Aluno: Jhonatas Rodrigues Sousa

Nº Matrícula: 219116813

Relatório: Jogo de Batalha Naval com Sincronização de Threads

Introdução

Este relatório descreve a implementação de um jogo de Batalha Naval utilizando várias técnicas de sincronização de threads para resolver condições de corrida. O objetivo é fornecer uma visão clara do jogo, suas regras e como as técnicas de sincronização, como Mutex, Semáforo, Troca de Mensagens, Barreira e Monitor, são aplicadas para garantir a execução correta do jogo em um ambiente multi-thread.

Descrição do Jogo

Batalha Naval é um jogo de estratégia jogado por dois ou mais jogadores. Cada jogador possui um tabuleiro onde posiciona seus navios de forma oculta ao adversário. O objetivo do jogo é adivinhar as posições dos navios do adversário e afundá-los antes que o adversário afunde os seus.

Regras do Jogo

1. Tabuleiro: Cada jogador tem um tabuleiro de tamanho fixo (neste caso, 5x5).
2. Navios: Os navios são posicionados aleatoriamente no tabuleiro de cada jogador. Cada jogador possui vários navios de tamanhos variados.
3. Turnos: Os jogadores se revezam para adivinhar as posições dos navios adversários, escolhendo coordenadas no tabuleiro do adversário.
4. Acertos e Erros: Se um jogador adivinhar corretamente a posição de um navio adversário, é marcado um acerto; caso contrário, é marcado um erro.
5. Vitória: O jogo termina quando um jogador consegue afundar todos os navios do adversário.

Implementação com Threads e Sincronização

Para simular o jogo em um ambiente multi-thread, utilizamos threads para representar cada jogador e técnicas de sincronização para resolver condições de corrida que podem ocorrer quando múltiplas threads acessam recursos compartilhados.

Threads

As threads são unidades de execução que permitem a execução de múltiplas tarefas simultaneamente. No contexto do nosso jogo, cada jogador é representado por uma thread separada. A função `player_turn` implementa a lógica de cada turno do jogador, permitindo que a thread execute a jogada do jogador.

Detecção de Condições de Corrida

Condições de corrida ocorrem quando múltiplas threads tentam acessar e modificar recursos compartilhados simultaneamente. A função `detect_race_condition` simula a ocorrência de condições de corrida com uma chance de 50%.

```
Python


def detect_race_condition():
    global race_condition_occurred
    if random.random() < 0.5:
        race_condition_occurred = True
        return True
    return False
```

Quando uma condição de corrida é detectada, a função `resolve_race_condition` escolhe uma técnica de sincronização para resolver o problema e imprime uma mensagem indicando a técnica utilizada.

```
Python


def resolve_race_condition():
    global race_condition_occurred
    print(f"Condição de corrida detectada! Utilizando {resolution_method}
para resolver.")
    race_condition_occurred = False
    return resolution_method
```

Técnicas de Sincronização
1. Mutex

Um Mutex (Mutual Exclusion) é um mecanismo de sincronização que garante que apenas uma thread pode acessar um recurso compartilhado por vez. Quando uma thread adquire o mutex, outras threads que tentarem adquiri-lo serão bloqueadas até que o mutex seja liberado.

Uso no Jogo: A função `mark_shot` é protegida por um mutex para garantir que apenas uma thread possa executar a operação de marcar um tiro no tabuleiro ao mesmo tempo.

Vantagens:

- Simplicidade na implementação.
- Eficiente para garantir exclusão mútua em operações críticas.

Desvantagens:

- Pode levar a problemas de desempenho se muitas threads competirem pelo mutex frequentemente.

2. Semáforo

Um Semáforo é um mecanismo de sincronização que controla o acesso a um recurso compartilhado usando um contador. Quando uma thread deseja acessar o recurso, ela deve adquirir o semáforo, o que decrementa o contador. Se o contador for zero, a thread é bloqueada até que outra thread libere o semáforo.

Uso no Jogo: A função `mark_shot` é protegida por um semáforo para garantir que apenas uma thread possa executar a operação de marcar um tiro no tabuleiro ao mesmo tempo.

Vantagens:

- Flexibilidade para permitir múltiplas threads acessarem o recurso simultaneamente até um limite.
- Pode ser usado para limitar a concorrência em um recurso compartilhado.

Desvantagens:

- Complexidade adicional na gestão do contador do semáforo.
- Pode levar a problemas de desempenho se o contador for muito baixo ou muito alto.

3. Troca de Mensagens

A Troca de Mensagens é uma técnica de sincronização onde threads se comunicam através de uma fila de mensagens, evitando acesso direto a dados compartilhados.

Uso no Jogo: A função `mark_shot` utiliza a fila de mensagens para gerenciar a comunicação entre threads, garantindo que a operação de marcar um tiro seja realizada de forma segura.

Vantagens:

- Elimina a necessidade de acesso direto a dados compartilhados, reduzindo a chance de condições de corrida.
- Facilita a comunicação entre threads de maneira segura.

Desvantagens:

- Pode introduzir latência na comunicação devido ao uso da fila de mensagens.
- Complexidade adicional na implementação e gestão da fila de mensagens.

4. Barreira

Uma Barreira é um mecanismo de sincronização que faz com que todas as threads esperem em um ponto de encontro até que todas tenham chegado, garantindo que todas as threads avancem juntas.

Uso no Jogo: A função `mark_shot` utiliza a barreira para sincronizar as threads, garantindo que todas as threads avancem juntas após um ponto de encontro.

Vantagens:

- Garante que todas as threads avancem juntas após um ponto de sincronização.
- Útil para sincronizar operações em fases distintas.

Desvantagens:

- Pode introduzir atrasos se uma thread demorar mais para alcançar a barreira.
- Complexidade adicional na gestão do ponto de encontro.

5. Monitor

Um Monitor é uma estrutura que encapsula dados e operações em um objeto, garantindo que apenas uma thread possa executar uma operação no objeto a qualquer momento.

Uso no Jogo: A função `mark_shot` é protegida por um monitor para garantir acesso exclusivo ao recurso compartilhado.

Vantagens:

- Encapsula dados e operações em um objeto, facilitando a gestão de sincronização.
- Garante exclusão mútua de forma estruturada.

Desvantagens:

- Pode introduzir complexidade adicional na implementação do monitor.
- Pode levar a problemas de desempenho se muitas threads competirem pelo monitor frequentemente.

Comparação das Técnicas de Sincronização

Cada técnica de sincronização tem suas próprias vantagens e desvantagens, e a escolha da técnica adequada depende das características específicas do problema que estamos tentando resolver.

Técnica	Vantagens	Desvantagens
Mutex	Simplicidade, eficiência na exclusão mútua	Problemas de desempenho com alta contenção
Semáforo	Flexibilidade, controle de concorrência	Gestão complexa do contador, impacto no desempenho
Troca de Mensagens	Comunicação segura entre threads, reduz condições de corrida	Latência na comunicação, complexidade na implementação
Barreira	Sincronização em fases, avanço conjunto de threads	Atrasos com threads lentas, complexidade na gestão

Monitor	Encapsulamento de dados e operações, exclusão mútua	Complexidade na implementação, impacto no desempenho