Kirby’s Adventure - Grupo 4

Guilherme Lima Barbosa - 232026399 | José Edson Martins Bezerra da Silva - 231003380
Nirva Neves de Macedo - 232009585 | Rodrigo Rafik Menêzes de Moraes - 232009502
Sara Lima Gaspar - 221017121

Universidade de Brasília, Dep. de Ciências da Computação, Brasil

Fig. 1 - Capa do Projeto de Kirby’s Adventure

Resumo

Este relatório tem como objetivo apresentar o desenvolvimento de uma versão do jogo Kirby’s Adventure para SNES, feita em Assembly RISC-V para funcionar em um processador RV32IMF implementado em uma FPGA DE-1-SoC. O projeto foi desenvolvido para colocar em prática os conteúdos ministrados na disciplina de Organização e Arquitetura de Computadores durante o segundo semestre de 2024.

Destacamos detalhadamente os métodos utilizados na implementação do aplicativo, incluindo diversos princípios teóricos e trabalhos que foram relevantes fontes de consulta e possibilitaram o êxito deste projeto.

Palavras-chave: Jogos, Assembly, RISC-V, Kirby, Kirby’s Adventure, FPGA.

1. Introdução

Kirby’s Adventure é um jogo de plataforma que acompanha Kirby, um personagem rosa com a habilidade de absorver os poderes de um inimigo ao ingerí-lo. Nessa aventura, lançada originalmente em 1993 para o Super Nintendo Entertainment System, nosso herói busca derrotar Pesadelo, uma entidade do mal que tem causado problemas em Dream Land. Apesar de o jogo original se distribuir por 7 mundos distintos, cada um deles com suas fases, bosses, minigames e desafios diversos, nossa versão reduzida do jogo acompanha Kirby por um mundo simplificado, com 4 fases que apresentam as principais mecânicas do jogo, como a absorção de poderes do heroi, 3 inimigos com funcionamentos diferentes e uma fase de boss, de forma a funcionar na placa FPGA DE-1-SoC com processador RV32IMF.

Durante o semestre o grupo desenvolveu mecânicas solicitadas pelos monitores para o projeto em Assembly RISC-V, realizando adaptações ao mundo mágico de Dream Land para trabalhar os conceitos desenvolvidos na disciplina de Organização e Arquitetura de Computadores.

2. Fundamentação Teórica e Trabalhos Relacionados

Um sistema de colisão e um sistema de objetos foram adotados com base naqueles do jogo Super Mario 64, dos quais entramos em mais detalhes na seção 3.

Apesar da ausência de chamadas de sistema no processador, considerando que o jogo é implementado sem um sistema operacional subjacente, os arquivos SYSTEMv24.s e MACROSv24.s não foram utilizados por completo, apenas com o mínimo de suas funções que seriam utilizadas sendo importadas para que funcionem na placa FPGA.

Para o sistema de músicas do jogo realizamos a conversão dos áudios coletados do HookTheory utilizando o sistema de conversão criado pelo Davi Paturi, no item 8 da bibliografia. Para a conversão das imagens, utilizamos o projeto bpm2oac3 presente no projeto LAMAR, no item 4 da bibliografia.

3. Metodologia

Ao passo que não é possível implementar exatamente o mesmo jogo, por limitações técnicas e temporais, foi possível recriar boa parte das mecânicas do Kirby. Decidimos não realizar uma releitura devido à falta de tempo e recursos, pois essa personalização dificultaria a implementação de funções mais relevantes ao projeto final do jogo. Utilizamos os materiais sugeridos pelos monitores na definição do projeto para auxiliar esses processos de captura de música, efeitos sonoros e visuais do jogo original.

Alguns sistemas não estão completamente perfeitos, como o sistema de respawn de inimigos que os faz reaparecerem assim que atingem os limites da tela, independente de estarem no campo de visão do jogador. Por causa de limitações na memória, não foi possível

adicionar algumas imagens e sprites, como as do fundo do hub. Algumas animações extras também não foram possíveis pelo mesmo motivo.

3.1 Memória

A disponibilidade de memória no processador utilizado é limitada. Por isso, foram necessárias algumas técnicas de redução de utilização da memória. Entre os dados que mais ocupam espaço, de longe, as imagens são as campeãs. Por exemplo, se utilizássemos uma imagem inteira para representar o fundo do jogo na tela, supondo um mapa 320 × 240, estaríamos ocupando 76808 bytes (considerando ainda as duas words que indicam a altura e largura da imagem). Isso já é uma fração considerável da memória disponível, então não é prático considerando que os níveis não ocupam apenas um tamanho de tela, e que são vários níveis. Para resolver esse problema, utilizamos uma técnica de uso de memória dinâmica, como descrito no ponto 3.5.

O sistema sonoro do jogo é outro exemplo de utilização da memória, onde as músicas e efeitos sonoros utilizados no jogo ficam armazenados em códigos .data, com informações das notas e seu tempo de reprodução, que são utilizados pelo sistema de reprodução de áudio e que é melhor descrito no ponto 3.3. Alguns sistemas menores também afetam o armazenamento, como a implementação de objetos.

3.2 Implementação na Placa

Para fazer o áudio funcionar na placa, precisamos trocar as ecalls de aquisição de tempo pela utilização de um dos registradores de status e controle 3073, utime. Para o áudio em si, como não temos sistema operacional, obtivemos pequenos snippets de código dos arquivos SYSTEMv24.s e MACROSv24.s, com funções que detectam se o jogo está rodando na placa e outras que implementam a funcionalidade de música nela. O jogo roda no processador uniciclo com o divisor de frequência em 4. Tentamos rodar o jogo no processador pipeline, em uma frequência maior, mas não obtivemos sucesso - a primeira frequência estável foi a dividida por 4, e estava processando mais lentamente que o uniciclo.

Conseguimos utilizar buffer de scancodes para leitura do teclado na placa, o que possibilita um movimento muito mais mais fluido, pois diferente do simulador KDMMIO do RARS, que realiza a leitura de inputs do usuário como se fosse a escrita de um texto, considerando apenas uma letra por vez e os delays do teclado, os buffers de scancodes são capazes de detectar diversas teclas pressionadas ao mesmo tempo, já que cada tecla do teclado é representada por um bit dos buffers. Dessa forma, decidimos preparar uma função DE1KeyPress, que junto de duas words de apoio, monitoram as teclas “w”, “a”, “s”, “d”, “e”, “espaço” dos scancodes quando é detectado que o jogo está rodando na placa.

3.3 Temporização

Foi implementado no projeto uma função “Clock” onde é utilizado o registrador utime de status e controle para realizar a busca do horário atual, comparar esse valor com o antigo horário salvo e retornar a diferença entre ambos, já que esse valor representa quantos ms se passaram entre ambos os períodos e é utilizado para o cálculo de frames por segundo (FPS) do projeto, pois após a passagem de 20 ms é “atualizado o frame” (todas as funções de lógica do jogo acontecem uma vez antes do código voltar ao loop do Clock), permitindo que o jogo seja rodado a 50 FPS. Da mesma forma, o sistema de música do jogo utiliza de base o clock do jogo, além de uma função própria, que funciona por meio da leitura de um vetor .data de cada música, contendo os tempos e identificação de cada nota que serão reproduzida pela placa.

```
.data

# Numero de Notas a tocar
# lista de nota,duracao,nota,duracao,nota,duracao,...
DuracaoTitle:   .word 40, # Victory star
NotasTitle:     .half 79,278,79,278,79,278,81,556,79,278,76,278,79,278,84,278,79,834,

DuracaoHub:     .word 174, # Grape garden
NotasHub:       .half 53,183,60,183,67,183,69,183,65,183,48,183,52,183,59,183,66,183,

DuracaoLvl:     .word 88, # Green greens
NotasLvl:       .half 79,249,79,83,84,998,84,249,88,83,91,166,96,166,95,166,93,166,91

DuracaoBoss:    .word 269, # Gourmet race
NotasBoss:      .half 79,323,77,161,76,161,74,161,71,161,67,323,72,161,74,161,76,161,

DuracaoWin:     .word 34, # Kirby wins
NotasWin:       .half 65,120,67,120,69,120,71,120,69,120,71,120,72,240,67,120,64,360,

```

Fig. 2 Músicas codificadas

3.4 Entrada e Saída

Para a entrada de dados está sendo utilizada uma leitura do teclado onde é feita a verificação se alguma tecla foi pressionada e caso tenha ocorrido, é chamada a função da respectiva tecla se for um caso especial, ou a tecla é armazenada na memória, para ser comparada em outras funções do jogo.

A saída de vídeo é feita através do bitmap display, sendo constantemente trocados os seus frames 0 e 1 a cada frame de Clock, de forma que a próxima cena do jogo é renderizada no frame “escondido” do bitmap antes de aparecer para o jogador, o que garante uma visualização fluida do jogo.

Para a saída de som, quando o jogo não está rodando na placa, são utilizados os ecalls de midi já implementados no RARS, enquanto que na placa FPGA, são utilizadas as funções previamente mencionadas. Para a geração dos arquivos .data das músicas foi utilizado o script feito pelo Davi Patury, disponibilizado pelo professor no Git de materiais e no pdf do projeto, que converte músicas do HookTheory para o tipo .data. As músicas que foram definidas para nossa versão de Kirby’s Adventure foram: Green Greens, Victory Star, Kirby Wins, Grape Garden e Gourmet Race.

Como descrito no item 3.3, os sistemas de música e de visuais funcionam em conjunto com o clock, o que assegura que o jogo mantenha uma consistência que melhora a gameplay.

3.5 Renderização

A técnica empregada para a renderização das fases do jogo foi a de dividir a imagem do mapa em "tiles" de tamanho 16×16. Isso permite uma espécie de "compressão" das imagens, considerando que alguns *tiles* podem se repetir e, de fato, se repetem. Um "banco de dados" de cerca de 140 itens foi criado com esses *tiles* e é acessado com um mapeamento das imagens dos mapas baseado em valores de 0 até o último número de *tile*, de forma que podem ser criados sprites coloridos com as mesmas cores que representam esses valores, para que quando convertidos para arquivos .data, eles funcionem como "grids" de valores de tile, que configuram a forma que eles estão sendo chamados na tela. Dessa forma, é possível criar os mapas sem a necessidade de colocar todas as imagens finais dos mapas dos níveis.

Fig. 3 Mapa de Tiles do nível 1 implementado no projeto

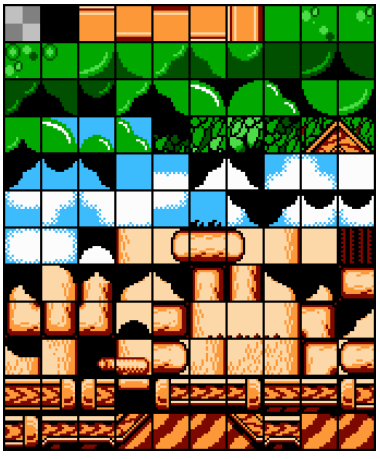

Fig. 4 Gráfico dos tiles de imagem do jogo

A figura 4 mostra os tiles empregados na configuração visual das fases principais do projeto, que quando são aplicados no mapa representado na figura 3 resultam em uma imagem coesa, como a representada abaixo (figura 5), do mapa completo utilizado no nível 1 do projeto.

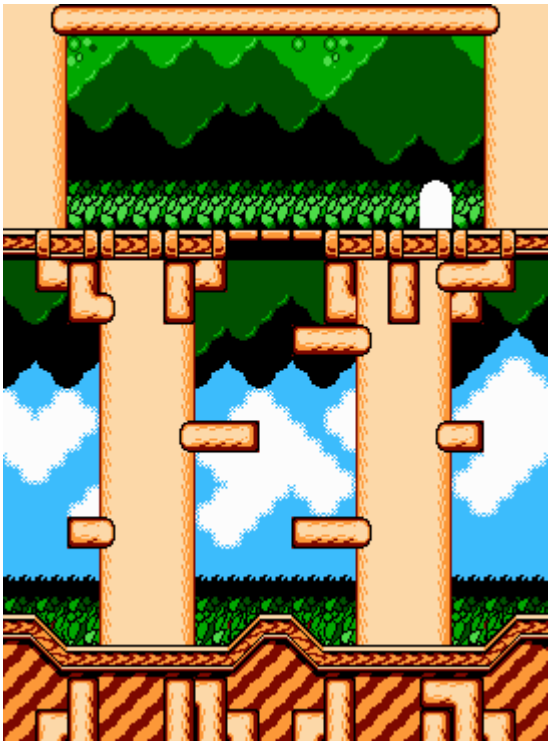

Fig. 5 Mapa implementado do nível 1 do projeto

3.6 Colisões

A colisão do jogo foi feita inspirada na lógica de colisão do Super Mario 64, que possui comportamentos diferentes para casos de colisão do jogador com o ambiente, diferenciando entre colisões com o teto, paredes e chão. Usando esse funcionamento como referência, foi decidido utilizar um mapa de cores para realizar as verificações de colisão, de forma que cada pixel do mapa de aparência tenha um equivalente em um mapa de colisão, e dependendo da cor detectada, o jogador e cada objeto realize diferentes ações/reações.

Para possibilitar isso, foi feito um mapa de colisões para ser carregado "por baixo" do mapa impresso na tela. Inicialmente esse mapa inteiro era carregado na memória para ser feito a análise de colisões, entretanto, por limitações na memória da DE1, isso foi reduzido para uma área de 32 pixels que é criada ao redor do personagem principal e dos oponentes com base nas suas posições.

A análise de colisão é feita através de cores associadas a cada entidade, logo, quando é detectado que duas cores estão se sobrepondo no mapa de colisões, o projeto analisa como essas cores devem interagir, resultando nas ações consequentes desta colisão. Cada cor representa uma forma de interação diferente do personagem com os objetos em cena, como os dois quadradinhos na parte inferior da porta, responsáveis pela transição entre as telas e acesso aos níveis, ou as cores verdes, azul e

vermelha, que representam, respectivamente o chão, as paredes e os tetos do jogo.

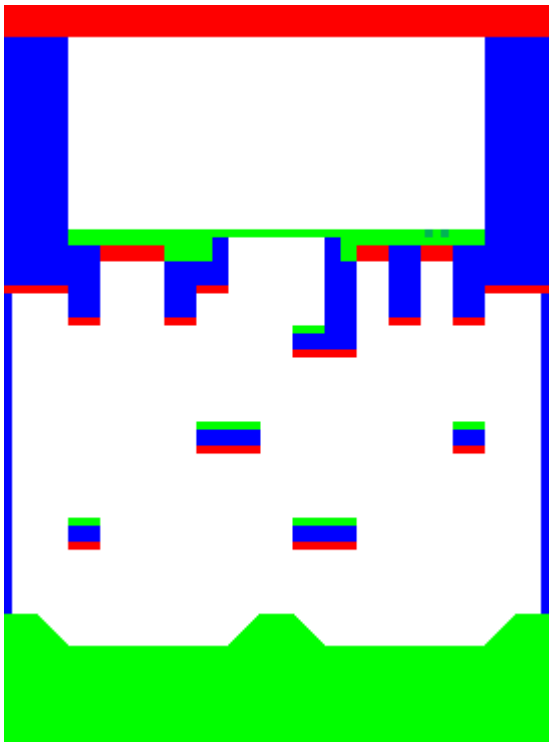

Fig. 6 Mapa de colisão representado por cores

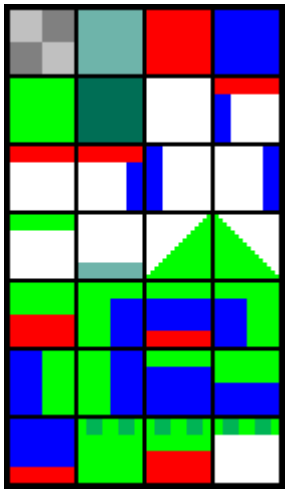

Fig. 7 Gráfico dos Tiles de colisão configurados

O sistema de colisão com o mapa foi implementado de forma similar aos mapas de Tiles, onde o gráfico apresentado na figura 7 representa as possibilidades de colisão em 16x16, que serão aplicadas em um mapa como o da figura número 6 com base em um grid que nem o da imagem 8, sendo que após ser chamada a função “UpdateCollision”, que recebe um endereço representativo do jogador ou dos objetos, atualiza o mapa 32x32 de colisão com base na posição deles, para então ser processado e fazê-los reagirem como necessário.

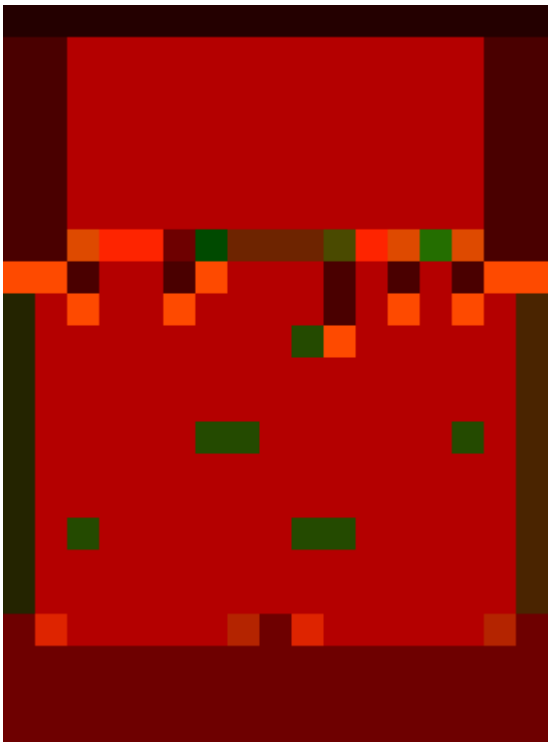

Fig. 8 Grid de colisão do nível 1 do projeto

3.7 Inimigos

Responsáveis pelas principais dinâmicas do jogo, os inimigos são fundamentais para a aventura e para que o Kirby possa interagir com o jogo em toda sua glória. Elegemos 3 tipos básicos de inimigos que aparecem pelos mapas, cada um com suas individualidades de ataques e movimentação, mas todos causam danos ao encostar no personagem, a menos que estejam sendo ingeridos. Além disso, na última fase Kirby se encontra com Whispy Woods, o boss do primeiro mundo de Kirby’s Adventure.

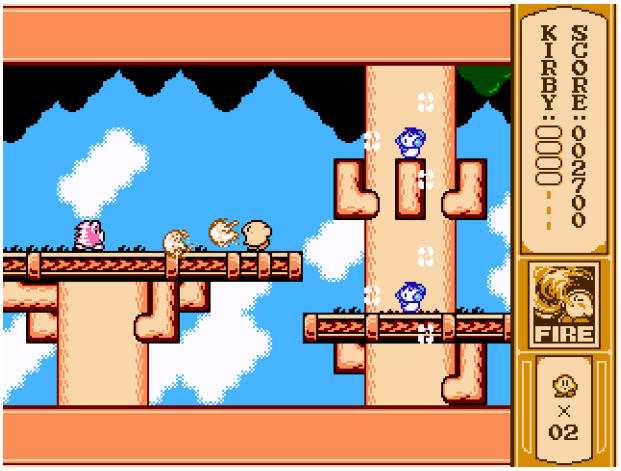

Fig. 9 Kirby implementado no projeto com o poder de Hot Head. Chilly é visto nas plataformas ao lado

Hot Head interage com Kirby lançando fogo como ataque, se movimenta andando lateralmente em uma plataforma e, ao ingerí-los, 300 pontos são recebidos para o score e Kirby adquire o poder de fogo e fica com a cor laranja, como visto na figura número 9.

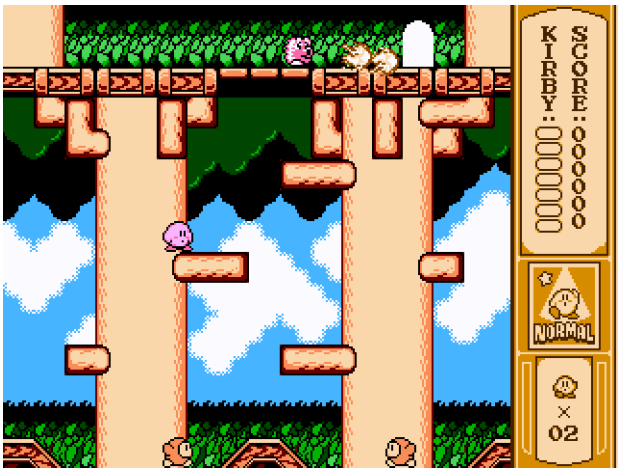

Fig. 10 Kirby, Hot Head na plataforma superior e Waddle Dees na plataforma inferior

Waddle Dee interage com Kirby apenas andando e não ataca, causando dano somente ao bater no personagem. Eles andam sempre na direção do jogador e é capaz de cair de plataformas. Ao ingerí-los, Kirby não ganha habilidade especial nem muda de cor, mas 150 pontos são somados ao score e ele pode cuspir o Waddle Dee como estrela em forma de ataque.

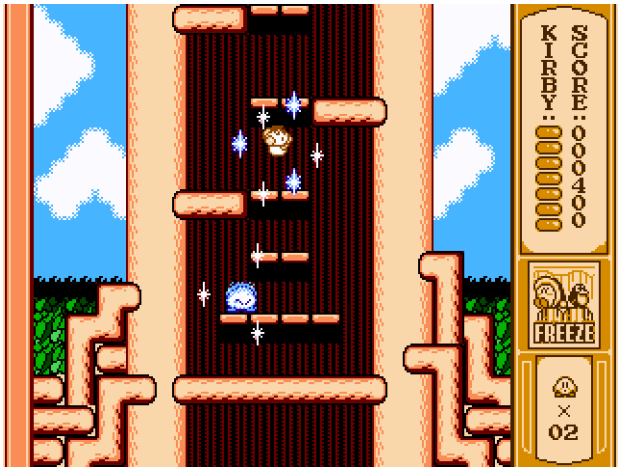

Fig. 11 Kirby usando o poder congelante de Chilly no projeto

Chilly interage com Kirby soltando múltiplos raios de gelo em um espaço ao seu redor como forma de ataque, fica parado em um mesmo lugar, mas ocasionalmente pula e, ao ingerí-los, outros 300 pontos são ganhos e Kirby adquire o poder de gelo, também atirando raios de gelo aleatórios ao seu redor e fica com a cor azul.

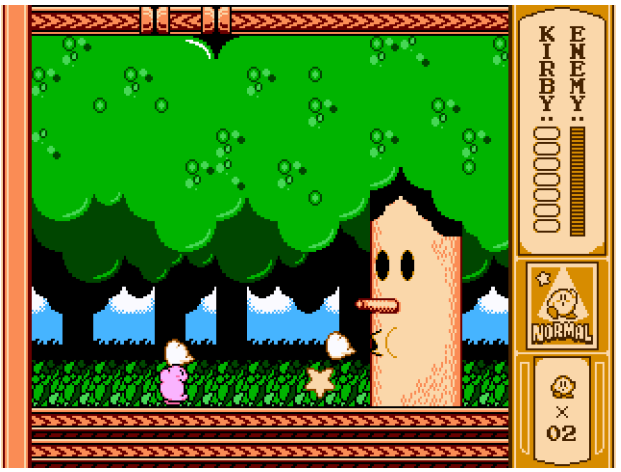

Fig. 12 Kirby e Whispy, o Boss do primeiro mundo, no projeto

Whispy Woods é o único Boss implementado na nossa versão de Kirby's Adventure, interage com Kirby realizando ataques de sopro em sua direção e fazendo Waddle Dees caírem de suas folhas. Por ser uma árvore, ele não se movimenta além de olhar para os lados é impossível de ser ingerido, sendo que Kirby não pode ingerí-lo mas pode subir em seu nariz. Para derrotar o boss, Kirby precisa usar seus poderes, ataques de vento ou de estrela para machucá-lo, podendo usar os Waddle Dees que aparecem para atacá-lo mais eficientemente se estiver sem poderes.

3.8 Sistema de objetos

Para que todos os diferentes tipos de inimigos e ataques do jogo pudessem ser gerenciados e padronizados,foi desenvolvido um sistema de objetos, no qual alguns espaços da memória foram reservados com "variáveis" padrões para armazenarem valores importantes para os objetos.

No estado final do jogo existiam 20 espaços para objetos, cada um ocupando 20 bytes com informações de ID, posição, direção, status, tempo de vida e de apoio, sendo que todos eles são construídos ao ser chamada a função "BuildObject" junto de um valor de ID e outras informações de suporte, o que faz uma instância do objeto em questão ser criada no banco de dados deles se houver espaço. Após isso, a função "DrawObjects" do loop principal do jogo analisa todos os espaços de objeto e processa todos que estiverem ativos, realizando suas ações específicas e os desenhando no jogo

3.9 Funcionamento do jogador

Em resumo, o jogador funciona com base em uma máquina de estados bastante complexa, de forma que a maior parte de seus estados são definidos na função "PlayerControls", que analisa os botões de movimento e ataque sendo pressionados e suas colisões para definir variáveis como "PlayerPowState" e "PlayerAnimState", as quais são posteriormente analisadas em outras partes do código para realizar certas interações, mas principalmente pela "PlayerAnimation", que define a animação e sprite atual do jogador, para enfim ele ser desenhado na tela pela função "DrawPlayer".

3.10 Outros métodos

Para apoio de diversas funções do código e impressão do jogo no bitmap, foram criadas as funções:
- “Print”, que realiza a impressão de sprites com base na posição relativa deles na tela, analisando se com base nas suas coordenadas ele apareceria fora da tela ou não, evitando erros de memória. Sendo que ela também pode receber valores para indicar a direção esquerda ou direita do sprite e se algum filtro de cor deve ser aplicado;
- “PrintMapa”, que utiliza o offset determinado pela posição do jogador para mover e imprimir o mapa de maneira correta na tela, possuindo casos específicos para as colunas e linha máximas e mínimas dele para evitar problemas;
- “SimplePrint”, uma função que apenas recebe um sprite e uma posição do bitmap para desenhar esse sprite na tela, sendo usada manualmente diversas vezes para desenhar telas que nem o título, menu e finalização;
- “PrintFill”, que recebe um valor de cor, uma posição do bitmap, um valor de largura e um de altura e um valor que indica se a impressão estará alinhada com a memória para desenhar um bloco de cor sólida no bitmap display. Essa função foi bastante utilizada junto com a de “SimplePrint”, também para desenhar certas telas do jogo.

4. Resultados Obtidos

Esta versão, embora mais simples que o jogo original, simula bem várias mecânicas dele. Nossas maiores dificuldades foram o prazo para desenvolver o projeto e as limitações do RARS, especialmente aquelas relacionadas ao tamanho do código ao nos aproximarmos do fim do projeto. Apesar disso, o jogo tem boa performance e oferece uma experiência satisfatória ao jogador.

Conseguimos implementar de forma satisfatória sistemas gerais de usabilidade, como reprodução de áudio e vídeo, sistema de colisão, interação com objetos diversos, menu de usuário com informações do personagem, etc., realizando as adaptações necessárias para rodar sem grandes problemas em uma placa FPGA DE-1-SoC. Dessa maneira, o resultado final do nosso projeto se apresenta suficiente para simular o original em condições reduzidas de memória e desempenho, demonstrando o aprendizado das questões fundamentais de OAC, como configurações de temporização para otimização do programa, por exemplo.

4.1 O Jogo, níveis e mecânicas

Ao iniciar o arquivo do jogo é aberta uma tela inicial que apresenta o título do jogo e ao apertar espaço ou a tecla “e” você será redirecionado para o jogo. No hub, nós escolhemos desenvolver um menu semelhante ao do jogo original onde é possível se movimentar para escolher a fase. escolhemos fazer a fase da floresta que é, semelhante ao jogo, composta por 4 estágios além da zona do boss.

4.2 Diferenças em Relação ao Original

Infelizmente com as limitações temporais e do hardware que deve ser utilizado não é possível fazer algo tão extenso e complexo quanto o jogo original, fazendo nos limitarmos a uma área reduzida do todo. Deixamos de fazer diversas das fases existentes, como também sofremos limitações com relação à parte sonora do jogo, pois a limitação de 127 instrumentos básicos do midi, ou o único instrumento da placa, e a falta de uma forma simples de aplicar mais de uma voz, nos fizeram utilizar apenas melodias e efeitos sonoros simples.

4.3 Semelhanças em Relação ao Original

Escolhemos fazer uma recriação do jogo sem uma temática diferente, logo o jogo possui diversas semelhanças com o original. Dentre elas estão os sprites que tiramos diretamente do jogo, a maioria das animações do Kirby, a barra de menu com com o score e vida do jogador, que funciona de forma semelhante à do original, mas que difere dele ao estar na vertical e na direita ao invés de na parte inferior, sendo que foi feita dessa forma para corrigir a diferença da altura e largura do bitmap display e tornando a área da tela do jogo mais próxima a um quadrado. Outras semelhanças são os inimigos, que como mencionado anteriormente, são alguns que existem no jogo original e possuem interações semelhantes a desses oponentes, fazendo as habilidades de cópia do Kirby também serem similares.

5. Conclusão

O projeto foi bem-sucedido e proveitoso para os integrantes do grupo, expandindo nossos conhecimentos em programação em uma linguagem de baixo nível com a qual não éramos muito familiarizados, trabalhando nossa criatividade em solucionar problemas e habilidade de cooperar para desenvolver um projeto em equipe. As dificuldades enfrentadas no desenvolvimento do jogo nos fizeram pesquisar mais e conhecer novos recursos e metodologias até então inexploradas nas outras matérias do curso, e essa experiência certamente se mostrará útil no futuro da nossa vida acadêmica e profissional.

Fig. 13 Tela final do projeto com Kirby dando tchau

Referências Bibliográficas

[1] HookTheory. Disponível em: https://www.hooktheory.com>

[2] Kirby's Adventure. Wikipedia, 2024. Disponível em:<https://pt.wikipedia.org/wiki/Kirby%27s_Adventure>

[3] Kirby's Adventure (NES), 2011. Disponível em: <https://shugames.blogspot.com/2011/03/kirbys-adventure-nes.html>

[4] Lisboa, Victor. LAMAR - Learning Assembly for Machine Architecture and RISC-V. Disponível em: <https://github.com/victorlisboa/LAMAR?tab=readme-ov-file>

[5] MACROSv21.s, Systemv21.s e bmp2oac2 foram arquivos disponibilizados pelo professor Marcus Lamar durante o semestre, que foram utilizados no projeto.

[6] NES Longplay [063] Kirby's Adventure, 2010. Disponível em: <https://www.youtube.com/watch?v=rJXM4EPbPe0>

[7] NES - Kirby's Adventure - The Spriters Resource. Disponíveis em: <https://www.spriters-resource.com/nes/kirbyadv/>

[8] Patury, Davi. HookTheory to RIC-V midi. Disponível em: <https://gist.github.com/davipatury/cc32ee5b5929cb6d1b682d21cd89ae83>

[9] SM64's Invisible Walls Explained Once and for All, 2024. Disponível em: <https://www.youtube.com/watch?si=k8d3rlfTtiAlLK9W&v=YsXCVsDFiXA&feature=youtu.be>

[10] WIKIRBY. Disponível em: <https://wikirby.com/wiki/>