- **UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE**
- **INSTITUTO METRÓPOLE DIGITAL**

Professor: Julio Melo

Trabalho Prático 2 – Threads

Vimos nas aulas que as Threads são uma forma mais amigável, e com menos overhead, de lidar com multiprogramação. Assim como os processos, as *threads* executam em paralelo, em diferentes processadores se possível, porém, elas mantem o contexto de memória do processo principal, o que é uma grande vantagem em termos de memória compartilhada, por exemplo.

No entanto, o fato das *threads* executarem em paralelo causa os mesmos problemas que eram observados nos processos quando tentamos usar memória compartilhada entre elas. Nesse caso, temos ainda que usar mutexes ou semáforos, da mesma forma que era feito nos processos.

Exemplo prático de aplicação com threads

A utilização de threads é corriqueira em sistemas computacionais, muito mais do que a utilização de processos paralelos, inclusive. Uma das aplicações mais simples, mas que é comumente usada é em sistemas que precisam de interface gráfica.

Assim como muitos recursos do sistema operacional, um sistema de interface gráfica é, comumente não reentrante. O que significa que múltiplos programas não podem acessar, ao mesmo tempo, funções que desenham, modificam ou mesmo interagem (um simples *get*, por exemplo) com os elementos da interface.

Dessa forma, é importante manter os componentes que usam a parte de desenho, totalmente isolados, ou garantir que não haverá acesso paralelo a uma mesma função da interface.

Assim, para esse trabalho aplicaremos os conceitos de threads à uma aplicação com interface gráfica. A aplicação a ser desenvolvida foi batizada de Sankes (em homenagem ao jogo da cobrinha). Nesse caso temos múltiplas *Snakes* em um mesmo cenário como mostra a Figura 1:

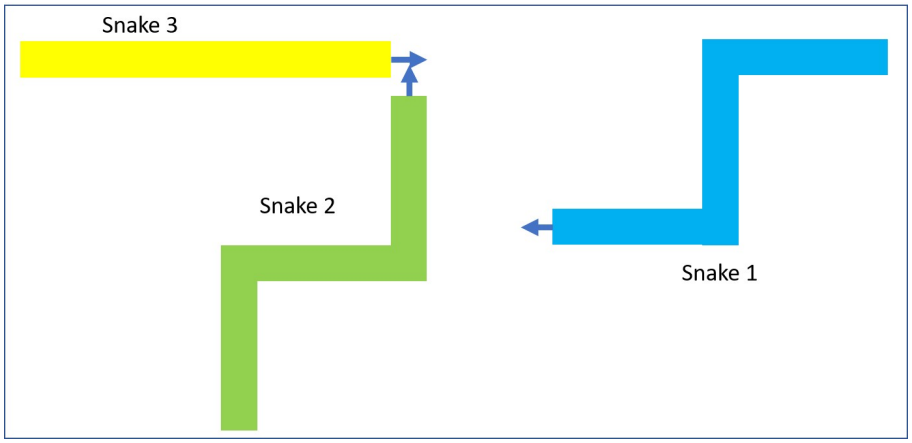

Figura 1, 3 Snakes na tela da aplicação. As setas indicam a direção atual do movimento das Snakes.

Diferentemente do aplicativo original, a nossa aplicação conterá múltiplas snakes num mesmo tabuleiro, que devem seguir as regras do jogo no geral:

- Uma *snake* é destruída se:
  - Colidir com as paredes do cenário
  - Colidir com uma *Snake* (inclusive consigo mesma)
- Uma vez destruída a snake deve ser removida do tabuleiro.

As *snake* andam aleatoriamente pelo tabuleiro e deve ser do controle do usuário quando uma nova *snake* deve ser inserida no tabuleiro. Dessa forma, implemente:

- Um programa principal capaz de desenhar um tabuleiro para a aplicação *Snakes*. A movimentação das *snakes* deve seguir a mesma movimentação do jogo original (procure vídeos na internet sobre a movimentação caso não tenha visto o jogo).
- Cada *snake* deve ser controlada por uma *thread* separada e deve se movimentar aleatoriamente no tabuleiro.
- O usuário deve ser capaz de adicionar uma *snake*, com tamanho especificado, ao tabuleiro usando a interface gráfica.

Player vs (Dummy)IA

A aplicação pode ficar um pouco mais complexa com a adição de uma *snake* controlada por um humano

- Adicione uma opção à interface para adicionar uma *snake* que é controlada por um usuário. Essa *snake* também deve ser controlada por uma *thread* separada.

Modelagem do Programa

Vimos que, em programas paralelos, existem alguns problemas difíceis de modelar e que, muitas vezes requerem ferramentas específicas, com as Redes de Petri, que ajudem na visualização dos estados do programa, por isso é importante sempre pensarmos em como ficaria a documentação e modelagem dos programas paralelos que desenvolvemos, por isso faça:

- Um modelo da aplicação Snakes usando Máquinas de Estado e Diagramas de Sequência.
- (Extra) Um modelo das regiões críticas e acessos concorrentes dessa aplicação usando Redes de Petri.