Aluno: Gabriel da Conceição Miranda
Matrícula: 190106921
Flappy Bird para 2 Jogadores na Placa DE1-SoC

Jogabilidade

A meta é sobreviver o maior tempo possível, acumulando pontos a cada cano ultrapassado com sucesso. Qualquer colisão, seja com os próprios canos ou com os limites superior e inferior da tela, encerra a tentativa do jogador.

A jogabilidade central é controlada pelos botões da placa. O jogador utiliza o KEY1 para dar um impulso para cima ao pássaro amarelo do Jogador 1, enquanto o KEY2 comanda o pássaro vermelho do Jogador 2, caso o modo de dois jogadores esteja ativo. Para encerrar a aplicação a qualquer momento, o jogador pode pressionar o botão KEY0, que finaliza o jogo de forma segura, liberando todos os recursos de hardware.

O design é intuitivo, com um switch na posição para baixo, a configuração é mais fácil, e para cima, mais difícil. Os controles mestres são o SW9, que funciona como uma pausa universal, congelando toda a ação e exibindo um ícone na tela para indicar o estado, e o SW8, que alterna entre o modo de um jogador e o modo competitivo para dois jogadores, ativando ou desativando o segundo pássaro.

Os outros switches permitem um ajuste fino de todos os parâmetros de dificuldade. A dificuldade de controlar o pássaro pode ser modificada pelo SW7, que altera o tamanho do pássaro, tornando-o um alvo maior no modo difícil; pelo SW6, que define a força do pulo, resultando em um 'flap' mais forte no modo difícil; e pelo SW5, que controla a intensidade da gravidade, fazendo o pássaro parecer mais 'leve'. O desafio do cenário também é totalmente customizável. O SW4 oferece a mudança mais drástica: no modo fácil, dois canos aparecem com espaçamento relativamente grande, mas no modo difícil, a tela é preenchida com três canos muito mais próximos. A dificuldade pode ser ainda mais refinada com os switches SW2 e SW3, que juntos oferecem quatro níveis para a altura da abertura entre os canos, e com os switches SW0 e SW1, que também em conjunto, definem a velocidade de avanço dos obstáculos em quatro incrementos.

O sistema de pontuação é duplo. Na tela principal da VGA, um placar no canto superior direito exibe a soma dos pontos da rodada atual para os jogadores que continuam ativos, funcionando como um indicador de progresso. Um ponto é concedido no momento exato em que um jogador consegue atravessar um obstáculo com sucesso. Em paralelo, os displays físicos de 7 segmentos na placa mostram os recordes persistentes, ou 'high scores'. Os displays HEX1 e HEX0 são dedicados ao recorde do Jogador 1, enquanto os HEX5 e HEX4 mostram o do Jogador 2. Estes recordes só são atualizados no final de uma partida, caso o jogador tenha superado sua marca anterior.

O fim da partida ocorre quando as condições de colisão são atendidas. No modo de um jogador, o jogo termina assim que o Jogador 1 colide. Já no modo para dois jogadores, a partida continua até que ambos os jogadores tenham sido eliminados, adicionando um elemento estratégico. A partida pode ser reiniciada a qualquer momento pressionando

KEY1 ou KEY2. Uma característica importante é que o jogo, ao reiniciar, aplica imediatamente todas as configurações de dificuldade selecionadas nos switches, permitindo que os jogadores ajustem o desafio dinamicamente entre as rodadas sem precisar reiniciar o programa.

Fluxo de Execução do Programa

A execução do programa começa com uma fase de preparação e inicialização. O primeiro passo é estabelecer a comunicação com o hardware da placa através da função init_hardware(). Se esta etapa falhar, o programa é encerrado imediatamente, pois não há como prosseguir. Em seguida, uma área de memória, o back_buffer, é alocada dinamicamente; este buffer é a peça central da técnica de double buffering, que garante que os gráficos sejam desenhados em um "quadro escondido" para depois serem exibidos de uma só vez, eliminando qualquer tremulação na tela. Após inicializar o gerador de números aleatórios para garantir a imprevisibilidade dos obstáculos, o programa declara todas as variáveis de estado necessárias, como as que controlam os jogadores, os obstáculos e os placares. Finalmente, antes de iniciar o jogo de fato, ele realiza um reset inicial, lendo a configuração do modo de 1 ou 2 jogadores e chamando a função reset_game com parâmetros de dificuldade padrão para criar a primeira tela de forma consistente e previsível.

Com tudo preparado, o programa entra em seu coração operacional: o loop principal infinito (while (1)). Cada iteração completa deste loop representa um único quadro (frame) do jogo. A primeira ação dentro de cada quadro é ler o estado atual de todos os periféricos de entrada: os botões (KEYs) e todos os 10 interruptores (switches).

Imediatamente após a leitura, ocorre a fase de análise dos controles e adaptação da dificuldade. O valor inteiro lido dos switches é sistematicamente decodificado usando operações de bit a bit (bitwise operations). O programa verifica switches específicos para ativar ou desativar modos de jogo, como o modo de 2 jogadores (SW8) e a pausa (SW9). Simultaneamente, ele analisa grupos de switches para traduzir suas combinações em parâmetros de jogo concretos para aquele quadro. A posição dos switches SW0 a SW7 define dinamicamente a velocidade dos canos, o tamanho da abertura entre eles, o espaçamento horizontal, a força da gravidade, a potência do pulo e até o tamanho do pássaro. Esta seção funciona como um "tradutor" em tempo real, convertendo as configurações físicas da placa em variáveis que ditam o comportamento e o desafio do jogo no momento.

O fluxo então entra na máquina de estados principal (switch(state)), que determina qual das duas lógicas centrais deve ser executada: GAME_RUNNING ou GAME_OVER. No estado GAME_RUNNING, a primeira verificação é se o jogo está pausado (via SW9). Se não estiver, a progressão do jogo acontece: o programa processa os pulos dos jogadores, aplica a física da gravidade para atualizar suas posições verticais, move os obstáculos horizontalmente, verifica se algum obstáculo saiu da tela para "reciclá-lo" no final da fila, e por fim, executa a rotina de detecção de colisão. Se uma colisão resulta na morte de todos os jogadores ativos, o estado do jogo é alterado para GAME_OVER. Independentemente de estar pausado ou não, a rotina de desenho é sempre executada. Ela desenha o cenário, os

canos e os pássaros no back_buffer, adiciona o símbolo de pausa se necessário, e finaliza copiando a imagem completa para a tela, garantindo uma animação fluida.

Caso o estado do jogo seja GAME_OVER, a lógica é muito mais simples. O programa entra em um modo passivo, onde a física e o movimento estão congelados. Ele apenas aguarda que o jogador pressione uma das teclas de reinício. Ao detectar o pressionamento, ele chama a função reset_game, passando os parâmetros de dificuldade atuais lidos dos switches, e altera o estado de volta para GAME_RUNNING, iniciando uma nova partida instantaneamente com a dificuldade selecionada. Ao final de cada iteração do loop principal, a função usleep(16666) é chamada, criando uma pequena pausa que limita a execução a aproximadamente 60 quadros por segundo, o que garante uma velocidade de jogo consistente e evita o uso desnecessário de 100% da CPU. Este ciclo se repete até que a tecla de saída (KEY0) seja pressionada, quebrando o loop e permitindo que o programa prossiga para sua finalização, onde a memória alocada para o back_buffer é liberada.

Flickering

Durante o desenvolvimento inicial deste projeto, foi identificado um problema visual de cintilação, ou "flickering", que comprometia a experiência de jogo. A causa deste erro estava na abordagem original da minha lógica de desenho, que operava de forma direta sobre a memória de vídeo visível (o framebuffer). A cada quadro, o programa primeiro limpava a tela inteira com a cor de fundo para, em seguida, redesenhar sequencialmente cada elemento do jogo, como os canos e os pássaros. O conflito surgia porque o controlador de vídeo da placa lê essa mesma memória de forma contínua e independente para atualizar o monitor. Isso resultava em momentos onde a tela era atualizada para o monitor enquanto estava parcialmente desenhada, exibindo uma imagem incompleta. Essa rápida alternância entre quadros completos e em processo de renderização era percebida pelo olho humano como a forte cintilação que precisou ser solucionada.

Para resolver este problema, foi implementada a técnica de Double Buffering. A essência desta abordagem é nunca desenhar diretamente na memória de vídeo que está sendo exibida. Em vez disso, alocamos um segundo buffer, uma "tela de rascunho" ou "back buffer", em uma área separada da memória RAM do sistema. Todo o processo de renderização de um novo quadro acontece exclusivamente neste buffer escondido. O programa executa a mesma sequência de antes (limpar, desenhar o fundo, os canos, os pássaros e o placar) mas faz isso nesta área invisível ao usuário.

A etapa final do double buffering é a atualização da tela visível, que deve ocorrer de forma a parecer instantânea. Uma vez que a cena no back buffer está completa, o programa executa uma única e rápida operação, que é a cópia de todo o conteúdo do back buffer para o framebuffer principal, a memória de vídeo real. Para isso, utilizamos a função memcpy, que é otimizada para transferir grandes blocos de memória com máxima eficiência. Essa transferência é tão veloz que ocorre em um intervalo de tempo menor que um ciclo de atualização do monitor. Do ponto de vista do jogador, a tela não é redesenhada aos poucos; ela simplesmente se transforma de um quadro completo para o próximo. Isso acaba com o efeito da cintilação.