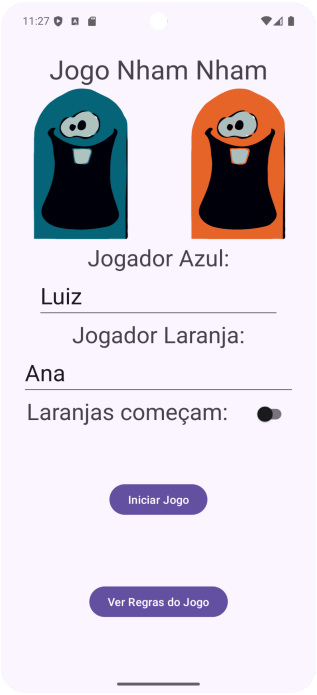
Instituto Federal de Minas Gerais - Campus Ouro Branco

Docente: Saulo Henrique Cabral Silva
Discente: Luiz Filipe Ferreira Ramos

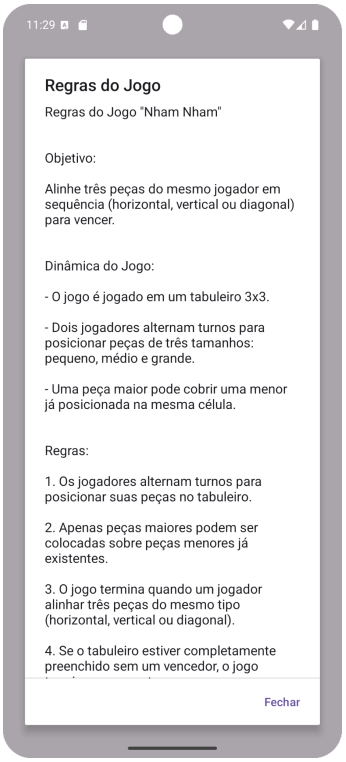
Trabalho Prático 3
Nham Nham

1. Introdução
Este trabalho tem como objetivo apresentar a implementação de um jogo chamado "Nham Nham", desenvolvido no contexto do ensino de Programação para Dispositivos Móveis. O jogo permite que dois jogadores se enfrentem em uma partida dinâmica em uma grade 3x3, com peças que podem ser movidas por arrasto, comer peças em um tabuleiro, e com regras simples de interação e vitória. A implementação do jogo foi realizada utilizando a linguagem de programação Java para Android.

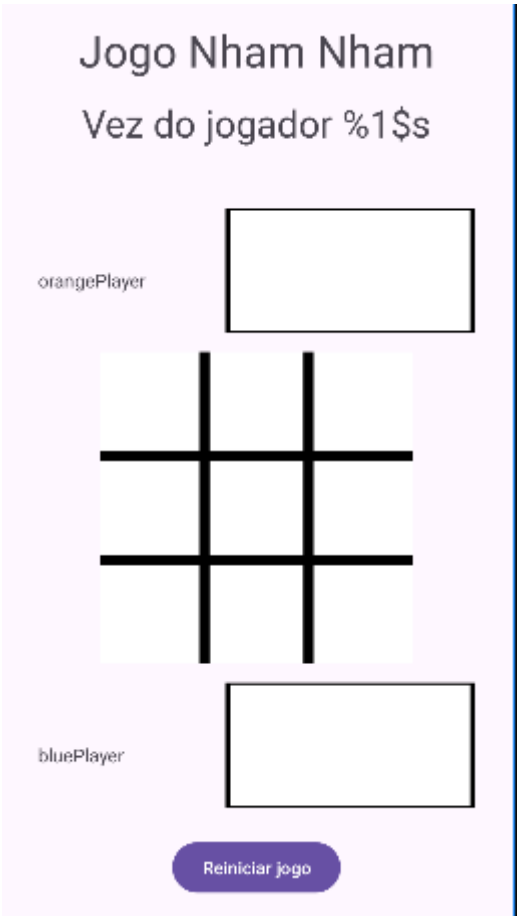
2. Implementação
2.1 Layouts
> activity_main.xml:
Este é o layout da tela inicial. Nela tem uma arte das duas pecinhas que são usadas durante o jogo. É possível digitar o nome do jogador laranja e azul, e determinar por meio de um switch que cor começa o jogo. O botão de Iniciar Jogo leva para uma nova Acitivity que possui o jogo completo. O botão Ver Regras do Jogo exibe um PopUp com elas.



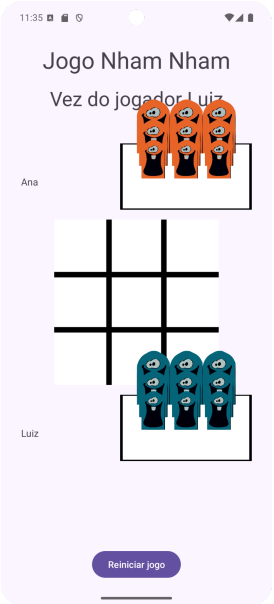
PopUp de regras:



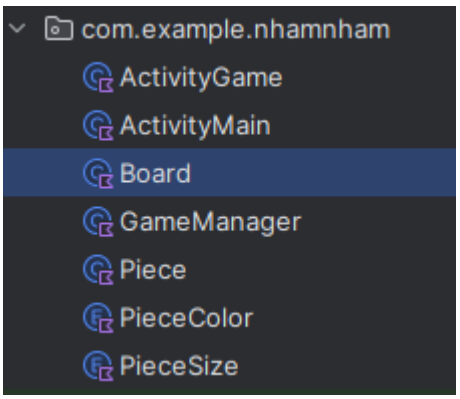
> activity_game.xml:



Nele, a existe uma ImageView para cada casinha inicial das pecinhas e uma uma ImageView para o tabuleiro central. Elas são definidas no Layout como referência para posteriormente, de forma programática, posicionar dinamicamente as peças.
Tela com o jogo funcionando:



2.2 Classes do Jogo.
Estão são as classes do jogo:



> Classe MainActivity
É a atividade principal do aplicativo e serve como ponto de entrada para o jogo. Ela gerencia a interface de configuração inicial, onde o usuário pode inserir seus nomes e escolher quem começará a jogar. Após a configuração, a classe redireciona o usuário para a tela do jogo, onde a interação com o tabuleiro ocorre. A classe também permite exibir as regras do jogo ao apertar um botão.

> Classe ActivityGame:
É responsável por gerenciar a tela principal do jogo, onde os jogadores interagem com o tabuleiro.

Ao ser iniciada, a ActivityGame recebe informações enviadas pela ActivityMain através de uma Intent, incluindo os nomes dos jogadores e a configuração de qual cor começa a partida. Essas informações são usadas para configurar a interface do jogo, exibindo os nomes dos jogadores nos respectivos lados do tabuleiro.

A classe instancia e utiliza o GameManager, que é responsável por gerenciar o jogo.

Além disso, a ActivityGame oferece um botão de reset que permite reiniciar a partida com os mesmos jogadores e configurações. O botão cria uma nova Intent para recarregar a tela, limpando o estado atual do jogo e voltando à configuração inicial sem precisar retornar à ActivityMain.

> Classe GameManager:

Ela é responsável por gerenciar a lógica central do jogo, incluindo o estado do tabuleiro, a interação com as peças e o fluxo da partida.

Métodos:
setupGame(): Inicializa o jogo, configurando o tabuleiro e posicionando as peças em suas localizações iniciais.
canDragPiece(piece: Piece): Boolean: Verifica se uma peça específica pode ser arrastada, com base no turno atual e no estado do jogo.
handleInBoardPositioned(piece: Piece): Atualiza o estado do jogo quando uma peça é posicionada no tabuleiro, alternando o turno, verificando condições de vitória ou empate e atualizando a interface.
addPiecesDynamically(): Cria as peças dinamicamente para ambos os jogadores e as posiciona em suas áreas iniciais, usando de medições dos componentes já renderizados.
createPieces(color: PieceColor, targetList: MutableList<MutableList<Piece>>): Cria um conjunto de peças de uma cor específica, associando-as a uma lista-alvo.
setPiecesToInitialPos(referenceView: View, targetList: MutableList<MutableList<Piece>>): Posiciona as peças em suas posições iniciais na interface, com base em um ponto de referência.
Métodos estáticos:
makeDraggableObject(guiObject: View, actionUpHandler: (View) -> Unit, canDrag: () -> Boolean): Transforma um objeto da interface em um elemento arrastável, com comportamentos definidos para os eventos de toque (movimentação e soltura).
removeDraggability(guiObject: View): Remove a funcionalidade de arrastar de um objeto da interface.
Essa classe gerencia o estado do jogo, o comportamento das peças e as interações dos jogadores com o tabuleiro, funcionando como o núcleo lógico do aplicativo.

>Classe Piece:
Ela representa uma peça do jogo, incluindo suas propriedades, comportamento e interação com o tabuleiro e o layout da interface.

Métodos:
createPiece(gameAreaLayout: FrameLayout): Adiciona a peça ao layout da interface e a torna arrastável.
canDrag(): Boolean: Verifica se a peça pode ser arrastada, consultando o GameManager.

adjustPiecePosition(piece: View): Ajusta a posição da peça ao soltá-la. Reposiciona-a no tabuleiro se válido; caso contrário, retorna à posição inicial.
positionPiece(referenceView: View, numOfPiecesInLine: Int, inLinePosition: Int, numOfPiecesInColumn: Int, inColumnPosition: Int): Calcula e define a posição inicial da peça em relação a um elemento de referência.
getCenter(): PointF: Obtém as coordenadas do centro da peça para cálculos de posicionamento.
*Métodos auxiliares:
calculateHeight(pieceWidth: Int): Int: Calcula a altura proporcional da peça com base na largura e na proporção do drawable associado.
Essa classe encapsula a lógica de uma peça, incluindo seu estado (como cor, tamanho e se está no tabuleiro) e sua interação com o jogo e a interface.

>Board:
Ela gerencia o estado do tabuleiro do jogo, incluindo o mapeamento das coordenadas, validação de posicionamento de peças, e verificação das condições de vitória e empate. Sua lógica assegura que as regras do jogo sejam respeitadas durante as interações.

Métodos
Gerenciamento do Tabuleiro
mapBoardGridCoordinates(boardImg: ImageView): List<List<PointF>>
Mapeia as coordenadas centrais de cada célula do tabuleiro com base no layout da imagem fornecida.

determineInBoardPosition(piece: Piece): Point?
Retorna a célula mais próxima para a peça, considerando a distância mínima para posicionamento válido.

canPositionPiece(piece: Piece, gridPoint: Point): Boolean
Verifica se a peça pode ser posicionada em uma célula específica com base nas regras de cor e tamanho.

positionPieceAndGetCoordinate(piece: Piece, gridPoint: Point): PointF
Posiciona a peça na célula indicada e retorna as coordenadas centrais da célula.

Condições de Jogo
determineWinner(): PieceColor?
Verifica as condições de vitória e retorna a cor vencedora, se houver.

determineDraw(): Boolean
Determina se o jogo terminou em empate, analisando o estado do tabuleiro e as peças disponíveis.

*Métodos auxiliares
distanceBetween(p1: PointF, p2: PointF): Float
Calcula a distância entre dois pontos, usado para determinar a célula mais próxima para uma peça.

>Descrição do Enum PieceColor
Armazena as cores disponíveis para as peças do jogo (laranja e azul), permitindo identificar a qual jogador uma peça pertence. Também inclui lógica para alternar entre as cores, facilitando o gerenciamento de turnos no jogo.

>Descrição do Enum PieceSize
Representa os tamanhos das peças (pequeno, médio e grande) e associa a cada tamanho um valor numérico. Esses valores ajudam a determinar a prioridade no posicionamento no tabuleiro, já que peças maiores têm mais poder sobre peças menores.

3. Regras do Jogo
O jogo é jogado em uma grade 3x3.
Dois jogadores alternam turnos, colocando peças de cores diferentes em células vazias do tabuleiro.
As peças podem ser pequenas, médias ou grandes, e cada jogador deve posicionar suas peças no tabuleiro. É possível uma peça comer uma peça menor no tabuleiro, se posicionando naquele ponto.
O objetivo do jogo é conseguir alinhar 3 peças do mesmo jogador na horizontal, vertical ou diagonal.
Se todas as células forem preenchidas e um jogador não puder se mover e ninguém tiver vencido, o jogo termina em empate.

4. Conclusão
A implementação do jogo "Nham Nham" foi uma oportunidade de aplicar conceitos de programação orientada a objetos, interação de drag-and-drop, lógica de jogo e uso de Intents em uma aplicação Android. O projeto aplicou gerenciamento do estado do jogo, criação de interfaces gráficas interativas e uma experiência de usuário que possa interagir de maneira intuitiva.