UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Documentação de Projeto – Parte 2
Design, Estudo da Plataforma

Projeto: Embedded Snake

Autores: Gustavo Henrique Zeni
Lucas Perin Silva Leyser

Versão: 20-Nov-2022

Parte 2a – Design

1 Introdução

O objetivo deste documento é detalhar, formalmente, todas as etapas, estudos e habilidades empregadas para implementar as especificações do projeto ***Embedded Snake***, determinadas no primeiro documento, o CONOPS do projeto.

2 Arquitetura Funcional

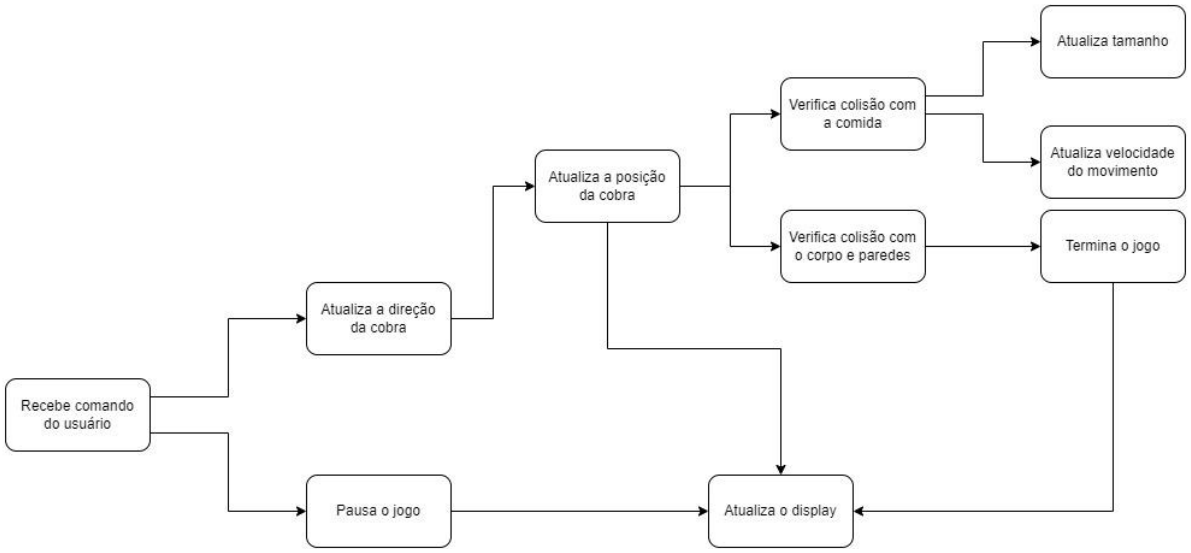
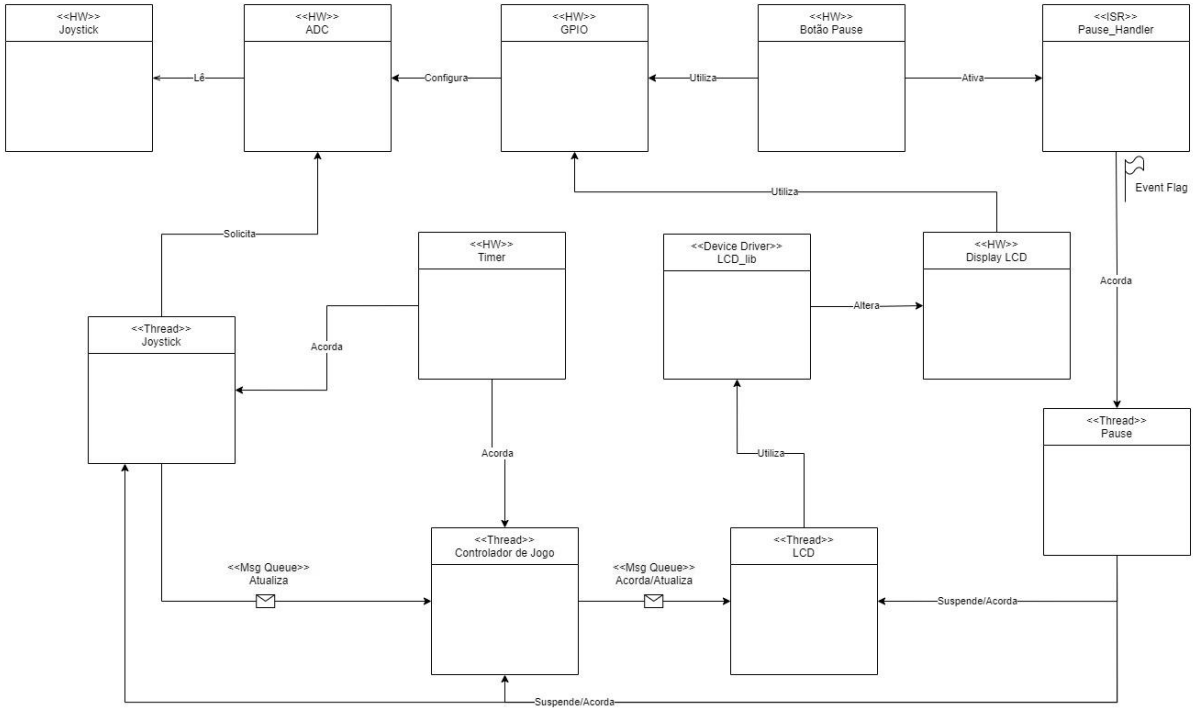


Diagrama da Arquitetura Funcional.

O usuário pode interagir com o sistema de duas maneiras, através do botão de *pause* e do *joystick*. O pressionar do botão pause, ocasiona no pausamento do jogo e da exibição de uma mensagem na tela, que indica que o jogo foi pausado.

Quando o usuário interage com o joystick, ele modifica a direção que a cobra está se movimentando. Após isso, a cobra anda e é necessário atualizar o display e conferir se houve alguma colisão, seja com o próprio corpo da cobra, com uma das paredes do cenário ou com a comida, caso o último caso tenha acontecido, a velocidade e tamanho da cobra são atualizados, se um dos dois primeiros casos ocorrer, o jogo termina e uma mensagem é exibida na tela indicando o fim da partida.

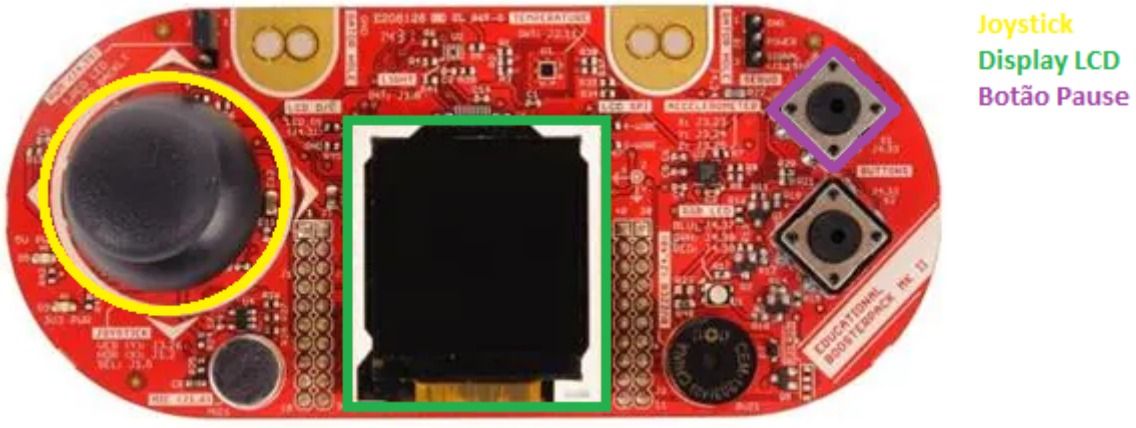
3 Arquitetura Física



Arquitetura do Sistema

4 Interface com o Usuário

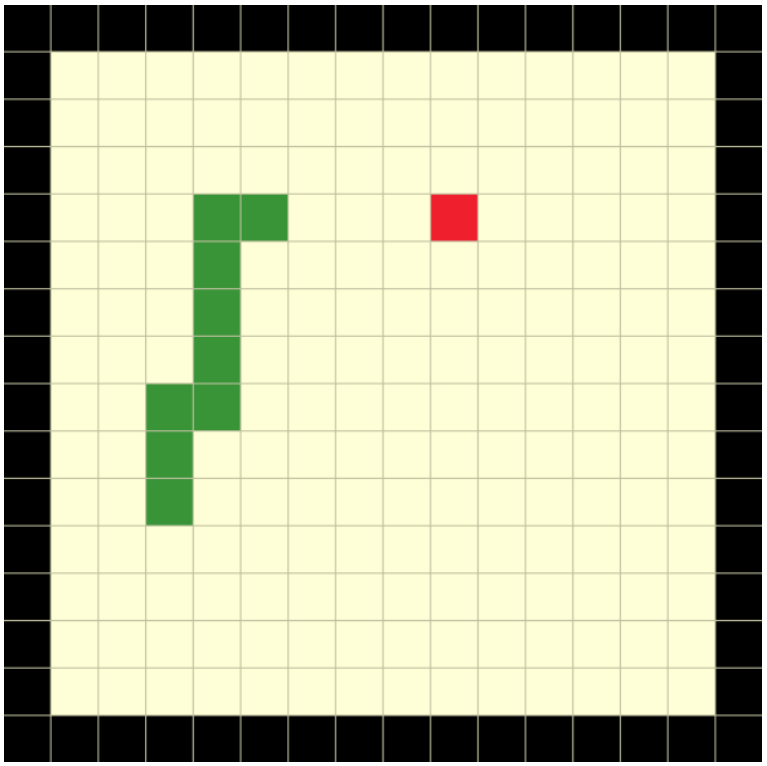
Como exemplificado no documento 1, todo o interfaceamento com o usuário será dado através do kit educacional. O *joystick* receberá a direção que o usuário deseja que a cobra siga, o botão S1 será utilizado para pausar e despausar o jogo e o display LCD exibirá o cenário e todas as informações pertinentes ao usuário.



Interface com o kit educacional.

O cenário do jogo será exibido no Display LCD do kit educacional, ele consiste em um tabuleiro de 16x16 posições, com as primeiras e últimas linhas e colunas demarcando as paredes, restando uma área de 14x14 para a movimentação da cobra. Como o display

possui resolução de 128x128 pixels, cada posição do cenário será representada por um conjunto 8x8 de pixels que serão tratados como uma mesma posição.



Prévia da Interface do Jogo.

5 Mapeamento da Arquitetura Funcional à Arquitetura Física

FUNCIONALIDADES	ARQUITETURA FÍSICA													
	Joystick (HW)	ADC	Thread Jostick	Botão	Pause Handler	Event Flag	Thread Pause	LCD (HW)	LCD Driver	Thread LCD	Thread Controlador de jogo	GPIO	Timer	Message Queue
Input do usuário	X	X	X									X	X	X
Atualiza direção			X										X	X
Atualiza posição											X		X	X
Colisão comida											X			
Colisão corpo/paredes											X			
Atualiza tamanho											X			
Atualiza velocidade											X			
Termina o jogo								X	X	X	X	X		X
Pausa o jogo				X	X	X	X	X	X	X		X		
Atualiza Display								X	X	X	X	X		X

6 Arquitetura do Hardware

Para a correta implementação do sistema aqui descrito, vários recursos de hardware já presentes na placa Tiva e no booster pack são utilizados, nessa seção iremos discorrer sobre cada um deles assim como a sua função no sistema.

6.1 Joystick.

O joystick é o responsável por receber o comando do usuário para alterar a direção da movimentação da cobra de acordo com a decisão do mesmo. Funciona basicamente

como dois potenciômetros cuja leitura informa a variação, em relação ao centro, de ambos os eixos de movimento, horizontal e vertical.

6.2 Botão Pause.

Um botão de uso geral, com apenas dois estados: “HIGH” ou ”LOW”. Como o nome sugere, é o botão que, quando acionado, pausa totalmente a execução do jogo.

6.3 Display LCD.

Uma tela LCD de 128x128 pixels, nela é feita toda a representação visual do atual estado do jogo para o usuário.

6.4 ADC.

Um conversor analógico-digital, utilizado para “traduzir” a leitura dos potenciômetros do joystick em valores digitais que serão utilizados pelo controlador para determinar qual a nova direção que o usuário deseja que a cobra se movimente.

6.5 Timer.

Um dispositivo que conta quantos ciclos de clock ocorreram desde sua ativação, como o clock da placa é, praticamente, constante, essa contagem pode ser utilizada para determinar espaços de tempo. No sistema, os timers são utilizados para disparar as rotinas de leitura da posição do joystick e da atualização do display LCD.

6.6 GPIO.

Pinos de entrada e saída gerais, fazem a maior parte da comunicação entre os periféricos (joystick, botão, display) com o microprocessador.

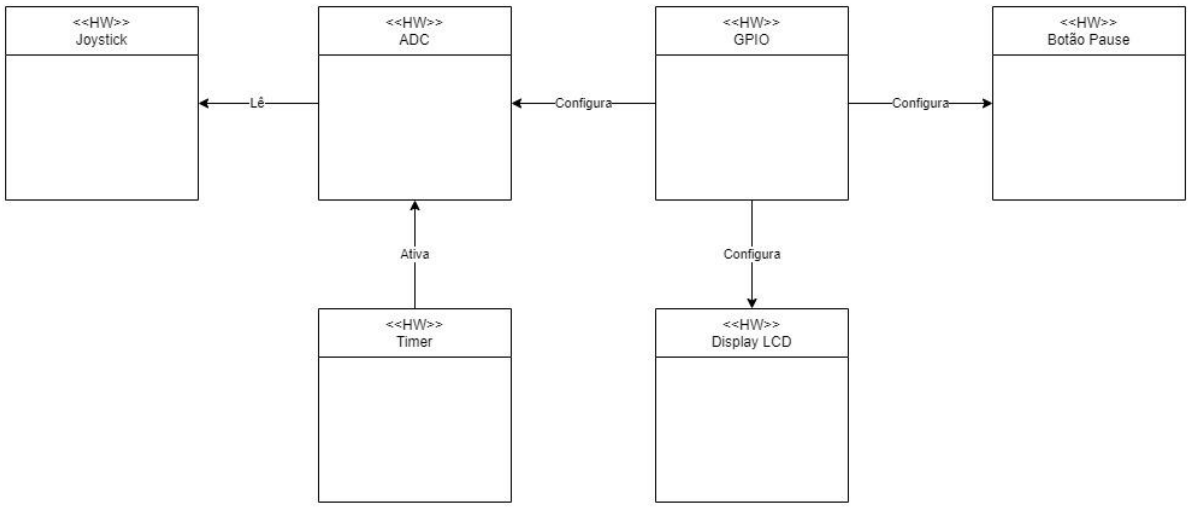


Diagrama da Arquitetura de Hardware.

7 Design Detalhado

7.1 Abordagem geral.

Por ser um projeto complexo, com muitos estados possíveis, um diagrama mais simples e abrangente pode ser observado na figura abaixo:

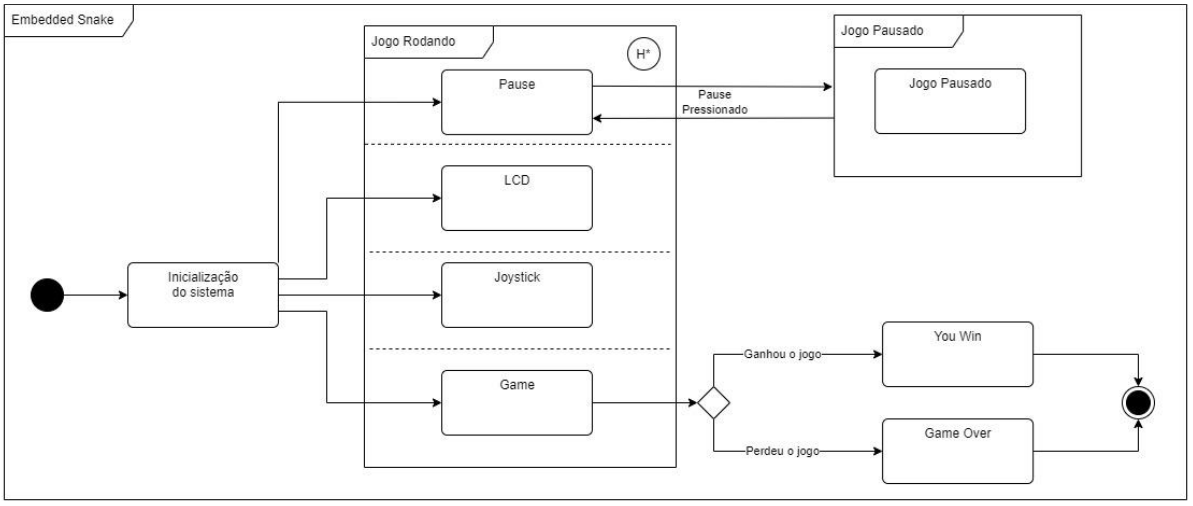


Diagrama de estados do sistema.

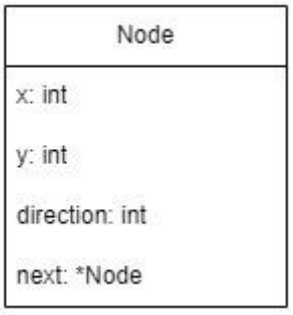
Após a inicialização do sistema, entra em vigor o estado "Jogo Rodando", em que quatro *threads* são executadas paralelamente, uma responsável pela atualização do display LCD, outra pela leitura do joystick, a terceira controla todo o estado do jogo e a última faz as mudanças relativas à pausa do sistema.

Só é possível sair desse estado ao pausar o jogo, levando a um estado "vazio" que apenas espera um novo pressionar do botão pause, ou pelo término da partida, sendo por "Game Over", colisão com o corpo da cobra ou com as paredes, ou por "You Win", o tamanho da cobra cobre totalmente a área jogável do cenário. Em todos os casos, o display exibe a respectiva mensagem, informando o usuário do estado em que a sua partida se encontra.

7.2 Estrutura da cobra.

Para a representação interna da cobra, o sistema utiliza uma lista encadeada do tipo "Node" criado pelos desenvolvedores, esse tipo conta com quatro parâmetros: "x", "y" "direction" e "next", esses parâmetros representam, respectivamente, coordenada horizontal dessa parte do corpo, coordena vertical da mesma parte, direção de movimento e próxima parte do corpo. A variável "direction" assume os seguintes valores:

- 0 - Centro
- 1 - Direita
- 2 - Esquerda
- 3 - Cima
- 4 - Baixo



Estrutura Node (Cobra).

A comida também é representada pela estrutura "Node", com as coordenadas horizontal e vertical correspondentes à sua posição no cenário, "direction" igual à zero e o ponteiro "next" nulo.

7.3 Atualização do display LCD.

A fim de exibir, com fidelidade, o estado do jogo no display, é necessário atualizá-lo sempre que houver qualquer alteração nas posições dos elementos do jogo, considerando que as paredes são fixas e a comida só muda de posição quando colide com a cobra, o único momento em que existem mudanças nas posições do jogo é quando a cobra se movimenta. A velocidade da cobra é igual a quantas posições do cenário ela andará em um segundo, ou seja, a taxa de atualização do display. Por isso, o display é atualizado sempre que a thread Game envia uma mensagem para a thread LCD, essa mensagem diz se o display deve desenhar a cobra inicial do jogo assim como a primeira comida, se deve apenas fazer a cobra se mover ou se a cobra deve aumentar de tamanho e uma nova comida deve ser mostrada. Após a atualização, a thread fica esperando até que uma nova mensagem seja recebida.

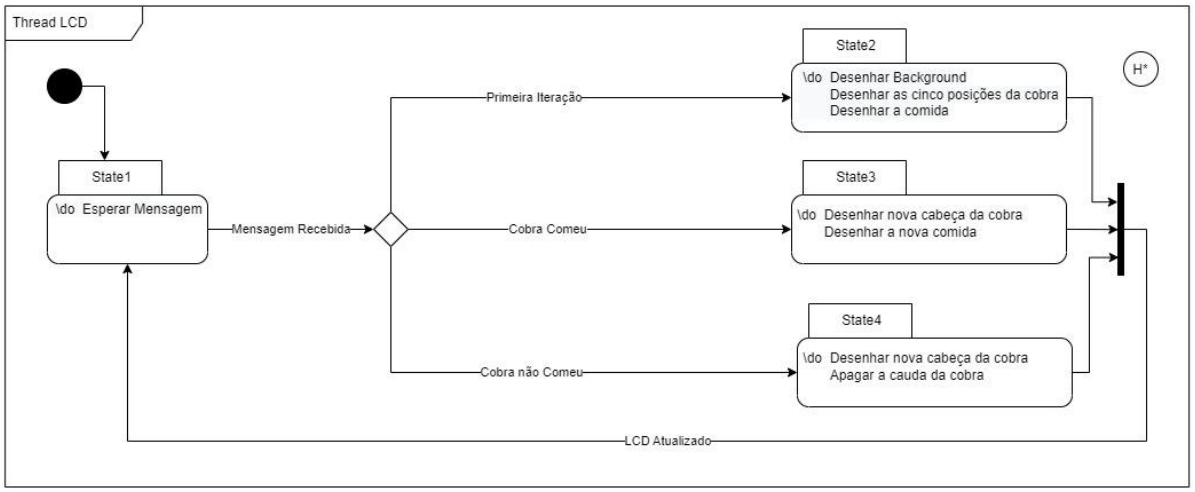


Diagrama de estados da Thread LCD.

7.4 Leitura do joystick.

Por se tratar de um sinal analógico, o uso de interrupções para a leitura dos potenciômetros do joystick não é uma opção viável, por isso, uma thread é responsável por acionar os respectivos ADC's e solicitar a conversão dos valores horizontal e

vertical do joystick, atualizando a direção da cobra. Como, naturalmente, as conversões têm variações mesmo que o joystick não tenha sido movimentado, não é necessário recalcular a direção do movimento sempre que os valores lidos divergirem do centro do joystick, isso é feito apenas quando a variação excede os 25%. Além disso, é preciso se atentar aos movimentos inválidos, onde a cobra faria uma mudança de 180º em relação à direção de movimento atual, esses comandos são ignorados pelo sistema. Caso ambas as condições de guarda sejam satisfeitas, a direção do movimento é calculada e, caso seja diferente da atual, uma mensagem é enviada a thread Game contendo a direção lida, dessa maneira é possível fazer uma bufferização dos comandos do usuário, que serão representados pela cobra na ordem em que foram descritos pelo joystick. É importante ressaltar que a cobra não se movimenta na diagonal, então, a direção resultante calculada pelo sistema é puramente horizontal ou vertical (aquela que possuir a maior variação com relação ao centro do joystick).

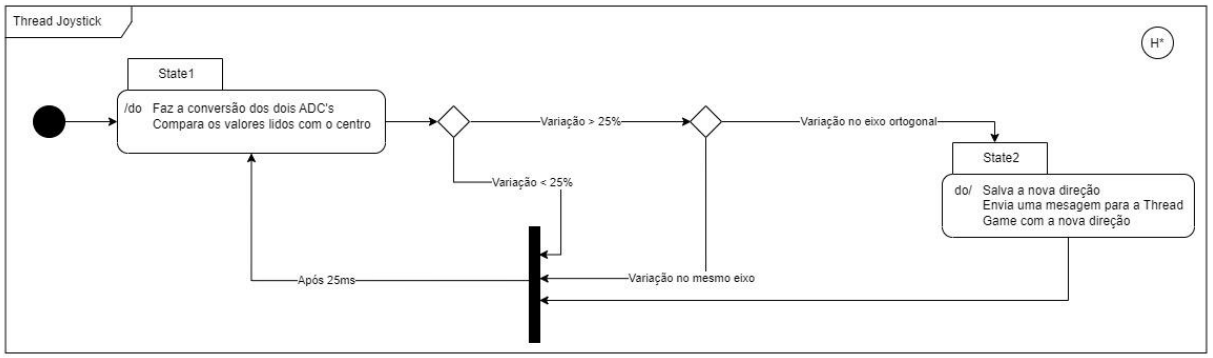


Diagrama de estados da Thread Joystick.

7.5 Controle do jogo.

A cada loop executado do jogo, é verificado se o usuário inseriu um comando para alteração da direção da cobra ou se a direção se mantém a mesma da iteração anterior.

Então, o movimento da cobra é simulado para verificar se a cabeça colidirá com a comida no próximo movimento, se esse for o caso, a cobra não altera a sua posição, um nó com as coordenadas da comida atual é adicionado, aumentando o tamanho da cobra em uma posição, se esse incremento tornar a cobra com o tamanho máximo possível (14x14 = 196 posições), o jogador venceu o jogo e o display mostra uma mensagem "You Win" seguida do tamanho final da cobra, se a cobra não chegou ao tamanho máximo, uma nova comida é gerada em um local aleatório do cenário que não seja uma posição da cobra ou das paredes. Além disso, a variável que controla a velocidade de movimentação da cobra é aumentada em 5% do seu valor atual, respeitando o limite superior de 20 posições por segundo, devido ao limite da taxa de atualização do display. Quando essa simulação falha, todos os nós do corpo da cobra têm suas coordenadas e direções atualizadas. Para a atualização das coordenadas dos nós da cobra, a cabeça recebe a atual direção que o usuário comandou através do joystick e cada um dos outros nós da lista recebe a direção do nó anterior à ele, após essa atualização da direção, a posição final dos nós é calculada.

Além da atualização do corpo da cobra, é preciso verificar se houve alguma colisão após esse movimento. Como apenas a cabeça pode colidir com alguma coisa, primeiramente, as coordenadas da cabeça são comparadas com ambos os valores

mínimo (0) e máximo (15) das coordenadas do cenário, caso uma das coordenadas da cabeça esteja nessa posição, ela colidiu com uma das paredes e o jogo termina. Se não houve colisão com as paredes, ainda é preciso verificar se a cabeça da cobra não colidiu com o restante do corpo, para isso, as coordenadas da cabeça são comparadas com as coordenadas de todos os outros nós da cobra, caso a cabeça possua as mesmas coordenadas que algum dos nós, a cobra colidiu com si mesma e o jogo acaba. Em ambos os cenários, uma mensagem "Game Over" seguida do tamanho final da cobra é exibida no display LCD.

Após esses passos, a atualização do estado interno da cobra foi concluída e uma mensagem é enviada, acordando a thread LCD que representa esse estado no display. É necessário inserir um delay entre as chamadas de atualização do display, além de respeitar os limites do componente, ele também é preciso para que o usuário consiga acompanhar o jogo. Esse delay é calculado com base em dois valores, a velocidade de movimento (guardada em posições por segundo) e o tamanho da cobra. Seria possível fazer esse cálculo apenas com a velocidade de movimento, assumindo que o tempo para o cálculo do próximo estado da cobra é rápido o suficiente para ser desprezado na temporização, porém, conforme a cobra aumenta de tamanho, o loop que verifica se a cabeça colidiu com alguma outra parte do corpo, passa ter uma execução cada vez mais impactante no tempo total da iteração geral, por isso, delay esperado é dividido entre o tempo de delay real, onde a thread fica suspensa, e o tempo de execução do loop de verificação de colisão com o corpo. Analisando o código em Assembly gerado após a compilação do sistema e de posse do clock da placa, foi aproximado um tempo de 0.686ms para conferir a colisão da cabeça com um dos nós do corpo da cobra. O tempo de suspensão da thread é dado pela seguinte equação:

$$Delay(ms) = 1000/Snake\_Speed - 0.686 * Snake\_Size$$

Como a thread só pode ser suspensa por um intervalo inteiro de ms, o valor calculado é truncado no final.

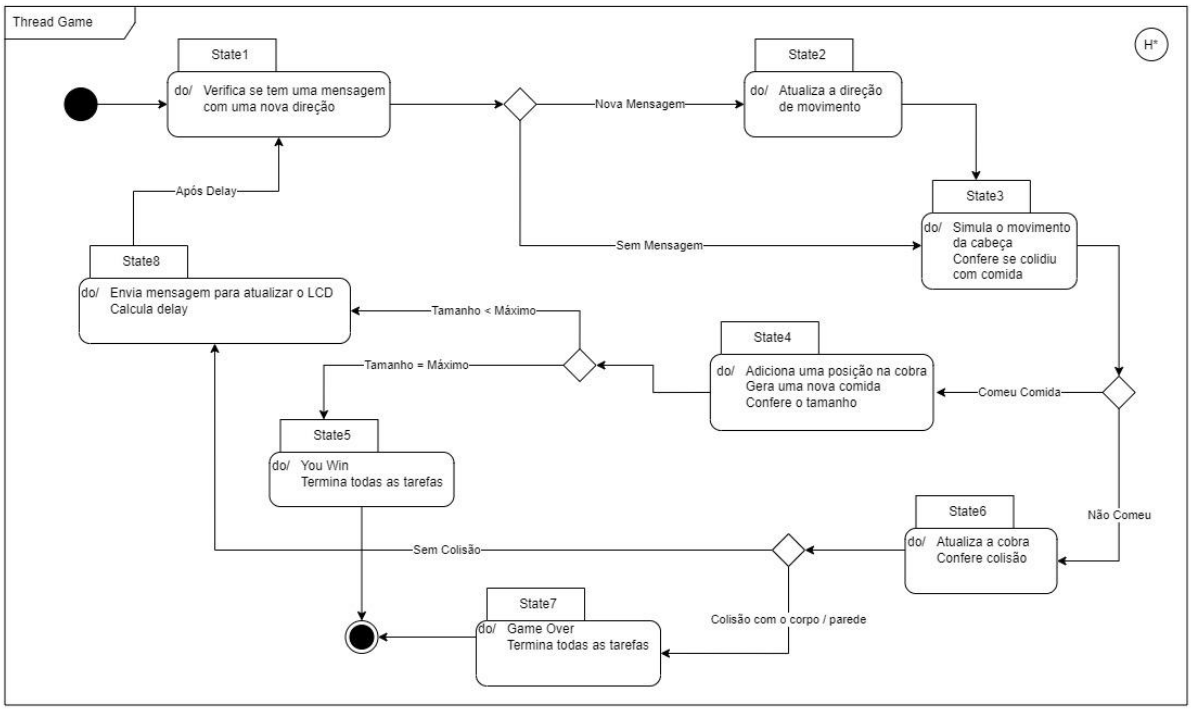
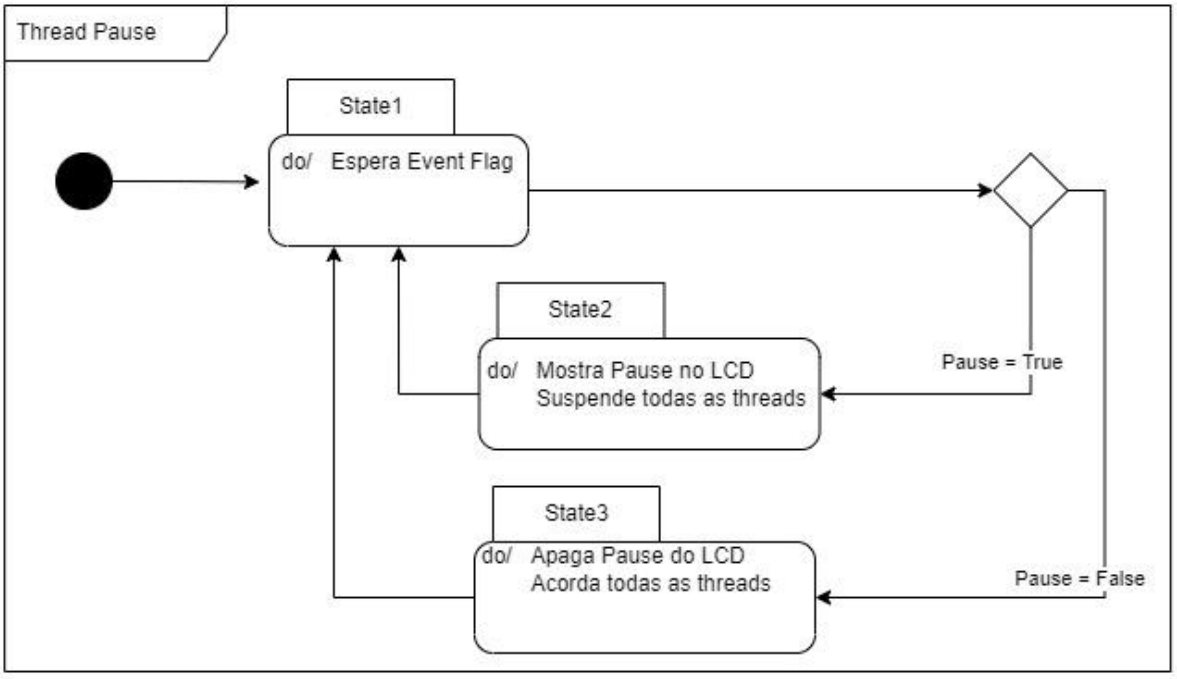


Diagrama de estados Thread Game.

7.6 Pause do jogo.

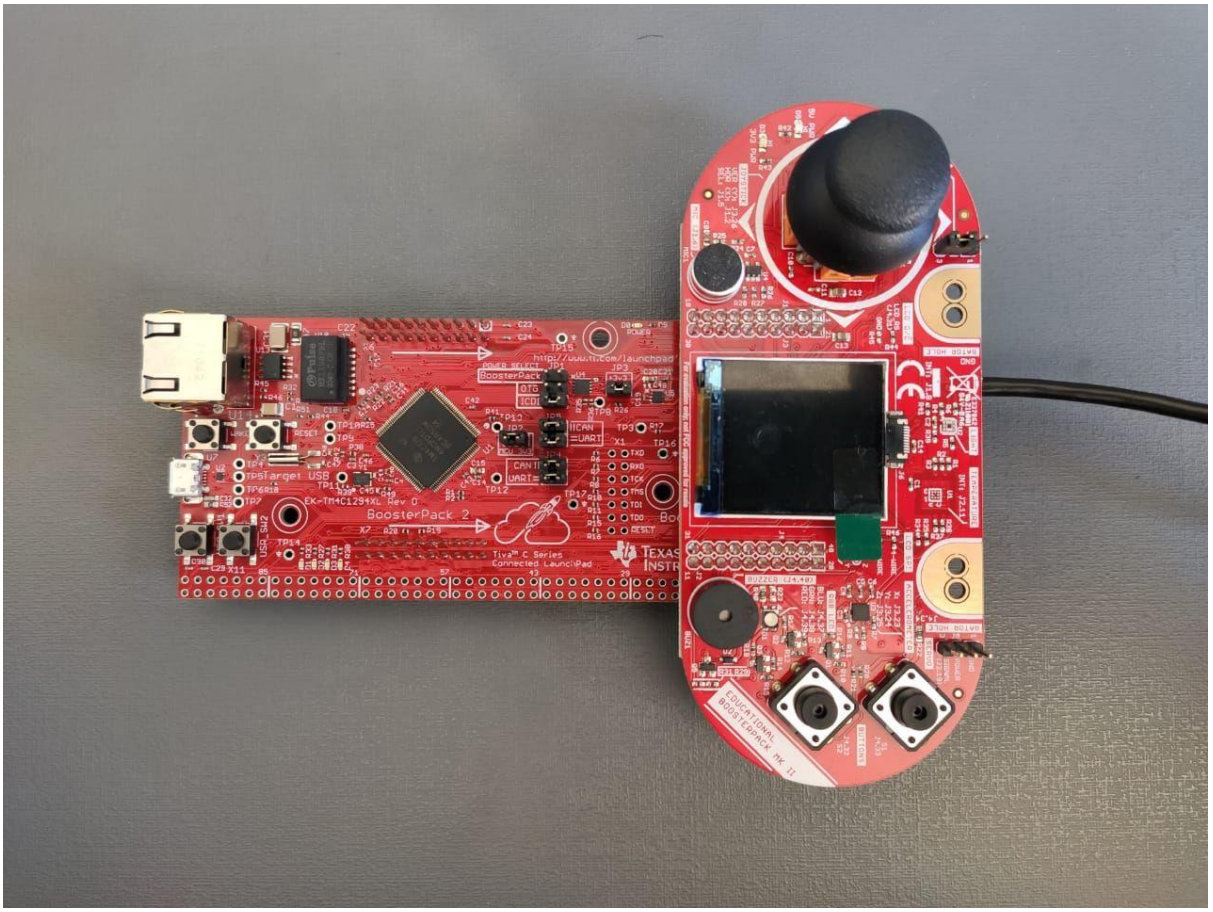
A pausa do sistema é feita através de uma interrupção atrelada ao botão S1 do kit educacional, assim que o usuário pressiona o botão a ISR associada é executada. A ISR apenas inverte o valor de uma variável global "Pause" e ativa uma *Event Flag* que acorda a thread "Pause".

Após ser acordada pela *Event Flag*, a thread confere o valor da variável "Pause". Caso a variável seja verdadeira, significa que o usuário deseja pausar o jogo, então as outras três threads são suspensas e o símbolo de pausa é exibido no canto superior direito do LCD, se for falsa, o usuário deseja retomar o jogo, portanto, as outras três threads são acordadas e o símbolo de pausa é apagado do LCD.



Parte 2b – Estudo da Plataforma

Para o devido funcionamento do sistema é necessário que a conexão entre a placa Tiva e o booster pack seja feita conforme a foto abaixo:



1 Leitura do Joystick

O joystick presente no booster pack se resume em um botão, que não será utilizado no sistema, e dois potenciômetros, cuja leitura marca a variação no eixo horizontal/vertical, tratando-se de variações analógicas, é necessário fazer a conversão para valores digitais utilizando um ***ADC***, conversor analógico-digital, para cada um dos potenciômetros.

Com o kit na posição correta, a leitura analógica dos potenciômetros do joystick é feita através dos pinos GPIO PE3 (vertical) e PE4 (horizontal). O conversor analógico-digital da placa Tiva possui dois módulos ACD que compartilham 20 entradas de 12 bits cada, a conversão pode ser ativada por timers, comparação, PWM, GPIO e software.

2 Leitura e Interrupção através do Botão Pause

A leitura do botão pause é bastante simples, funciona como qualquer outro botão. Na posição correta, o pino PL1 é a conexão da placa que recebe o valor do botão, para a leitura de botões, é necessário ativar, via software, um resistor de *pull-up* , já presente na placa, no pino correspondente, dessa maneira, a leitura funciona com lógica negativa, ou seja, o pino assume o nível lógico baixo enquanto o botão é pressionado e nível alto caso

contrário. Embora não utilizado neste projeto, a leitura do botão do joystick se dá da mesma maneira, apenas mudando a leitura para o pino PC6 da placa.

Para garantir que o jogo pare no menor tempo possível após o pressionamento do botão, é necessário ativar uma interrupção no sistema nesse instante. Assim que o usuário pressiona o botão, uma ISR, rotina de tratamento de interrupção, que deve parar a execução de todas as threads do sistema, é iniciada.

3 Interfaceamento com o Display LCD

Primeiramente, para o funcionamento do backlight do display, é necessário que o jumper J5, logo acima do joystick no booster pack, esteja na posição 2, curto circuitando o segundo e terceiro pino da conexão.

A comunicação entre a placa tiva e o display é feita através do protocolo SSI *Synchronous Serial Interface*, uma comunicação serial sincronizada com o clock da placa. Toda a configuração do display e da comunicação entre ele e a placa é feita pela função "cfaf128x128x16Init()" que foi disponibilizada no arquivo "cfaf128x128x16.c".

Uma vez que o display está configurado, a biblioteca "grlib" do TivaWare pode ser utilizada para facilitar a representação de diversas formas, para esse sistema, as funções que desenham texto e retângulos foram as mais utilizadas, além de outras funções para a configuração do LCD antes de executá-las:

- ***GrRectFill*()** - Desenha e preenche um retângulo especificado pelos dois pontos (x, y) que formam a sua diagonal, a origem das coordenadas do display fica no canto superior esquerdo.
- ***GrStringDraw()*** - Escreve uma string passada por parâmetro no display, através da manipulação de outros parâmetros, é possível escrever em diferentes "linhas" do LCD.
- ***GrFlush()*** - Reseta o contexto do LCD, como fonte e cor de desenho.
- ***GrContextFontSet()*** - Define qual o tamanho da fonte da escrita no LCD.
- ***GrContextForegroundSet()*** - Define qual cor será utilizada, serve para o preenchimento dos retângulos e para a cor dos caracteres da *string*.

4 ThreadX

Por se tratar de um RTOS, sistema de tempo real, o uso de threads se faz necessário, não só para possibilitar a execução "paralela" de mais de um bloco de código, mas também para prover a devida temporização e escalonamento da ordem de execução dos mesmos. A biblioteca *ThreadX* foi utilizada para implementar as quatro threads previstas para o sistema.

Cada thread possui o seu próprio código, que foi detalhado nas subseções 2a-7.X, e sua prioridade com relação às outras threads do sistema. Além disso, algumas threads influenciam em outras, e as seguintes estruturas dessa biblioteca utilizadas para tal foram:

- **Message Queue** - Filas de mensagens foram utilizadas para a thread Joystick enviar os novos comando que o usuário deu ao sistema para a thread Game, e também para a thread Game informar à thread LCD qual o tipo de atualização à ser feita no display para a representação do estado de jogo.
- **Event Flags** - Sinalizadores de eventos, um deles foi utilizado para acordar a thread Pause assim que a rotina de tratamento de interrupção do botão pause for acionada.
- **Thread Sleep** - A temporização do sistema fica a cargo da função *tx_thread_sleep()* que suspende uma tarefa por um determinado número de ticks do sistema, utilizando o clock do sistema em questão, cada tick representa 1ms.