



Documentação Projeto Mobile: NhamNham

Aluno: Pedro Arthur Cunha Anício

Sumário

1. Introdução -----	3
2. Objetivos -----	3
3. Decisões de implementação -----	3
4. Visão geral do funcionamento -----	3
5. Testes executados-----	4
6. Descrição dos módulos e sua interdependência -----	4
7. Descrição dos códigos -----	5
8. Problemas Encontrados -----	9
9. Conclusão -----	9

1. Introdução

O "Nham Nham" é um jogo de tabuleiro digital desenvolvido para dispositivos Android. Ele simula um jogo de estratégia onde dois jogadores alternam movimentos em um tabuleiro 3x3, buscando alinhar três peças de mesma cor sobrepondo peças menores com maiores.

2. Objetivos

- Implementar um jogo de tabuleiro com conceitos de POO;
- Proporcionar uma interface simples e responsiva para os jogadores, desenvolvidas a partir dos layouts do Android Studio;
- Garantir uma lógica robusta e funcional para manipulação de regras do jogo.

3. Decisões de implementação

- Linguagem e plataforma: Projeto desenvolvido na linguagem Kotlin, para as regras de negócio, objetos e manipulação dos dados. Interface gráfica projetada utilizando os componentes nativos do Android Studio;
- Estrutura do projeto: Foi adotada uma arquitetura simples modular, dividindo o código em classes claras como “GameActivity”, “PieceSize”, “RulesActivity”.
- Regras do jogo: As regras foram implementadas com base no conceito de alinhamento de peças, incluindo a funcionalidade de peças maiores cobrirem menores, o que acrescenta estratégia ao jogo.

4. Visão geral do funcionamento

- Tela inicial: O jogador é recebido pela tela inicial que apresenta as funções de iniciar o jogo, mostrar as regras e sair do jogo;
- Tabuleiro: Tabuleiro 3x3 onde as peças são posicionadas. O jogador escolhe o tamanho da peça a ser posicionada e clica no campo desejado.
- Mecânica de jogo: A cada jogada, são verificadas as condições de vitória, empate ou continuidade. Caso um jogador vença ou o jogo fique empatado, é exibido uma mensagem para sair ou reiniciar o jogo.

5. Testes executados

Foram executados, sobretudo, testes funcionais em dispositivos Android para verificar o funcionamento correto de todas as ações do jogo.

6. Descrição dos módulos e sua interdependência

I. MainActivity:

- Tela inicial do jogo;
- Apresenta um botão para iniciar o jogo (GameActivity) e outro para exibir as regras(RulesActivity);
- Depende diretamente de GameActivity e RulesActivity.

II. GameActivity:

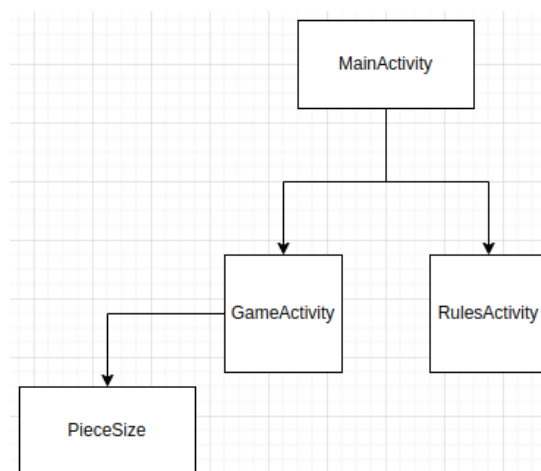
- Gerencia a lógica do jogo principal e a interface com o tabuleiro.
- Contém a lógica de interação dos jogadores, como seleção de peças, movimentação no tabuleiro e alternância entre turnos.
- Interage com a enumeração PieceSize para determinar o tamanho das peças e validações.
- Exibe resultados e gerencia o estado do tabuleiro e das peças disponíveis.
- Tem dependências com recursos do layout (R.layout.activity_game) e imagens das peças (R.drawable).

III. RulesActivity:

- Apresenta as regras do jogo.
- Possui uma dependência apenas para retornar à MainActivity.

IV. PieceSize:

- Enumeração que define os tamanhos das peças disponíveis (PEQUENO, MEDIO, GRANDE) e seus valores associados.
- Usada no GameActivity para lógica de validação e exibição.



7. Descrição dos códigos

I. onCreate:

```
override fun onCreate(savedInstanceState: Bundle?) {  📌 pedroanicio
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_game)

    gridBoard = findViewById(R.id.gridBoard)
    txtCurrentPlayer = findViewById(R.id.txtCurrentPlayer)
    spinnerPieceSize = findViewById(R.id.spinnerPieceSize)

    val btnRestart = findViewById<Button>(R.id.btnRestart)

    setupBoard()
    setupSpinner()

    btnRestart.setOnClickListener {
        resetGame()
    }

    // voltar à tela inicial
    val btnBack = findViewById<Button>(R.id.btnVoltar)
    btnBack.setOnClickListener {
        finish()
    }
}
```


- Esse método tem como função configurar os elementos visuais como tabuleiro;
- Define ações para botões:
 - Reiniciar o jogo (resetGame()).
 - Voltar para a tela inicial (finish()).
- Inicializa o tabuleiro (setupBoard()) e o spinner (setupSpinner()).

II. SetupBoard:

```
private fun setupBoard() {  pedroanicio
    for (i in 0 until 3) {
        for (j in 0 until 3) {
            val cell = gridBoard.getChildAt(index: i * 3 + j) as ImageView
            cell.setImageResource(0) // Limpa a imagem
            cell.setOnClickListener {
                onCellClicked(cell, i, j)
            }
        }
    }
}
```

- Configura um loop para percorrer as 9 células (3x3) do tabuleiro.
- Limpa a imagem de cada célula (setImageResource(0)).
- Define um listener para cada célula, que chama onCellClicked ao ser clicada, com as coordenadas da célula.

III. OnCellClicked:

```
private fun onCellClicked(cell: ImageView, row: Int, col: Int) { 
    val selectedSize = when (spinnerPieceSize.selectedItem.toString()) {
        "PEQUENO" -> PieceSize.PEQUENO
        "MEDIO" -> PieceSize.MEDIO
        "GRANDE" -> PieceSize.GRANDE
        else -> return
    }

    val currentPiece = board[row][col]
    if (currentPiece.first.isEmpty() || selectedSize.value > currentPiece.second.value) {
        // Atualizar tabuleiro
        board[row][col] = Pair(currentPlayer, selectedSize)

        // Define a imagem correspondente à peça
        val pieceImageRes = getPieceImageResource(currentPlayer, selectedSize)
        cell.setImageResource(pieceImageRes)

        // Remover peça usada do jogador
        piecesAvailable[currentPlayer]?.remove(selectedSize)

        // Verificar vencedor ou alternar jogador
        if (checkWinner()) {
            txtCurrentPlayer.text = "$currentPlayer venceu!"
            disableBoard()
        } else {
            switchPlayer()
            updateSpinner()
        }
    }
}
```

- Obtém o tamanho da peça selecionada no spinner.
- Verifica se a célula está vazia ou se a peça selecionada é maior do que a existente.

Atualiza:

- Tabuleiro lógico (board[row][col]).
- Imagem da célula (setImageResource).
- Lista de peças disponíveis do jogador.
- Checa se houve um vencedor:
 - Caso positivo: exibe mensagem e desabilita o tabuleiro.
 - Caso negativo: alterna o jogador e atualiza o spinner.

IV. SwitchPlayer:

```
private fun switchPlayer() {  @pedroanicio
    currentPlayer = if (currentPlayer == "Jogador 1") "Jogador 2" else "Jogador 1"
    txtCurrentPlayer.text = "Vez do $currentPlayer"
}
```

- Alterna o jogador atual entre "Jogador 1" e "Jogador 2".
- Atualiza o texto na interface para indicar de quem é a vez.

V. CheckWinner:

```
private fun checkWinner(): Boolean {  @pedroanicio
    for (i in 0 until 3) {
        if (board[i][0].first == currentPlayer && board[i][1].first == currentPlayer && board[i][2].first == currentPlayer) {
            disableBoard()
            showWinnerDialog("$currentPlayer venceu!")
            return true
        }
        if (board[0][i].first == currentPlayer && board[1][i].first == currentPlayer && board[2][i].first == currentPlayer) {
            disableBoard()
            showWinnerDialog("$currentPlayer venceu!")
            return true
        }
    }
    if (board[0][0].first == currentPlayer && board[1][1].first == currentPlayer && board[2][2].first == currentPlayer) {
        disableBoard()
        showWinnerDialog("$currentPlayer venceu!")
        return true
    }
    if (board[0][2].first == currentPlayer && board[1][1].first == currentPlayer && board[2][0].first == currentPlayer) {
        disableBoard()
        showWinnerDialog("$currentPlayer venceu!")
        return true
    }
    return false
}
```

- Verifica vitórias nas linhas e colunas.
- Verifica vitórias nas diagonais principais e secundárias.
- Retorna true se uma condição de vitória for atendida; caso contrário, false.

8. Problemas Encontrados

Durante o desenvolvimento do jogo "Nham Nham", enfrentei desafios como implementar corretamente a lógica de sobreposição de peças no tabuleiro, garantir a verificação de vitória considerando peças empilhadas e manter a interface responsiva para diferentes dispositivos. Além disso, houve dificuldades em preservar o estado do jogo após a rotação da tela e em ajustar o spinner de seleção de tamanhos para refletir as peças disponíveis. Esses problemas foram solucionados com ajustes na lógica do código, uso adequado de layouts no XML e implementação de métodos para salvar o estado da aplicação.

9. Conclusão

A criação do jogo "Nham Nham" foi uma vivência valiosa, possibilitando a aplicação de princípios de Programação Orientada a Objetos e a criação de interfaces para Android. Mesmo com os obstáculos encontrados, o projeto gerou uma aplicação útil, interativa e que cumpre os requisitos estabelecidos. Esta vivência destacou a necessidade de um planejamento meticuloso, testes frequentes e uma estratégia iterativa para a resolução de problemas no desenvolvimento de software.