Arcade Machine com gestor de janelas

Álvaro Silva, Bruno Maurício

15 de Janeiro de 2020

Resumo

Neste trabalho simulamos uma Arcade Machine usando a placa UNO32. Para o fazer, conectamos a placa um ecrã LCD (GMG12864-06D) de 64 *128 bits, bem como um rato (M-SBF96), além de um teclado. Os jogos que desenvolvemos foram o PONG (controlado por dois potenciómetros) e o SNAKE (controlado por teclado). Grande parte da complexidade do trabalho encontra-se no sistema do gestor de janelas que permite minimizar e fechar os jogos/programas em execução, assim como gerir as janelas e suas interações.

Capítulo 1

Interface com periféricos:

O código apresentado encontra-se comentado com informação mais diretamente ligada ao funcionamento exato do projeto.

1.1 Teclado:

1.1.1 Protoco do teclado :

O protoco de comunicação entre o teclado e o arduino é o PS2, que é um protoco de comunicação série com um dataline e um clockline. Quando o teclado quer enviar informação verifica se a clockline está ativa. Esta terá que estar ativa durante pelo menos 50 microssegundos para que o dispositivo comece a transmitir a informação. A comunicação é feita com 11 bits , sendo o primeiro o start bit (sempre colocado a zero), 8 bits de informação , 1 bit de paridade ,e um stop bit que é sempre colocado a 1. [Chapweske, 2003]

1.1.2 Implementação :

Recorremos à implementação feita por PaulStoffregen em https://github.com/PaulStoffregen/PS2Keyboard , fazendo ligeiras alterações (retirar funções que não usamos , retirar defines que não usamos referentes a outras placas e retirar dicionários de conversão entre teclados de vários paises) e, para assegurar o bom funcionamento do programa seguimos o que aprendemos e alteramos a linha presente no final do método PS2keyboard.begin (attachInterrupt(irq_num, ps2interrupt, FALLING); por attachInterrupt(digitalPinToInterrupt(irq_num), ps2interrupt, FALLING);.

1.1.3 Funcionamento da implementação:

A parte pricipal da implementação é o interrupt (ps2interrupt) que trata de atualizar a variável incoming (com cada bit lido ,se estes forem os bits de informação, a variável incoming fica com o valor desse bit guardada nos seus bits através de um or lógico na posição correta) .Quando o contador de bit chega a 11 , final da transmissão ele guarda a informação util variável incoming num buffer de chars (que obrigatoriamente se apaga se o tempo entre um bit recebido e outro for superior a 250 ms. (assim pelo tamanho ocupado do buffer é possivel

distinguir as teclas que quando sao primidas enviam varios bytes do que carregar numa tecla normal várias vezes) (de referir que o buffer e o incoming e os respetivos tamanhos são variáveis globais).

Quando se quer ler chama-se o método PS2keyboard.avaiable que retorna 1 basicamente se tiver caracters no buffer , seguido de um PSkeyboard:read que pega nesse buffer de bytes e converte no código que o utilizador espera char de 8 bits (mediante o país do teclado e o tipo de moficador do caracter ,usando uma série de funções para esse efeito) e o retorna.

1.2 Rato:

1.2.1 Protoco do rato :

O protocolo do rato também é o PS2, pelo que a comunicação com este funciona da mesma forma. No entanto, o rato devolve quatro respostas em "sequência" diferentes, uma com informações gerais (de estado de comunicação, botões atualmente premidos, e quadrante geral de direção do movimento), uma com a velocidade (cuja precisão é definida na inicialização) segundo X, outra segundo Y, e a última com a velocidade da roda de scroll (caso esta exista, como é o caso).

1.2.2 Implementação :

Recorremos à implementação apresentada no seguinte link, nos arquivos da arduino.cc https://playground.arduin
Tal como para o teclado, removemos as secções desnecessárias à nossa utilização.

1.2.3 Funcionamento da implementação:

Ao contrário do teclado, o rato etá continuamente a enviar informações, mesmo quando quieto. Como tal, é feito um poll periódico ao rato, quando este é necessário. A utilização de interrupts neste caso, prejudicaria seriamente o desempenho geral.

1.3 Monitor:

1.3.1 Protoco do monitor :

Foi implementado o protocolo SPI manualmente, com base em diversos recursos online, bem como datasheets de LCDs semelhantes (pois não conseguimos encontrar muita informação sobre o que nós possuímos, GMG12864). Claramente não a melhor solução, mas de início tentámos evitar bibliotecas externas ao máximo, acabando inclusive por perder muito tempo a ler datasheets e demonstrações online até termos um protótipo funcional, mas tendo percebido melhor o funcionamento do protocolo.

1.3.2 Implementação :

Na tentativa de uma maior abstração, foram criadas múltiplas funções e estruturas para controlar o LCD. Em geral, só deverá de ser necessário utilizar as funções ST7565_Print, ST7565_Print_Int e ST7565_write_pixel na escrita geral para um ecrâ.

1.3.3 Funcionamento da implementação:

Para a inicialização das funções, RequestScreen para reservar um ecrâ.

A nível de definições, MoveUp para pôr o ecrâ como o mais "acima".

ST7565_RefreshEntireScreen faz refresh de todo o ecrâ.

Capítulo 2

Jogos e programas:

2.1 Considerações gerais:

Ambos os jogos são encapsulados em funções de nome SnakeApp() ,PongApp() que permitem com que estas sejam executadas apartir do main apenas chamando essas funções (que por sua vez chamam a classe respetiva e fazem a gestão dos ecrãs), a lógica delas está descrita em baixo.

2.2 PONG:

2.2.1 Estrutura de código:

O PONG está estruturado como uma classe (pongGAME), com os métodos newBallspeed (que altera a velocidade da bola, se necessário, quando bate nos pads dos jogadores ou nas margens), newRound (que inicia uma nova ronda caso tenha havido uma marcação de ponto), newResult (que executa o update dos resultados dos jogadores) e o atualizeGame (que controla a lógica do jogo).

As variáveis guardadas na classe para a execução do jogo são: o número de pontos necessários para vitória, a posição das barras (é guardada a coordenada vertical da posição do canto superior esquerdo delas), os pontos de cada jogador) a posição da bola (que necessita de 2 inteiros para guardar a posição vertical e horizontal do canto superior esquerdo da bola), e a velocidade da bola (também guardada com dois inteiros).

2.2.2 Funcionamento do ciclo de jogo:

O mapa é iniciado com a criação de um objeto pongGame, que coloca os pontos dos jogadores a zero, centra a bola no centro do ecrã, inicia a posição dos pads e gera um vetor de velocidade pseudoaleatório para iniciar o movimento da bola. Após este procedimento, o jogo está a executar o método atualizeGame em loop. Este método atualiza a posição das barras dos jogadores mediante os valores lidos nos potenciómetros. De seguida, atualiza a posição da bola mediante a velocidade atual e chama o método newBallspeed, que altera o vetor velocidade

da bola, se for caso disso, e retorna a 1 ou 2 se a bola se encontrar numa posição de golo para o jogador 1 ou 2, respetivamente, e a 0 se não houver golos. Por fim, se for golo, atualiza o resultado usando o método newResult e executa o método newRound. O método atualize game retorna a 1 ou 2 se o jogo tiver acabado com vitória do jogador 1 ou 2, e a 0 caso ainda continue (este valor vem do retorno do método newResult).

De referir ainda que o método newBallspeed altera o vetor velocidade da bola $[V_x, V_y]$quando bate em cima ou em baixo da borda do jogo para $[V_x, -V_y]$ e, quando bate num pad de um jogador, para$[-V_x, (|V_y| + 1) \times rand(-1, 1)]$. Quando bate num pad de um jogador a velocidade x inverte, a velocidade em y aumenta 1 em módulo e vai para cima ou para baixo consoante o número pseudoaleatório (com média testada 0) for maior do que 0 ou não.

2.2.3 Interface com impressão no ecrã :

A classe também guarda variáveis e tem métodos próprios para executar a sua impressão no LCD. Ela guarda um unsigned char screen, que representa um screen dos 7 disponíveis onde será atualizado o jogo, e um inteiro o ControlledDelay que representa o delay entre jogadas sucessivas.

São usadas as funções criadas para o LCD (ST7565_Print e ST7565_Print_Int) para imprimir os pontos de cada jogador e a dificuldade do jogo. Para imprimir os pads e a bola foi criada a função (Pong_paint), que imprime um retângulo ou quadrado e recebe a linha e a coluna onde quer escrever, a altura e largura do retângulo, além de escrever um 0 ou um 1 (esta função não faz mais do que chamar a função ST7565_write_pixel num duplo ciclo for). Foi também criada outra função (Pong_write) que é apenas um chamamento duplo da função Pong_paint para imprimir os pads dos jogadores.

É de sublinhar que, em cada interação, o jogo apaga e reescreve as bolas e os pads, sendo apenas essas secções que recebem o refresh no screen.

2.2.4 Controladores :

A posição dos pads é controlada pelo nível de tensão dos potenciómetros através da expressão seguinte, que atualiza a altura do pad mediante a fração do valor máximo recebido pelo potenciómetro (1023) e a altura em que ele se pode mover (this->ScreenHeight-CursorHeight):

1. NewPosition1 = (movplayer1*(this->LCDHeight-CursorHeight))/1023; (no ficheiro Pong.cpp)

O movplayer , valor trazido para dentro da função que representa o valor lido pelo potenciómetro vem da chamada a analogread através da função Listen da struct Controller C (defininda em ControlsMain.cpp) .

2.2.5 Observações:

A conciliação do tempo do programa com o tempo do refresh do LCD foi problemático, não sendo possível fazer um mecanismo de velocidade que ativasse todos os bits pelo caminho (à

medida que a bola ia passando), porque tornaria o jogo muito lento para os jogadores, assim a velocidade no momento k faz atualizar a posição k+1 pela equação $p_{k+1} = p_k + v_k$. .

Sempre que há um toque num pad de um jogador a nova velocidade gerada da bola tem componente vertical sempre crescente, mantendo-se a horizontal, o que faz com que o jogo se torne cada vez mais difícil.

2.3 SNAKE:

2.3.1 Estrutura de código:

O snake está estruturado numa classe snake_Game ,tendo os métodos new_food_point (que cria um novo ponto de comida no mapa) ,move_to_local (que atualiza as posicões x,y mediante o movimento escolhido pelo jogador) ,check_if_can_move (que verifica se a posição para onde se vai mover é válida),hit_food_point (que retorna um se a snake atingiu o ponto de comida e 0 caso contrário) e o método principal game_mov que executa a lógica do jogo.

As variáveis guardadas na classe são : 3 pontos ,cada um que guarda 2 int (coordenadas em x e y) ,referentes à posição da frente ,trás da cobra e do ponto de comida. Um vetor de unigned char que guarda a posição relativa dos pontos que ocupa a snake , e o tamanho atual da snake.

2.3.2 Funcionamento do ciclo de jogo:

É inicializada a classe que coloca a snake no centro do ecrã e gera um ponto de comida aleatório. É chamado em loop o método game_mov que verifica se o movimento que o jogador quer fazer é válido após isso se o tamanho for 0 (so tiver um ponto), coloca a tail da snake na mesma posição que a frente , move a frente da cobra ,se acertar num ponto de comida acrescenta um elememto ao vetor snake e atualiza a posição anterior com o movimento que executou e gera um novo ponto de comida ,se não acertou em nenhum ponto , apaga a tail e faz shift ao vetor de posições relativas da snake.

2.3.3 Interface com impressão no ecrã :

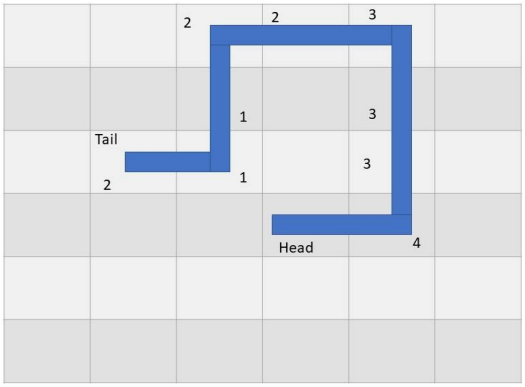
Para imprimir o jogo recorre às funções Write_Square (que apenas chama a funcao em duplo ciclo ST7565_write_pixel) de forma a desenhar um quadrado numa poisição especifica do lcd , esta função imprime tanto o foodpoint como imprime a head da SNAKE e apaga a TAIL. É de sublinhar que em cada interação o jogo apenas apaga a TAIL (se for caso disso ,não ter comido um ponto) , escreve a nova posição de HEAD (parte da frente) e se necessário desenha outro foodpoint .

2.3.4 Controladores:

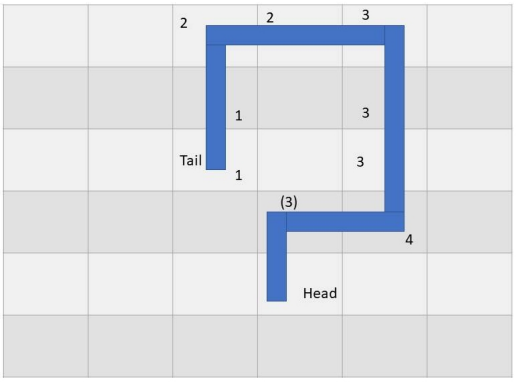
A snake é controlada pelo teclado pelas teclas das arrows , como se usou a biblioteca arduino PS2Keyboard com poucas alterações para ler do teclado usa-se a função LISTEN (que é um alias do método da biblioteca keyboard.avaiable() seguido de um keyboard.read() para

devolver o caracter correspondente) . Após ter o char converte-se o código do char para o mov que a classe snake_game espera receber .

2.3.5 Observações:



Snake antes



Snake depois

Para poupar memória o vetor de posições relativas guarda em cada posição um unsigned char de valor 1 se o proximo elemento da cobra estiver em cima , 2 se estiver á direita ,3 se estiver em baixo e 4 se estiver à esquerda (com esta forma de guarda os dados poupamos de 2 a 8 vezes mais memória , em vez de 2 unsigned chars ou 2 int que guardariam a posição absoluta da cobra ,mas perde-se a possibilidade de verificar se ela se comeu a si própria ,algo facilmente fazivel no lcd onde ela está impressa). A atualização deste vetor fica clara no esquema seguinte :

Considerando x crescente para a direita ,e y crescente para baixo , canto superior esquerdo (0,0). O vetor antes que guarda a posição relativa dos próximos elementos é [2,1,1,2,2,3,3,3] ,a head está em (3,3) e a tail em(1,2). Quando recebe a ordem para ir para baixo a nova head é (3,3+1)(head atualizada com ordem de movimento) ,a nova tail é (1+1,2).(tail atualizada com último elemento do vetor de próximos elementos), e o novo vetor passa a ser [1,1,2,2,3,3,3,(3)] (vetor antigo shiftado de um elemento com um elemento acrescentado correspondente ao movimento realizado. De referir também que o jogo tem um parâmetro ratio que permite escolher a largura da snake e do ponto no mapa desde que ele seja uma potência de base dois até 8.

Capítulo 3

Gestor de janelas:

3.0.1 Estrutura do código:

O código do gestor de janelas está maioritariamente presente em 2 bibliotecas. ScreenMainFuncts e ScreenBaseFuncts (header e cpp). A ScreenBaseFuncts para as funções que interagem diretamente com as estruturas de controlo criadas (mais à frente explicitadas) ou com o LCD em si, enquanto que a ScreenMainFuncts engloba as funções de uso mais "abstrato".

3.0.2 Funcionamento geral:

Os ecrãs são guardados num vetor de 64*128 bytes (ScreenScreen), sendo que cada um destes corresponde a um pixel do LCD em uso.

Inicialmente utilizávamos o tipo de variável "Bool", mas depressa nos apercebemos que este era na realidade um byte, tal como o tipo char, o que nos permitiu na mesma quantidade de memória ter 8 ecrãs em vez de 1 (cada um destes ecrâs correspondendo a um dos bits).

Como o ChipKit é mais rápido do que o LCD (80MHz para os estimados 30KHz do controlador do LCD), um dos ecrãs (correspondente ao MSB) é utilizado como buffer, no qual são acumulados e devidamente sobrepostos os outros ecrãs, e a partir do qual, o LCD é atualizado.

De modo a podermos guardar informação extra sobre os ecrãs (para além do que neles está guardado), um vetor auxiliar denominado ScreenControl, é utilizado (ver documentação para detalhes mais específicos) e ordenado consoante a prioridade dos ecrãs. É através deste vetor auxiliar (que também guarda tamanho absoluto), que é possível executar as ações atualmente presentes (como minimização), e eventualmente implementadas, bem como realizar um refrescamento de ecrâ mais eficaz, só prestando atenção ao que foi alterado, previamente intersetado com a janela em epecífico.

3.0.3 Controller:

Foi também criado um Controller que mantém informação sobre os diversos periféricos, de modo a facilitar a sua utilização global.

3.0.4 Observações:

O código escrito tem diversas funções e parâmetros que, à primeira vista parecem inúteis, mas que ainda serão desenvolvidos em features.

Uma das razões pelas quais achámos este projeto tão interessante, é pela possibilidade de debug direto de variáveis num microcontrolador, externo ao computador (através das funções ST7565_Print_Byte e ST7565_Print_Int).

Bibliografia

Adam Chapweske. The ps/2 mouse/keyboard protocol. *electronic file available: http://www. computer-engineering. org/ps2protocol*, 2003.