UNIVERSIDADE DE BRASÍLIA

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Disciplina: CIC0003 - Introdução aos Sistemas Computacionais - Turma 01 - 2024/2

Professor: Marcus Vinicius Lamar

Nome: Enzo Cardoso Martins
Nome: Caio Dias Fleury
Nome: Kaio Santos Araújo

PROJETO APLICATIVO
-BANANA OWNER-

Resumo:

O projeto descrito neste artigo é uma releitura do jogo fictício *Fix-It Felix Jr.*, desenvolvido pela produtora norte-americana Disney e lançado em 2012. O jogo Banana Owner foi programado na linguagem Assembly para a arquitetura RV32IM por meio do software RARS (*RISC-V Assembler and Runtime Simulator*). Neste artigo, será descrita a ideia do jogo original, metodologia de programação, desafios encontrados no desenvolvimento e o resultado final do projeto.

1. Introdução

O jogo *Fix-It Felix Jr.* foi criado como um jogo fictício dentro do universo do filme *Detona Ralph* (2012), da Disney. No filme, ele é apresentado como um clássico *arcade* dos anos 80, inspirado em títulos como *Donkey Kong* e *Rampage*. A mecânica do jogo gira em torno de Ralph, o antagonista que destrói as janelas de um prédio enquanto lança tijolos no jogador, enquanto Félix Jr., o protagonista, usa seu martelo mágico para consertar os danos causados enquanto desvia dos projéteis.

Figura 1: Início do jogo original

2. Desenvolvimento

Inicialmente o projeto foi desenvolvido usando exclusivamente o RARS como simulador RISC-V e ambiente para testes. Com o avançar do projeto, notamos algumas limitações da forma com que o RARS executava os códigos, impossibilitando-o de rodar normalmente o projeto. Para contornar esse problema, mudamos para um outro simulador RISC-V: o FPGRARS. Criado pelo estudante de Ciências da Computação da Universidade de Brasília, Leo Riether, o FPGRARS foi essencial para o desenvolvimento do projeto, permitindo sua execução fluída e dinâmica, assim, se tornando o principal meio de executar o projeto.

Figura 2: Tela de início de jogo

Por decisão conjunta do grupo, optamos por fazer o tema do jogo ambientado em um universo de macacos em uma fazenda, onde a história do jogo aborda o conflito por terras entre fazendeiros e os povos nativos.

Por ser uma releitura do jogo Fix-It Felix Jr. precisávamos de algumas mecânicas essenciais para cumprir nosso objeto. Dentre elas estão: um inimigo principal que se movimenta enquanto lança projéteis que, ao atingirem o personagem, aplica uma punição neste (no caso, a perda de uma “vida”); um sistema de “consertar janelas”, que, essencialmente, descreve o ato de o personagem, em uma região específica, realizar uma ação que faça com que essa esta seja atualizada; um inimigo secundário que frequentemente atravessa o mapa do jogo horizontalmente e, caso atinja o personagem, aplique-o uma punição; e um *power-up* que surge aleatoriamente na tela e, casso o personagem interaja com ele, ficará invulnerável por um certo período de tempo.

2.1. Metodologia

O desenvolvimento foi realizado de maneira com que o grupo sempre trabalhasse em conjunto, estando sempre se comunicando sobre as atualizações do desenvolvimento, mantendo a dinâmica de trabalho em equipe e dando ideias e opiniões sobre o projeto. Para tal organização, um repositório no Github foi criado, melhorando a eficiência e evitando conflitos entre versões diferentes do código. Portanto sempre que uma atualização era realizada, era enviada para o repositório do Github, tendo inclusive diversas *branches* responsáveis por partes específicas do projeto.

Figura 3: Mapa do jogo

2.2. Protagonista

O sprite do protagonista do projeto, de dimensões de 16x16 pixels, foi criado usando o Piskel, uma plataforma de editor de sprites. Para ser desenhado na tela, o sprite é **traduzido** para bytes de cor, que são armazenados, junto com o seu tamanho, em um arquivo .data bruto. Depois de traduzido, ele é carregado para o jogo utilizando o Renderizador, sendo que, após cada movimento realizado pelo jogador, seu sprite é modificado, e translocado byte a byte para a determinada direção, criando assim a animação de movimento do personagem.

2.3. Inimigos

Temos dois principais inimigos no jogo: um macaco indígena que, a cada dois segundos, aparecerá em uma das moitas na parte superior da plantação, e atirará flechas contra o protagonista; e o outro inimigo é um mosquito que avança horizontalmente por toda a plantação a cada dez segundos. Caso um dos inimigos se colida com o protagonista, será removido uma de suas vidas. Se estas vidas chegarem a zero, será o fim do jogo, tendo que partir do início novamente.

Figura 4: Tela de fim de jogo

2.4. Power-up

Há apenas um *power-up* no jogo: uma banana que, caso coletada pelo protagonista, o deixará invulnerável por dez segundos. Ela surgirá em um local aleatório do mapa a cada trinta segundos.

2.5. Colisões

As colisões do projeto foram feitas por meio de um **mapa de cores** que fica abaixo de todos os sprites visíveis. Caso seja branco, significa que não há colisão e o protagonista pode atravessar. E se a cor for ciano (especificamente a cor 147 no código hexadecimal da versão *SystemRARS* do professor Lamar), significa uma **colisão**, ou seja, o protagonista não pode ultrapassar aquele ponto.

Além dessas duas cores, cada plantação específica tem sua cor única. Desse modo, caso queiramos saber qual plantação o jogador está regando, basta verificarmos sua cor.

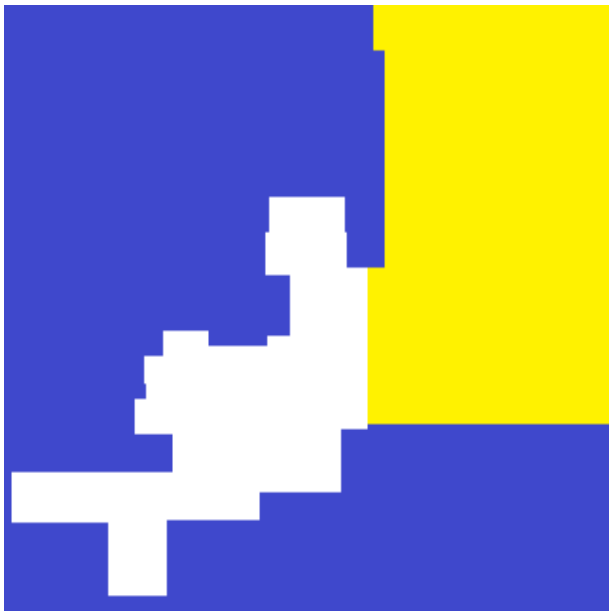

Figura 5: Colisões do primeiro mapa

Figura 6: Colisões do mapa principal

2.6. Mecânica Principal

Diferente do jogo original, neste jogo seu objetivo é **plantar o máximo de bananas possível**, dentro do prazo estabelecido de 140 (cento e quarenta) segundos. Para isso, o jogador tem que regar três vezes a mesma plantação, tendo um intervalo de quatro segundos para cada ação que realizar em um mesmo local. Quando uma plantação é regada três vezes, ela poderá ser coletada, incrementando contador de bananas no placar no canto superior direito da tela.

Após o segundo nível, começarão a aparecer cercas aleatoriamente no cenário, que complicarão a movimentação do jogador. Assim como todo o projeto, seu sprite também foi desenhado pelo editor Piskel.

2.7. Gamepad e Mapeamento de Teclas

Devido a limitações do emulador utilizado, não é possível pressionar mais de uma tecla por vez, sendo, portanto, impossível o movimento diagonal do protagonista. Para contornar esse problema, criamos um programa em C++ que, com o auxílio da API DirectInput da Microsoft, captura o *input* do gamepad ou do teclado e os traduz para outras teclas que servirão como o movimento diagonal. Por exemplo, caso movimentamos o analógico esquerdo para a diagonal superior direita, será pressionado a tecla P, que o programa reconhece como essa direção, e assim será para todas as direções possíveis.

Caso não seja reconhecido um controle, o programa mapeia o *input* do teclado, e identifica que, se duas teclas específicas estão sendo pressionadas, digitará uma tecla específica a aquela direção.

Os sprites usados no jogo foram todos feitos a mão usando o paint, e as artes do início do jogo, história e game over feitas pela IA ChatGPT.

Implementamos também o suporte a controle, onde era utilizado um código em *C++* para converter os botões pressionados do controle em teclas de teclado, permitindo assim a movimentação do personagem tanto pelo teclado como pelo controle.

Para a colisão usamos um mapa de colisão, onde era verificada a cor onde o personagem esta, e se fosse uma determinada cor o personagem não poderia seguir adiante. Usamos um método semelhante para fazer o sistema de irrigação das plantas, onde cada área que representa uma planta tem uma cor e se o personagem estiver na mesma posição dessa cor apertar a tecla “E” ela é atualizada para o estado seguinte.

3. Dificuldades

Por ser um projeto relativamente complexo e em uma linguagem de extremo baixo nível e nova para nós, diversas dificuldades foram encontradas ao longo do desenvolvimento. A seguir serão listadas as principais dificuldades encontradas.

3.1. Renderização

Inicialmente, utilizávamos um sistema de renderização onde a tela era atualizada por inteira, frame por frame. Ou seja, a cada movimento de um sprite, toda a tela precisava ser completamente renderizada. No entanto, esse método apresentou diversas limitações, tornando inviável sua aplicação no jogo. Para superar essas restrições, desenvolvemos um novo sistema de renderização baseado em camadas. Nesse método, o código fornece funções específicas para carregar, descarregar e renderizar imagens em uma estrutura de múltiplas camadas sobrepostas.

As funções LoadImage e UnloadImage são responsáveis por gerenciar essas camadas, que consistem em um conjunto de dados com tamanho igual a onze vezes a resolução da tela (320x240 bytes). Isso permite armazenar múltiplos elementos gráficos sem precisar renderizar completamente a tela a cada atualização.

Uma vez que as imagens já estão carregadas nas camadas, a função Renderizador é chamada. Essa função transfere os dados das camadas para a região de memória do bitmap display, seguindo um critério específico: primeiro, verifica o byte da camada superior; se esse byte for igual a 199 (ou, no caso do RARS modificado, transparente), ele ignora essa camada e verifica a seguinte. Se não, o valor é colocado diretamente na tela, garantindo que apenas os elementos visíveis sejam desenhados. Esse novo método de renderização otimizou o desempenho, permitindo uma renderização mais eficiente, fluída e flexível.

3.2. Música e Syscall

O código da música foi criado logo no começo do desenvolvimento do projeto antes mesmo de migrarmos do RARS para o FPGRARS. Na época tivemos dificuldade com a música graças a lenta velocidade de processamento do RARS, com isso, a música apresentava inconsistências na velocidade entre as notas. Esse problema foi resolvido quando trocamos o emulador para o FPGRARS, que, graças a sua maior eficiência de processamento, conseguiu acompanhar e acabar com as inconsistências entre as notas.

Porém com essa migração outro problema foi encontrado, após um certo número de *syscalls* de *time* realizado, o FPGRARS *crashava*. Para contornar o problema, pensamos em aumentar o tempo entre cada instrução, entretanto com isso o jogo inteiro era afetado por esse acréscimo, e não consertava o problema. Após conversas com os monitores da matéria, descobrimos que esse era um problema da versão do FPGRARS que estávamos utilizando, cujo máxima quantidade de *syscalls* suportados era pouco mais de 8000. Após essa descoberta, por recomendação de nossos monitores, mudamos para uma versão menos atualizada do FPGRARS que ainda não possuía esse problema. Com isso, o problema com a música foi finalmente resolvido.

4. Conclusão

Concluímos o projeto com extrema satisfação por termos conseguido implementar todas as mecânicas propostas e ainda mecânicas adicionais para deixar o jogo mais funcional e atraente. Esse projeto foi desafiador por ser desenvolvido em uma linguagem de baixo nível e que era estranha para todo o grupo, onde cada funcionalidade por mais simples que fosse demandava uma grande quantidade de tempo e esforço. Porém, nos proporcionou uma ótima experiência e aprendizado, aprofundando assim, o conhecimento em otimização e eficiência do código.

Além da parte técnica, aprimoramos nossa experiência em trabalho em equipe e organização, visto que por ser um projeto extenso e em grupo, demandava muita organização e comunicação de cada membro da equipe.

Em suma, concluímos o projeto fazendo um jogo funcional, visualmente bonito e com todas as mecânicas do jogo original e implementando mecânicas e ideias novas. Apesar das dificuldades e desafios encontrados ao longo do caminho estamos extremamente satisfeitos com o resultado final tanto pela parte técnica tanto pelas experiências vividas e pelo conhecimentos adquiridos.

Figura 7: Todos os sprites presentes em uma tela

Figura 8: Tela final de cada fase

5. Referências

1. **Jogo Original do Fix-It Felix Jr.**
   https://online.oldgames.sk/play/genesis/fix-it-felix-jr/10107

2. **Repositório do projeto Github**
   https://github.com/CaiodFleury/ProjetoISC-24-2

3. **Repositório LAMAR Github**
   https://github.com/victorlisboa/LAMAR

4. **Playlist do Youtube de funções básicas do RARS**
   https://www.youtube.com/playlist?list=PLL0Kob75DU32afhLBN5nY2KzOJ5k6lw-Q

5. **Material didático de ISC no Aprender**