

## Introdução ao jogo Nim

O jogo de Nim é um jogo de estratégia e lógica que envolve dois jogadores e um número (**n**) de palitos disposto numa mesma linha. O objetivo do jogo é ser o último jogador a retirar o último palito da pilha, sendo permitido retirar apenas (1,2 ou 3) palitos de uma única vez.

### Estratégia do jogo

A estratégia no jogo de Nim, onde (**n**) palitos estão dispostos em uma única coluna, baseia-se na manipulação da quantidade de palitos restantes para garantir a vitória. Quando ambos os jogadores jogam perfeitamente, um deles sempre terá a "vantagem". Para manter essa vantagem, a tática é forçar o adversário a jogar com uma quantidade de palitos que seja um múltiplo de 4.

Ao assegurar que o oponente se encontre nessa situação, você estabelece um cenário em que, na última jogada dele, restarão apenas 4 palitos. Como, de acordo com as regras do jogo, o jogador só pode retirar 1, 2 ou 3 palitos, não importa a escolha feita, você garantirá a vitória. Essa abordagem se fundamenta na premissa de que controlar as jogadas do adversário é crucial para determinar o resultado final do jogo.

Exemplo:

```
Digite a quantidade inicial de palitos: 10
| | | | | | | |
Numero de palitos restantes: 10
Digite quantos palitos pegar (1,2 ou 3): 2
| | | | | | |
Numero de palitos restantes: 8
O computador pegou 3 palito(s)
| | | |
Numero de palitos restantes: 5
Digite quantos palitos pegar (1,2 ou 3): 1
| | |
Numero de palitos restantes: 4
O computador pegou 3 palito(s)
|
Numero de palitos restantes: 1
Digite quantos palitos pegar (1,2 ou 3): 1
Fim de jogo -> O Jogador1 Venceu
```

### Arquitetura do Programa

O programa foi escrito em C++ e estruturado de forma procedural, o que significa que suas funcionalidades foram divididas em pequenas funções, cada uma com seus próprios métodos. Essa abordagem facilita a organização e a manutenção do código, permitindo que cada função tenha uma responsabilidade específica.

### Funções do Programa

#### Main()

Essa função é responsável por inicializar a quantidade de palitos que haverá no jogo na variável **npalitos** e chamar outras funções que serão apresentadas posteriormente. Ela inclui redundâncias importantes, como impedir que o jogo comece até que um número válido de palitos, maior que zero, seja digitado. Além disso, essa função oferece um recurso que permite reiniciar o jogo, proporcionando flexibilidade ao usuário.

```

int main(){
int npalitos = 0; //variavel que recebe a quantidade de palitos do jogo
cout<<"Digite a quantidade inicial de palitos: "; //pergunta ao jogador o número de palitos
cin>>npalitos; cout<<"\n";
while (npalitos <= 0) { // redundância para que a quantidade de palitos necessaria pra comecar o jogo seja >0
    cout <<"Quantidade invalida, digite um numero > 0 para iniciar o jogo"<<"\n";
    cout<<"Digite a quantidade inicial de palitos: ";
    cin >> npalitos; cout<<"\n";
}
JogoNim(&npalitos); //chama a função JogoNim

string reiniciar;
cout<<"\n"<<"\n"<<"Jogar Novamente? ";
cin>>reiniciar;
if(reiniciar == "sim" || reiniciar == "SIM"){ // da a opção do jogador reiniciar o jogo sem ter que fechar o terminal
    system("cls"); //limpa o terminal, funciona apenas em Windows, Apple ou Linux usar system("clear")
    main(); //função recursiva
}
return 0;
}

```

## TabuNim()

Essa função é responsável por exibir o número de palitos definidos na função main() no terminal. Além de apresentar a quantidade inicial de palitos, ela também será utilizada para mostrar o número de palitos restantes após a jogada ambos os jogadores. A cada turno, a função é chamada para mostrar visualmente a nova quantidade de palitos, garantindo que os jogadores saibam quantos restam até o final do jogo.

```

void Tabunim(int *npalitos){ //cria o método que imprime o tabuleiro do Nim no terminal

for(int loop = 0; loop<*npalitos; loop++){ //imprime a quantidade de palitos que foi escolhido
    cout<<"| ";
}
cout<<"\n"<<"Numero de palitos restantes: "<<*npalitos<<"\n";
}

```

## Jogador1 ()

Essa função é responsável por criar os métodos relacionados às jogadas do jogador1. Ela gerencia a interação do usuário com o jogo, permitindo apenas jogadas que estão dentro das regras estabelecidas. Além disso, possui algumas redundâncias para melhorar a coerência do jogo.

```

void jogador1(int *npalitos){ //os métodos para o jogador 1
int tpalitos; //recebe a quantidade de palitos que vão ser retirados

cout<<"Digite quantos palitos pegar (1,2 ou 3): ";
cin>>tpalitos;
if(tpalitos > 0 && tpalitos<=3){ //verifica se a quantidade de palitos que vão ser retirados está entre (1,2 ou 3)
    if(tpalitos <= *npalitos){ //verifica se o número de palitos retirado é menor do que há no jogo
        *npalitos=*npalitos-tpalitos; //retira os palitos
    }else{ //caso o numero de palitos retirado seja maior que a quantidade de palitos que sobraram no jogo
        cout<<"\n"<<"Quantidade indevida, sobrou apenas "<<*npalitos<<" palito(s) no jogo"<<"\n"<<"\n";
        Tabunim(npalitos); //mostra de novo o tabuleiro pro jogador não se perder
        jogador1(npalitos); //função recursiva
    }
}else{ //caso o número de palitos retirados não esteja no intervalo [1,3]
    cout<<"\n"<<"Quantidade invalida, pode retirar apenas (1,2 ou 3) palitos de uma vez "<<"\n"<<"\n";
    Tabunim(npalitos); //mostra de novo o tabuleiro pro jogador não se perder
    jogador1(npalitos); //função recursiva
}
}

```

1.1 Primeiramente o Programa verifica se a entrada na variável **tpalito** (variável responsável por armazenar a quantidade de palitos que vão ser retirados) está entre (1,2 ou 3). Caso não seja um valor válido, ele imprime um erro no terminal, e pede para escolher outro número. Caso seja escolhido um número dentro do intervalo estabelecido pelas regras, ele avança para a próxima verificação.

1.2 Nessa próxima etapa, o programa verifica se a quantidade de palitos que é retirado é menor que a quantidade de palitos que sobraram no jogo, essa verificação serve por exemplo para que quando reste 2 palitos não seja possível retirar 3 palitos. Caso seja tentado retirar um valor maior do que há no jogo, ele imprime um erro no terminal e pede para escolher outro número.

1.3 Passado por todas essas verificações, o programa atribui na variável **tpalitos** a quantidade de palitos escolhida pelo jogador1 para ser retirado do jogo.

### bot()

Essa função é responsável por determinar os métodos relacionado as jogadas do computador. Ela define a lógica que o computador seguirá a cada turno, levando em consideração as estratégias discutidas anteriormente. Dependendo do estado atual do jogo, a função toma decisões sobre quantos palitos o computador deve retirar, usando uma estratégia de minimização do risco ou de maximização da vantagem.

```
void bot(int *npalitos) { //os métodos para o bot
    if(*npalitos%4 == 0) { //verifica se a quantidade de palitos que sobrou da vez do jogador1 é multiplo de 4
        int aleatorio = rand() % 3 + 1; //Pega um número aleatório entre 1 e 3
        *npalitos = *npalitos - aleatorio; //tira uma quantidade aleatória de palitos
        cout<<"O computador pegou "<<aleatorio<<" palito(s)"<<"\n";
    } else { //verifica quando a quantidade de palito que sobrou da vez do jogador1 não é multiplo de 4
        int loop = 0;

        if(*npalitos>4) { //verifica se há mais de 4 palitos
            while ((*npalitos-loop)%4 != 0) { //verifica quantos palitos precisa tirar para sobrar um multiplo de 4
                loop++;
            }
        } else { //quando sobra menos de 4 palitos
            while ((*npalitos-loop) != 0) { //verifica quantos palitos precisa tirar para acabar com o jogo
                loop++;
            }
        }
        *npalitos = *npalitos - loop; //tira os palitos
        cout<<"O computador pegou "<<loop<<" palito(s)"<<"\n";
    }
}
```

- 1.1 Primeiramente o programa começa verificando se a quantidade de palitos restantes, após a jogada do jogador 1, é um múltiplo de 4. Se o número de palitos for um múltiplo de 4, isso indica que o computador está em desvantagem. Neste caso, como a escolha do movimento não terá um impacto significativo no resultado do jogo, o computador opta por remover um número "aleatório" de palitos. Essa estratégia é baseada na observação de que, se a situação de múltiplos de 4 persistir até o final do jogo, o computador provavelmente perderá.
- 1.2 Caso a quantidade de palitos restantes, após a jogada do jogador 1, não seja um múltiplo de 4, o computador tem a oportunidade de assumir a vantagem no jogo. Nesse cenário, o computador avança para uma nova verificação
- 1.3 Nessa nova etapa o computador verifica se a quantidade de palitos no jogo é maior que 4. Caso for, ele calcula quantos palitos deve ser retirado para que o número restante se torne um múltiplo de 4. Essa estratégia permite que o computador controle a dinâmica do jogo e aumente suas chances de vitória.
- 1.4 Se a quantidade de palitos restantes, for menor do que 4 palitos, o computador analisa quantos palitos deve retirar para garantir a sua vitória.

## Jogonim()

Essa função é responsável por determinar a ordem dos jogadores e inclui um método para verificar qual deles é o vencedor. Além disso, a função atualiza a quantidade de palitos exibida no terminal após cada jogada, garantindo que todos os jogadores estejam cientes do estado atual do jogo.

```
int JogoNim(int *npalitos) { //Junta todos os métodos numa função só
    Tabunim(npalitos); //inicia o tabuleiro
    jogador1(npalitos); //Inicia o método do jogador
    if(*npalitos==0){ //Verifica-se o jogador já venceu
        cout<<"Fim de jogo -> O Jogador1 Venceu ";
        return 0;
    }

    Tabunim(npalitos); //atualiza o tabuleiro
    bot(npalitos); //inicia o método do bot
    if(*npalitos==0){ //Verifica-se o Bot já venceu
        cout<<"Fim de jogo -> O Computador Venceu ";
        return 0;
    }
    JogoNim(npalitos); //função recursiva caso o jogo ainda não tenha terminado
}
```