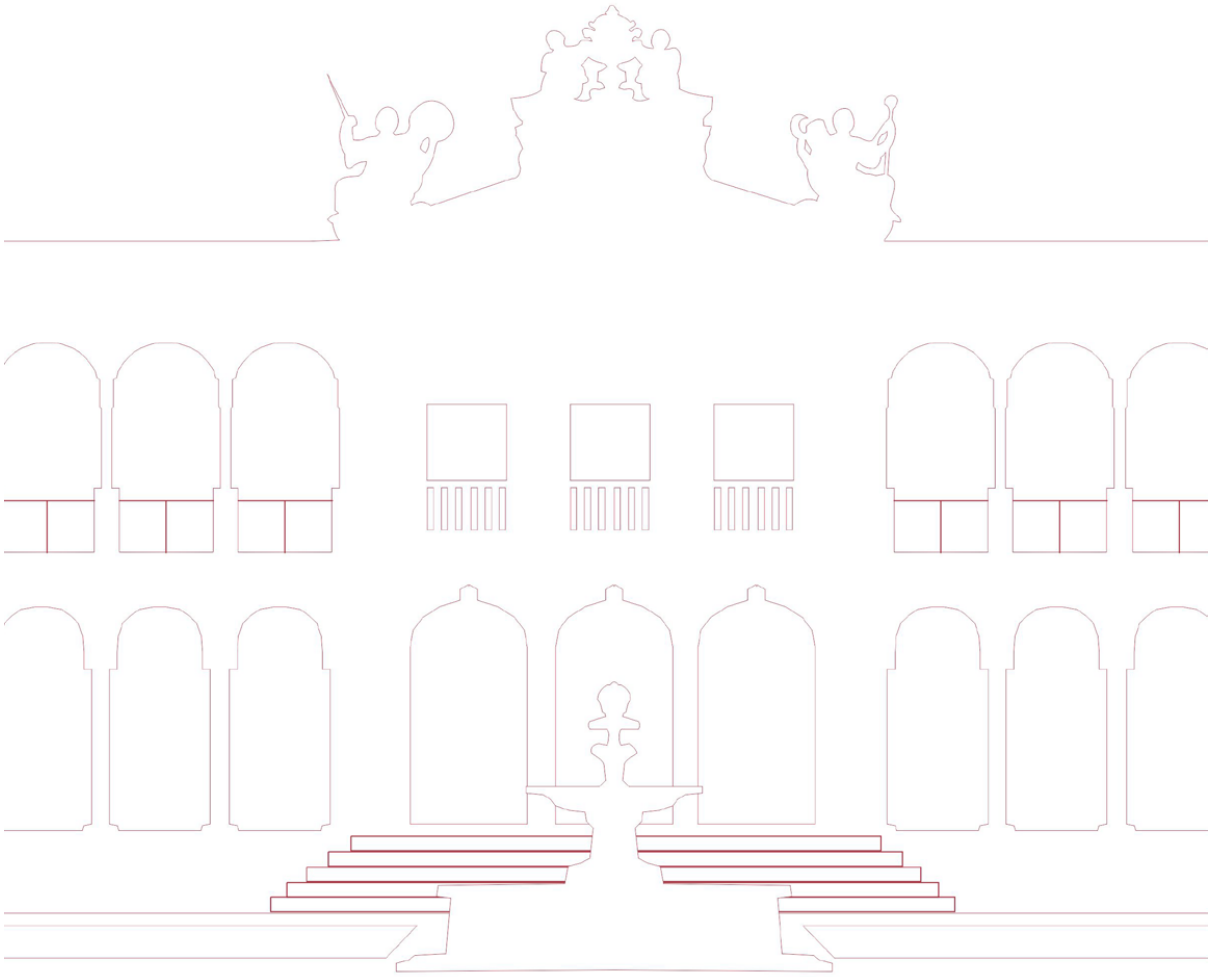
# Paktris - Trabalho prático de P3

UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Licenciatura em Eng. Informática



**Helder Godinho nº42741**
**Gustavo Oliveira nº46395**

*Évora, 20 de janeiro de 2024*

# Conteúdo

# 1 Introdução

O Paktris, inspirado no clássico jogo Tetris, desafia os jogadores a manipularem peças geométricas enquanto tentam preencher linhas completas em um tabuleiro de células. Neste projeto, foram consideradas quatro peças distintas: I, S, O e T, cada uma com diferentes formas e suscetíveis a reorientações por rotação. A tarefa consiste em criar um programa que aceite uma lista de jogadas, representadas por triplos (PEÇA, NROT, NDIR), onde PEÇA é o tipo de peça, NROT indica a quantidade de rotações, e NDIR representa o deslocamento para a direita antes da peça ser largada.

O objetivo principal é validar se a sequência de jogadas fornecida cabe no tabuleiro, seguindo as regras do Tetris. O tabuleiro é definido por uma matriz de NxM células, e as peças devem ser posicionadas de forma a não ultrapassar os limites do tabuleiro ou ficar acima do topo. Em caso de sucesso, o programa deve exibir o tabuleiro final após as jogadas.

Este relatório detalhará a implementação do Paktris, destacando a lógica por trás das rotações e deslocamentos das peças, a verificação da adequação no tabuleiro e a apresentação visual do resultado. No final, será possível observar o Paktris em ação, proporcionando uma compreensão abrangente de seu funcionamento e desafios.

# 2 Metodologia

## 2.1 Estrutura do código

1. **Definição de Tipo:**
   - O tipo peca define as diferentes peças do Tetris, incluindo rotações. O uso de um tipo variante é uma boa escolha para representar a variedade de peças.
2. **Função de Rotação:**
   - A função rotate_piece lida efetivamente com a rotação de cada peça do Tetris. O uso de declarações match aninhadas garante rotações corretas com base no tipo de peça e na contagem de rotação especificada.
3. **Verificação de Posicionamento:**
   - A função check_fit verifica se uma peça pode ser colocada no tabuleiro em uma determinada posição. Ela considera o estado atual do tabuleiro e garante que a peça não sobreponha blocos existentes.
4. **Configuração da Peça:**
   - A função piece_cells gera uma lista de células para uma determinada peça e sua orientação. Essa abstração torna o código mais legível e permite fácil modificação das configurações das peças.
5. **Tentativa de Posicionamento:**
   - A função try_place_piece tenta colocar uma peça em uma posição específica no tabuleiro. Ela tenta diferentes deslocamentos iterativamente até encontrar uma posição válida. Isso garante que as peças caiam até o fundo da coluna.
6. **Loop Principal:**
   - A função loop itera pela lista de movimentos e tenta colocar cada peça no tabuleiro. Se uma colocação válida for encontrada para todas as peças, ela retorna true; caso contrário, retorna false.
7. **Função de Impressão:**
   - A função imprimir_tabuleiro imprime o tabuleiro, fornecendo uma representação visual do estado final após o processamento de todos os movimentos. Isso é útil para depuração e compreensão do resultado.
8. **Tratamento de Erros:**
   - O código inclui algum tratamento de erros, como o uso de failwith para casos impossíveis na função rotate_piece. Isso ajuda a identificar situações inesperadas durante o desenvolvimento.

## 3 Resultados

O programa Paktris demonstrou eficiência na validação de sequências de jogadas e na visualização do tabuleiro após as manipulações das peças. Durante a implementação, focámos em garantir a precisão das rotações, o correto posicionamento das peças e a deteção de possíveis sobreposições no tabuleiro.
Na implementação do programa, optamos por adotar uma representação da matriz do tabuleiro em que as células ocupadas por uma peça são marcadas com o valor **1**, enquanto as células vazias permanecem com o valor **0**. Esta escolha facilita a visualização do estado do tabuleiro após todas as jogadas serem realizadas.
Considere então os seguintes exemplos do funcionamento do nosso trabalho:

```
tabuleiro = 4x4

# paktris [(I,0,0); (I,0,0); (O,0,2); (O,0,0)];;
```

```
# paktris [(I,0,0); (I,0,0); (O,0,2); (O,0,0)];;
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
true
- : unit = ()
```

```
# paktris [(I,1,0); (S,1,1); (O,0,1)];;
```

```
# paktris [(I,1,0); (S,1,1); (O,0,1)];;
1 0 0 0
1 0 1 0
1 1 1 0
1 1 0 0
false
- : unit = ()
```

```
tabuleiro = 10x7

# paktris[(T,3,0);(T,1,2);(T,0,2);(O,0,6);(O,0,8);(S,0,7)];;
```

```
# paktris[(T,3,0);(T,1,2);(T,0,2);(O,0,6);(O,0,8);(S,0,7)];;
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0 0 0
0 0 0 1 0 0 0 0 1 1
1 0 0 1 0 0 0 1 1 0
1 1 1 1 0 0 1 1 1 1
1 0 0 1 0 0 1 1 1 1
true
- : unit = ()
```

tabuleiro = 7x10

# paktris[(T,3,0);(T,1,2);(T,0,2);(O,0,6);(O,0,8);(S,0,7)];;

```
# paktris[(T,3,0);(T,1,2);(T,0,2);(O,0,6);(O,0,8);(S,0,7)];;
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 0 0 1 0 0 0
1 0 0 1 0 0 0
1 1 1 1 0 0 0
1 0 0 1 0 0 0
false
- : unit = ()
```

# 4 Conclusão

O desenvolvimento deste projeto proporcionou uma oportunidade valiosa para explorar e aplicar conceitos fundamentais de programação funcional na linguagem OCaml. A implementação do jogo **"Paktris"** permitiu-nos abordar diversos aspetos, como rotação de peças, colocação no tabuleiro e validação do encaixe.

Ao longo do processo, enfrentamos desafios relacionados à manipulação de matrizes, coordenação de movimentos e definição de padrões para as diferentes peças. A estratégia de dividir o problema em funções modulares, como rotate_piece e check_fit, facilitou a compreensão e manutenção do código.

A capacidade de visualizar o tabuleiro no final das jogadas proporcionou uma perspetiva clara do resultado final e ajudou na depuração do código. A representação gráfica ASCII demonstrou ser uma ferramenta eficaz para acompanhar as mudanças no estado do tabuleiro.

Apesar dos sucessos alcançados, há espaço para melhorias futuras, como a implementação de uma interface gráfica mais sofisticada para melhorar a experiência do utilizador. A realização deste projeto reforçou a importância do raciocínio lógico na programação e consolidou os conhecimentos em programação funcional.