UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Discente: Levy Gabriel da Silva Galvão
Data: 28/07/2020
Disciplina: ELE0621 - ARQUITETURA E PROGRAMAÇÃO DE MICROCOMPUTADORES
Docente: Prof. Dr. José Alberto Nicolau de Oliveira

Aplicação em Assembly: P0NG

Descrição da aplicação:

O trabalho propõe uma aplicação baseada no console do Windows em Assembly utilizando Microsoft Macro Assembler (MASM). A aplicação é um jogo baseado no jogo PONG da década de 1970. A aplicação apesar de ser simples, envolve dois jogadores (*multiplayer* sem inteligência artificial) que são capazes de rebater uma bola que se move na tela do jogo a todo instante.

Desafios:

Um dos maiores desafios a serem explorados no trabalho foi desenvolver uma maneira de mover os objetos no console do Windows de forma independente. Para isso é necessário uma rotina para ler as entradas do console sem que espere que alguma tecla seja pressionada

Apesar da biblioteca do Prof. Irvine Kip fornecer um procedimento capaz de ler um caractere inserido pelo teclado (ReadChar) e se propor a não possuir *echo* ou espera, este, aparentemente espera alguma entrada, de forma que inviabilizou seu uso na aplicação.

A alternativa foi usar a API do windows por meio da função **ReadConsoleInput**. Esta função funciona da forma como esperada, ela lê as entradas do console sem espera. Caso haja alguma tecla pressionada ela é armazenada em uma variável e posteriormente é usada para comparação. Apesar da documentação escassa sobre a API do windows, o trabalho foi facilitado a partir de discussões em fóruns sobre o MASM.

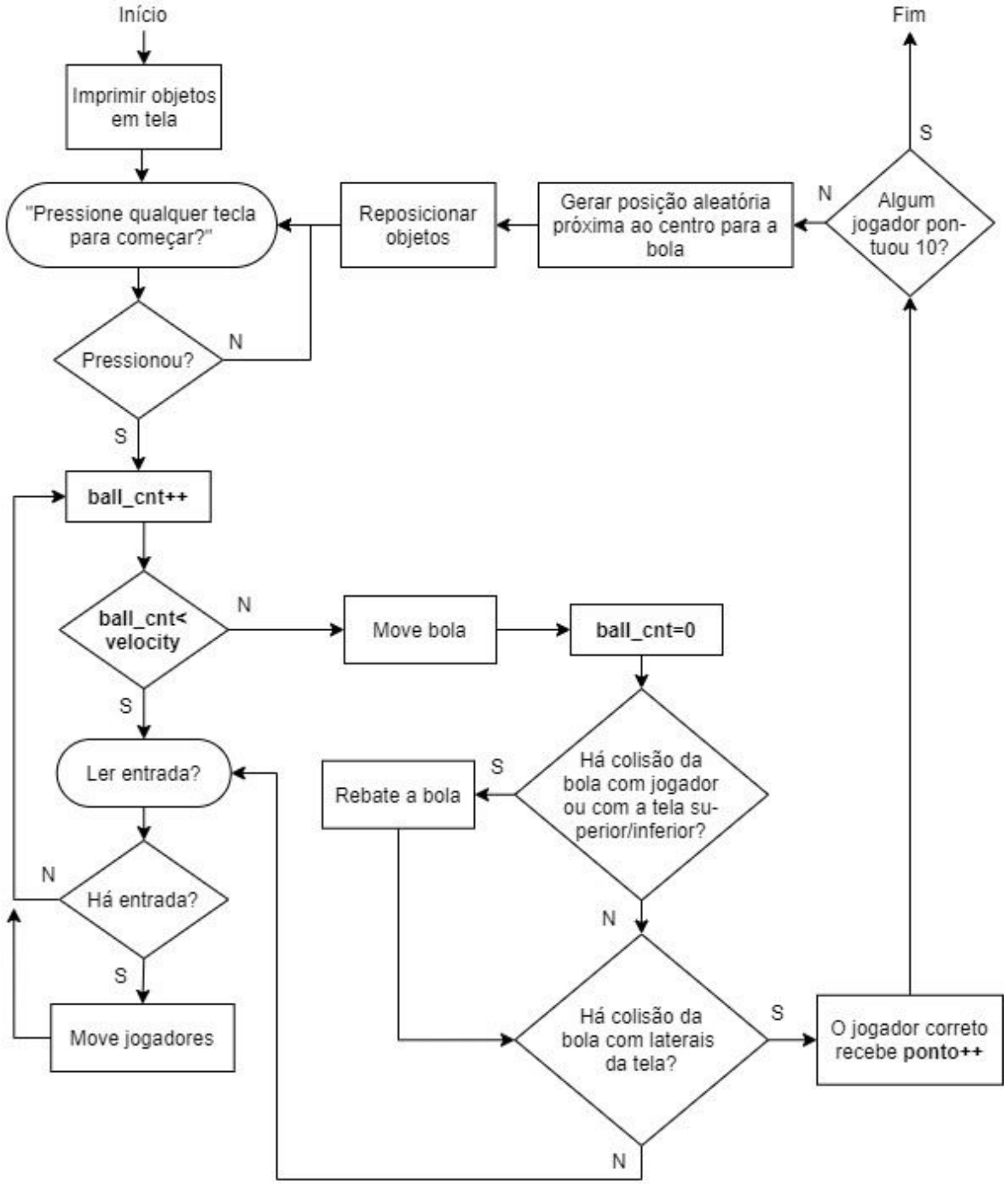
A biblioteca do Irvine possui diversos procedimentos que facilitaram a construção da interface do jogo. Vale destacar o **Gotoxy** que permite posicionar o cursor em qualquer posição do par XY na tela; **Writechar** que sempre foi utilizado com o procedimento anterior para permitir imprimir em qualquer posição da tela o elemento desejado; **Randomrange** foi utilizado apenas uma vez, mas foi um grande facilitador, pois permite gerar um inteiro pseudo randômico na faixa de valores desejado, sendo utilizado em conjunto com **Randomize** (resetar a semente de geração) para determinar a posição de início da bola após um ponto ser marcado; **SetTextColor** foi útil para modificar a cor dos caracteres imprimidos da forma desejada.

Para o movimento da bola, há um contador que é incrementado no *loop* infinito (mesmo *loop* que espera uma entrada do teclado) que ao chegar a certo valor é zerado e a rotina de movimento da bola é executada uma vez. Sem um atraso apropriado, por menor que seja, o código iria executar rapidamente fazendo com que o contador fosse incrementado rapidamente, sendo necessário até 32 bits para o contador para gerar um atraso mínimo. Assim, uma das funções mais úteis do Irvine foi

a **Delay** que foi utilizada para criar um atraso estrutural de 1 ms para garantir o funcionamento correto do código com um contador de largura de byte para garantir o movimento da bola.

Algoritmo:

O algoritmo de funcionamento da aplicação está apresentado no fluxograma abaixo:



Alguns pontos podem ser destacados:

- Colisões com a tela superior ou inferior causa reversão na componente Y da velocidade da bola;
- Colisões com os jogadores causa reversão na componente X da velocidade da bola, mas

caso o jogador esteja indo contra a velocidade Y da bola, a componente Y da velocidade da bola também será revertida;

- A pontuação vai para o jogador na lateral oposta a qual a bola colidiu;
- A variável **velocity** que representa a velocidade do jogo pode ser alteradas com as teclas “1”, “2” ou “3”;
- Pressionando a tecla “X” permite sair do jogo a qualquer momento.