Contents

Duel Isometry - Flux de Navigation

Document généré le 6 décembre 2025

Vue d'ensemble

Ce document trace le parcours complet depuis qu'un joueur rejoint une partie jusqu'à l'affichage du plateau avec le sélecteur de pièces.

☐ Fichiers impliqués

Fichier	Rôle
`screens/duel_isometry_join_screen.dart`	Écran de saisie code + pseudo
`screens/duel_isometry_waiting_screen.dart`	Écran d'attente du 2e joueur
`screens/duel_isometry_game_screen.dart`	Écran de jeu principal
`providers/duel_isometry_provider.dart`	Logique métier + WebSocket
`models/duel_isometry_state.dart`	État de la partie
`widgets/duel_isometry_plateau.dart`	Widget plateau de jeu
`widgets/duel_isometry_piece_slider.dart`	Widget sélecteur de pièces

☐ Phase 1 : Rejoindre une partie

DuelJoinScreen

```
DuelJoinScreen
(duel_isometry_join_screen.dart)


User: Entre code "AB12" + pseudo "Max"
```

```
_joinRoom() appelé

       if (!_formKey.validate()) return

       settingsProvider.setDuelPlayerName(name)

       duelIsometryProvider.notifier.joinRoom(code, name)


        [Voir Provider ci-dessous]
```

DuelIsometryNotifier.joinRoom()

```
DuelIsometryNotifier.joinRoom(roomCode, playerName)
(duel_isometry_provider.dart:138)


1. state = state.copyWith(connectionState: connecting)

2. HTTP GET /room/$roomCode/exists

     if (statusCode != 200)
        → state.copyWith(connectionState: error)
        → return false

     if (checkData['exists'] != true)
         → state.copyWith(errorMessage: "Code invalide...")
         → return false

3. _wsService = DuelIsometryWebSocketService(wsUrl)

4. _messageSubscription = _wsService.messages.listen(_onServerMessage)

5. _connectionSubscription = _wsService.connectionState.listen(...)

6. await _wsService.connect()

     if (!connected)
         → state.copyWith(errorMessage: "Impossible de se connecter")
         → return false

7. _wsService.send(JoinRoomMessage(roomCode, playerName))

8. state = state.copyWith(
      roomCode: roomCode,
      gameState: DuelGameState.waiting,    ←  ÉTAT: WAITING
      connectionState: connected
   )

9. return true
```

Navigation vers WaitingScreen

```
Retour dans DuelJoinScreen._joinRoom()


if (mounted) {
  Navigator.pushReplacement(
    context,
    MaterialPageRoute(
      builder: (context) => DuelIsometryWaitingScreen()
    )
  );
}
```

☐ Phase 2 : Écran d'attente

DuelIsometryWaitingScreen.build()

```
DuelIsometryWaitingScreen.build()
(duel_isometry_waiting_screen.dart)


final state = ref.watch(duelIsometryProvider)


   CONDITION DE TRANSITION VERS LE JEU

   if (state.plateau != null &&
       (state.gameState == DuelGameState.countdown ||
        state.gameState == DuelGameState.playing))

   → return DuelIsometryGameScreenContent(state)

     TRANSITION AUTOMATIQUE VERS L'ÉCRAN DE JEU


SINON: Affiche écran d'attente


   Scaffold(
     appBar: "Duel Isométries"
     body: Column([
       CircularProgressIndicator,
       "Code: ${state.roomCode}",

       if (state.opponent != null)
         "${opponent.name} a rejoint !"
```

```
        else
          "En attente du 2e joueur...",

        if (state.plateau != null)
          "Plateau généré, démarrage...",

        ElevatedButton("Annuler")
      ])
    )
```

☐ Phase 3 : Messages WebSocket du serveur

_onServerMessage() - Routage des messages

```
_onServerMessage(ServerMessage message)
(duel_isometry_provider.dart:585)


switch (message) {

  RoomJoinedMessage    → _handleRoomJoined()
  PlayerJoinedMessage  → _handlePlayerJoined()
  GameStartMessage     → _handleGameStart()      ←  GÉNÈRE PLATEAU
  CountdownMessage     → _handleCountdown()      ←  3, 2, 1, GO!
  PiecePlacedMessage   → _handlePiecePlaced()
  GameEndMessage       → _handleGameEnd()
  ErrorMessage         → _handleError()

}
```

_handleRoomJoined()

```
_handleRoomJoined(RoomJoinedMessage msg)
(ligne 541)


state = state.copyWith(
  roomCode: msg.roomCode,
  localPlayer: DuelPlayer(
    id: msg.playerId,
    name: _localPlayerName
  ),
  opponent: msg.opponentId != null
      ? DuelPlayer(id: opponentId, name: opponentName)
      : null,
  gameState: DuelGameState.waiting
```

```
)
```

_handlePlayerJoined()

```
_handlePlayerJoined(PlayerJoinedMessage msg)
(ligne 502)


if (msg.playerId != state.localPlayer?.id) {

  state = state.copyWith(
    opponent: DuelPlayer(
      id: msg.playerId,
      name: msg.playerName
    )
  )

}

→ L'UI affiche maintenant "X a rejoint !"
```

_handleGameStart() ☐ CRITIQUE

```
_handleGameStart(GameStartMessage msg)
(ligne 425)


if (msg.puzzleTriple != null) {

  try {
    final triple = msg.puzzleTriple!

    // Génération du plateau
    final plateau = _generatePlateauFromTriple(triple)

                        → [Voir Phase 4]

    //  MISE À JOUR DE L'ÉTAT AVEC LE PLATEAU
    state = state.copyWith(
      plateau: plateau,                    ←  PLATEAU CRÉÉ
      timeRemaining: msg.timeLimit,
      placedPieces: [],
      gameState: DuelGameState.countdown   ←  ÉTAT: COUNTDOWN
    )

  } catch (e) {
    print('[DUEL-ISO]  ERREUR: $e')
  }
```

```
}

→ La condition dans WaitingScreen est maintenant vraie !
→ Transition automatique vers GameScreenContent
```

_handleCountdown()

```
_handleCountdown(CountdownMessage msg)
(ligne 392)


if (msg.value == 0) {

  // GO! - Démarrage du jeu
  state = state.copyWith(
    gameState: DuelGameState.playing,  ←  ÉTAT: PLAYING
    clearCountdown: true
  )

  _startLocalTimer()  // Timer local pour le temps restant

} else {

  // 3, 2, 1...
  state = state.copyWith(countdown: msg.value)

}
```

☐ Phase 4 : Génération du plateau

_generatePlateauFromTriple()

```
_generatePlateauFromTriple(Map<String, int> triple)
(duel_isometry_provider.dart:320)


Entrée: { taille: 2, configIndex: 5, solutionNum: 42 }

1. Extraire les paramètres
   final taille = triple['taille']
   final configIndex = triple['configIndex']
   final solutionNum = triple['solutionNum']

2. Charger la configuration Pentoscope
   final configsForSize = pentoscopeData[taille]
```

```
    if (configsForSize == null || configIndex >= length)
        → return Plateau.empty(5, 5)

3. Extraire bitmask et nombre de solutions
   final (bitmask, numSolutions) = configsForSize[configIndex]

4. Décoder le bitmask en IDs de pièces
   final pieceIds = _bitmaskToIds(bitmask)
   // Ex: 0x0F3 → [1, 2, 5, 6, 7, 8]

5. Obtenir les dimensions du plateau
   final (width, height) = _getTaillePlateau(taille)

     taille 0 → (5, 3)  // 3×5 = 15 cases = 3 pièces
     taille 1 → (5, 4)  // 4×5 = 20 cases = 4 pièces
     taille 2 → (5, 5)  // 5×5 = 25 cases = 5 pièces

6. Récupérer les objets Pento
   for (final id in pieceIds) {
     selectedPieces.add(pentominos[id - 1])
   }

7. Résoudre avec PentoscopeSolver
   final solver = PentoscopeSolver(
     width: width,
     height: height,
     pieces: selectedPieces,
     maxSeconds: 5
   )
   final solution = solver.findSolution()

     if (solution == null)
         → return Plateau.empty(width, height)

8. Convertir en Plateau
   final plateau = _convertToPlateau(width, height, solution)

9. return plateau
```

☐ Phase 5 : Affichage du jeu

DuelIsometryGameScreen.build()

```
DuelIsometryGameScreen.build()
(duel_isometry_game_screen.dart)


final gameState = ref.watch(duelIsometryProvider)
final notifier = ref.read(duelIsometryProvider.notifier)
```

```
  _updatePlacementsFromState(gameState)

  final isPlaying = gameState.gameState == DuelGameState.playing
  final plateau = gameState.plateau


    if (plateau == null) {
      return Scaffold(
        body: Center(child: CircularProgressIndicator())
      )
    }


  return Scaffold(
    appBar: AppBar(title: "Duel Isométries", timer),
    body: isPlaying
        ? _buildPlayingState(gameState, plateau, notifier)
        : _buildGameState(gameState)
  )
```

_buildGameState() - États non-playing

```
  _buildGameState(DuelIsometryState state)
  (ligne 114)


  switch (state.gameState) {

    case DuelGameState.countdown:
      return _buildCountdown(state)
      // Affiche "3", "2", "1", "GO!" en grand

    case DuelGameState.ended:
      return _buildGameEnd(state)
      // Affiche gagnant et scores

    case DuelGameState.waiting:
      return _buildWaiting(state)
      // CircularProgressIndicator

    default:
      return CircularProgressIndicator()

  }
```

_buildPlayingState() ☐ ÉCRAN PRINCIPAL

```
  _buildPlayingState(gameState, plateau, notifier)
```

(ligne 128)

```
return Column([

    HEADER SCORES
    Container(color: Colors.blue.shade50)

    Row([
      Column([
        "${localPlayer.name}",
        "$_localScore"
      ]),
      Column([
        "Pièces",
        "${_myPlacedPieces.length}/12"
      ]),
      Column([
        "${opponent.name}",
        "$_opponentScore"
      ])
    ])



    PLATEAU
    Expanded(
      SingleChildScrollView(
        DuelIsometryPlateau(
          solutionPlateau: plateau,
          myPlacedPieces: _myPlacedPieces,
          opponentPlacedPieces: _opponentPlacedPieces,
          isEnabled: true,
          onCellTapped: (x, y) {
            if (_selectedPieceId != null) {
              _tryPlacePiece(notifier, plateau,
                             _selectedPieceId, x, y,
                             _selectedOrientation)
            }
          }
        )
      )
    )



    SÉLECTEUR DE PIÈCES
    Container(
      height: 200,
      color: Colors.grey.shade100,
      child: DuelIsometryPieceSlider(
        myPlacedPieces: _myPlacedPieces,
        opponentPlacedPieces: _opponentPlacedPieces,
```

```
          selectedPieceId: _selectedPieceId,
          isEnabled: true,
          onPieceSelected: (pieceId, orientation) {
            setState(() {
              _selectedPieceId = pieceId;
              _selectedOrientation = orientation;
            });
          }
        )
      )


])
```

☐ Widgets finaux

DuelIsometryPlateau

```
DuelIsometryPlateau
(duel_isometry_plateau.dart)


Props:
- solutionPlateau: Plateau (grille avec pieceIds)
- myPlacedPieces: Map<int, (x, y, orientation)>
- opponentPlacedPieces: Map<int, (x, y, orientation)>
- onCellTapped: (int x, int y) → void
- isEnabled: bool

Rendu:
- GridView.builder avec width × height cellules
- Chaque cellule:
  - MouseRegion (hover)
  - GestureDetector (tap)
  - Container avec couleur selon état

Couleurs:
- Blanc: cellule vide ou pièce placée
- Gris 15%: cellule de la solution (non placée)
- Bordure violette: hover
```

DuelIsometryPieceSlider

```
DuelIsometryPieceSlider
(duel_isometry_piece_slider.dart)
```

```
Props:
- myPlacedPieces: Map<int, (x, y, orientation)>
- opponentPlacedPieces: Map<int, (x, y, orientation)>
- selectedPieceId: int?
- onPieceSelected: (int pieceId, int orientation) → void
- isEnabled: bool

Structure:

  Row 1: Pièces 1-6

    1   2   3   4   5   6


  Row 2: Pièces 7-12

    7   8   9  10  11  12


  [Bouton TOURNER] (si pièce sélectionnée)


Chaque carte affiche:
- Numéro de pièce (1-12)
- Orientation actuelle (O0, O1, O2...)
- Nombre d'isométries (±N)
- Icônes si placée (  bleu = moi,   rouge = adversaire)

Couleurs de fond:
- Gris clair: disponible
- Bleu 30%: placée par moi
- Rouge 30%: placée par adversaire
- Mix: placée par les deux

Bordure:
- Violette épaisse: sélectionnée
- Bleue: placée par moi
- Rouge: placée par adversaire
- Grise: disponible
```

☐ Tableau récapitulatif des états

Phase	`gameState`	`plateau`	`opponent`	Écran affiché
1. Join	`waiting`	`null`	`null`	WaitingScreen (attente)
2. Adversaire rejoint	`waiting`	`null`		WaitingScreen (“X a rejoint”)

Phase	gameState	plateau	opponent	Écran affiché
3. GameStart reçu	countdown			GameScreen → _buildCountdown
4. Countdown 3→2→1	countdown			“3”, “2”, “1”
5. Countdown = 0	playing			GameScreen → _buildPlayingState
6. Fin de partie	ended			GameScreen → _buildGameEnd

☐ Diagramme de séquence simplifié

```
User          JoinScreen       Provider        WebSocket       Server

    [code+name]
                  joinRoom()
                                  HTTP GET
                                        {exists: true}
                                  connect()
                                         connected
                                  JoinRoom
                  true

           Navigator.push(WaitingScreen)

                                   RoomJoined
                                   PlayerJoined
                                   GameStart
                                    (puzzleTriple)

                                  generatePlateau()
                                   state.plateau = plateau
                                   state.gameState = countdown

                     (rebuild avec plateau)
     [GameScreen avec plateau + sélecteur]

                                   Countdown(3)
                                   Countdown(2)
                                   Countdown(1)
                                   Countdown(0)
                                    state.gameState = playing

                     [JEU EN COURS]
```

☐ Notes importantes

1. Transition automatique : Le passage de WaitingScreen à GameScreen se fait automatiquement grâce au `ref.watch()` qui reconstruit l'UI quand `state.plateau` devient non-null ET `gameState` passe à countdown/playing.
2. Génération locale : Le plateau est généré localement par chaque client à partir du `puzzleTriple` envoyé par le serveur. Cela garantit que les deux joueurs ont exactement le même plateau.
3. État local vs serveur :
   - `_myPlacedPieces` et `_opponentPlacedPieces` sont gérés localement dans le widget
   - `state.placedPieces` vient du serveur via WebSocket
4. Sélecteur de pièces : Le widget `DuelIsometryPieceSlider` gère sa propre map d'orientations (`pieceOrientations`) pour permettre la rotation avant placement.