 2

A fase 2 já foi mais simples e ágil de montar, pois as funções e o códigos base já estavam criados, bastando apenas alterar os parâmetros, sendo eles o do mapa de colisões e a troca do inimigo



utilizado.

Figura 9 : Mapa da fase 3

Nessa fase, foi encontrado um novo erro, o "Branch target word address beyond 12-bit range", causado por uma limitação na plataforma usada, o RARS. Basicamente, o "jump" não consegue acessar as informações pedidas por consequência do código estar grande. Dessa forma, a solução aplicada foi otimizar as estruturas. A mudança essencial foi feita no acesso aos mapas de colisão. Ao invés de declará-los toda vez no começo de cada comando de ação - CHAR_ESQ, por exemplo - , foram declarados os endereços correspondentes já no início de cada fase, a fim de deixar o código mais eficiente, o que evita excessos de instruções dentro de cada função.

2.5 Funções

Funções são um bloco de códigos reutilizáveis que fazem tarefas específicas. Em Assembly eles começam com um rótulo - uma label - seguido das instruções e sempre que for necessário o seu uso, basta chamá-la. Segue as funções utilizadas e suas respectivas funcionalidades:
– CHAR_ESQ, CHAR_DIR, CHAR_CIMA, CHAR_BAIXO: movimentar o personagem, onde a colisão é conferida.
– BLOCO_GELO: analisa a direção e detecta se é para criar ou quebrar bloco.
PRINT e PRINT_LINHA: local no qual as informações de funções print_char e print_inimigo são levadas. Código baseado no vídeo "RISC-V RARS - Renderização Dinâmica no Bitmap Display", por Davi Paturi
– KEY2: verifica se há alguma tecla pressionada e envia para as funções correspondentes.
– TEMPO: responsável por atualizar o tempo a cada loop, caso seja maior que ¼ (um quarto) de segundo, entra no loop da música e do inimigo.
– MÚSICA: executa o toque das notas sequencialmente a cada loop de TEMPO, assim que acaba os acordes é reiniciada a contagem das notas.
– INIMIGO: determina a movimentação do inimigo a partir do loop TEMPO.
Portanto, usar funções permitiu a construção de um código mais rápido e mais fácil de ser executado pelo processador.

2.6 Input

É evidente que esse entretenimento precisa de inputs para rodar e cumprir de fato a sua função. Dessa forma, após estabelecida a fase escolhida, o código entra em SETUP, acessa o SET_MUSICA, seguido do SET_TEMPO, e segue para loop, em que TEMPO e KEY_2 são constantemente acessadas, funções controlam a maior parte do jogo. A partir da label " KEY_2" é feita a leitura do KDMMIO Simulator que faz o personagem protagonista se movimentar caso seja selecionado uma das seguintes teclas: "W", "A", "S", "D" ou "Backspace", as quais significam, respectivamente: andar para cima, para esquerda, para baixo, para direita ou manipular os blocos de gelo.

2.7 Tempo

A fim de determinar algumas ações que ocorreriam de forma automática no jogo, foi decidido criar uma espécie de "timer", que acessasse as partes do código que seriam repetidas ao longo de sua execução Para isso, foi criada a função foi declarado um tempo inicial no começo do código. Dentro do loop do jogo, a função TEMPO é constantemente chamada, e o tempo atual constantemente atualizado. Além disso, é calculada a diferença entre o tempo atual e o inicial para obter o tempo transcorrido. Caso essa diferença for maior 250 milissegundos, então serão executados os loops MÚSICA e INIMIGO. Se o contrário for verdade, o jogo continua normalmente a partir de GAME_LOOP1.

2.8 Música e Efeitos Sonoros

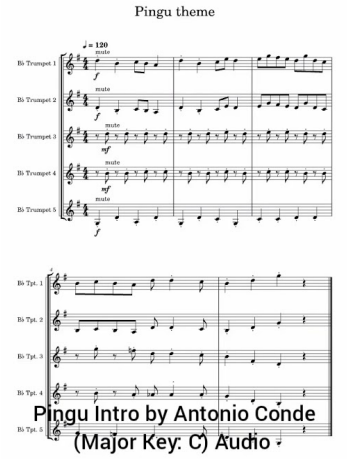


Figura 10: Partitura da música

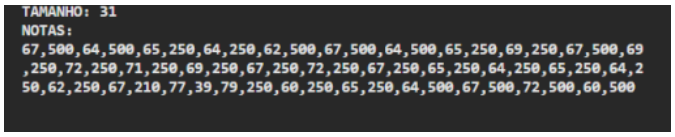


Figura 11: Transcrição no RARS

Escolher a música foi uma parte agradável, Ana, Maria e Eduarda se apoiaram na temática do jogo e no que ela transmite e, assim, escolheram a música "Pingu Intro" por António Conde. Para passar a música para o RARS foi preciso transpor, a partir das partituras, nota por nota, o que foi feito utilizando o HookTheory e o código de conversão "HookTheory to RISCV midi" desenvolvido por Davi Paturi.

No entanto, implementar a música no código, sem que fosse paralisada a execução do programa não foi uma tarefa trivial. Baseado no exemplo "midi.s" da pasta de exemplos do RARS, inicialmente foram utilizadas as syscalls 31,32 e 33. Porém, ao usá-las, foi observado que os demais procedimentos necessários no jogo eram congelados. Assim, foi decidido por integrar o loop da música ao loop do tempo, e utilizar apenas a syscall 31. Desse modo, uma nota é tocada a cada loop do jogo.

O primeiro número contido no endereço "NOTAS" se refere à nota utilizada e o segundo número, à duração da nota. O loop de execução da música carrega ambos os números e as passa como argumento para a chamada de sistema 31 - ou seja, MidiOut -, a qual permite que ocorra a execução simultânea da música e do resto do algoritmo. Para os efeitos sonoros também foi usado o sistema 31, no que tange ao personagem quebrar os gelos ou a comer as frutas, sendo notas solitárias.

3 Resultados Obtidos

Claramente, percebe-se que a versão recém construída é bem semelhante à original, diferenciando na complexidade. Isso, sem dúvidas, está relacionado à pouca disponibilidade de recursos no RARS que impossibilitou realizar maiores ações. Porém, o que estava ao alcance foi concretizado: a criação das fases com o aumento gradativo das dificuldades, a manipulação dos endereços e da memória, a implantação da música, a adesão do tempo e todos os demais aparatos foram inseridos com êxito.

4 Conclusão

Apesar de todo o esforço e trabalho que esse projeto deu às desenvolvedoras, elas afirmam com certeza que foi gratificante ver o resultado, assim como o trabalho em equipe que foi imprescindível. Ademais, descobrir o quão abrangente é a computação e perceber a afinidade de cada uma com um aspecto em específico tornou a elaboração desse projeto mais prazerosa. Fica esclarecido que todas colaboraram em todas as partes, porém vale destacar a parte artística e gráfica à Maria Eduarda, a parte lógica e programável à Maria Salgado e a parte de registrar e documentar à Ana Caroline. Por fim, é necessário relatar que os ensinamentos do educador Lamar foram de suma importância e a sua prática trouxe aprendizados a elas que, certamente, levarão por toda a graduação e até depois disso.

Referências Bibliográficas

[1] MACROSv21.s, Systemv21.s e bmp2oac3, arquivos disponibilizados pelo professor Marcus Lamar.

[2] RISC-V RARS - Renderização dinâmica no Bitmap Display. YouTube, 2021. Disponível em: https://www.youtube.com/watch?v=2BBPNgLP6_s

[3] Bad Ice-Cream - Nitrome, site oficial. Disponível em: https://nitrome.fandom.com/wiki/Bad_Ice-Cream e https://www.nitrome.com/

[4] GazzConecta - Indústria dos games: a mais lucrativa no mundo do entretenimento. Disponível em: https://gazzconecta.com.br/gazz-conecta/papo-raiz/industria-dos-games-mais-lucrativa-mundo-do-entretenimento/

[5] Hooktheory to RISCV midi - Conversor da partitura para a transcrição. Disponível em:https://gist.github.com/davipatury/cc32ee5b5929cb6d1b682d21cd89ae83

[6] Pingu Intro by Antonio Conde - melodia da música. Disponível em:https://www.hooktheory.com/theorytab/view/antonio-conde/pingu-intro

[7] Pingu Intro by Antonio Conde - Partitura por Major Key. C. Disponível em:https://www.youtube.com/watch?v=jz2jQvY67K