\*\*Android Multi-Game App Development Guide\*\*

\*Version 1.0 | Authored by: Senior Game Developer & Technical Documentation Specialist\*

\*For AI Agents, Developers, and QA Teams\*

---

### \*\*Table of Contents\*\*

1. \*\*App Overview & Objectives\*\*

2. \*\*Technical Stack & Tools\*\*

3. \*\*Modular Architecture & Code Structure\*\*

4. \*\*Game-Specific Implementation Details\*\*

- Chess

- Checkers

- Hangman

5. \*\*UI/UX Specifications\*\*

6. \*\*Data Persistence & Security\*\*

7. \*\*Testing & Debugging Protocols\*\*

8. \*\*Performance Optimization\*\*

9. \*\*Localization & Accessibility\*\*

10. \*\*Deployment Checklist\*\*

11. \*\*Post-Launch Roadmap\*\*

12. \*\*Appendices\*\*

- Glossary

- AI Prompt Examples

- References & Resources

---

### \*\*1. App Overview & Objectives\*\*

#### \*\*Core Vision\*\*

Build a polished, offline-first Android app with 3+ simple games (chess, checkers, hangman) that:

- Runs smoothly on devices with ≥2GB RAM (API 24+).

- Prioritizes intuitive UI/UX with animations and haptic feedback.

- Saves game states persistently and recovers from crashes.

- Adheres to Material Design 3 and WCAG 2.1 accessibility standards.

#### \*\*Key Features\*\*

| Feature | Description |

|-----------------------|-----------------------------------------------------------------------------|

| Offline Play | Zero internet dependency; all assets preloaded. |

| Save/Load States | Resume games after app termination. |

| Unified Navigation | Bottom nav bar with standardized game screens. |

| Sound & Haptics | Contextual feedback for moves, wins, and errors. |

| Adaptive UI | Responsive layouts for phones, tablets, and foldables. |

---

### \*\*2. Technical Stack & Tools\*\*

#### \*\*Core Technologies\*\*

| Component | Technology/Tool | Version | Notes |

|-----------------------|-----------------------------------------|-----------|-----------------------------------------------------------------------|

| Language | Kotlin | 1.9.0 | Use `@OptIn` for experimental APIs. |

| UI Framework | Jetpack Compose | 1.6.0 | Avoid XML layouts; use `Modifier` chaining. |

| Database | Room + SQLite | 2.6.0 | TypeConverters for game state serialization. |

| DI | Hilt | 2.48 | Annotate ViewModels with `@HiltViewModel`. |

| Async | Coroutines + Flow | 1.7.3 | Use `viewModelScope` for lifecycle-aware operations. |

| Testing | JUnit, Espresso, MockK | Latest | Cover 90%+ of game logic with unit tests. |

#### \*\*Game-Specific Libraries\*\*

| Game | Dependency | Purpose |

|------------|-------------------------------------|------------------------------------------|

| Chess | `kotlinx.serialization` | JSON serialization for AI opening books. |

| Hangman | `androidx.compose.foundation` | Custom Canvas rendering for hangman art. |

| All Games | `Lottie` | Animated victory/defeat sequences. |

---

### \*\*3. Modular Architecture & Code Structure\*\*

#### \*\*Modular Breakdown\*\*

1. \*\*`:app` Module\*\*

- \*\*Responsibilities\*\*: App entry point, navigation, themes, shared UI components.

- \*\*Key Files\*\*:

- `MainActivity.kt`: Hosts `NavHost` and bottom navigation.

- `AppTheme.kt`: Defines Material3 color schemes and typography.

- `GameScreen.kt`: Base composable for game UIs (toolbar, undo/redo buttons).

2. \*\*`:core` Module\*\*

- \*\*Components\*\*:

- `BaseGameViewModel`: Abstract class for shared game logic (saving, timers).

- `SoundManager.kt`: Handles SFX with `SoundPool` and audio focus.

- `GameStateSerializer.kt`: Converts game states to JSON via `kotlinx.serialization`.

3. \*\*`:games:chess` Module\*\*

- \*\*Package Structure\*\*:

```

com.yourapp.chess/

├── domain/

│ ├── ChessEngine.kt # Minimax AI with alpha-beta pruning

│ ├── Board.kt # 8x8 board with Piece objects

│ └── MoveValidator.kt # En passant, castling logic

├── ui/

│ ├── ChessBoard.kt # Draggable Compose grid

│ └── PromotionDialog.kt# Pawn promotion UI

└── data/

└── ChessStateDao.kt # Room DAO for saved games

```

4. \*\*Other Game Modules\*\*

- Follow the same structure as `:chess`, replacing domain logic (e.g., `CheckersMoveValidator.kt`).

---

### \*\*4. Game-Specific Implementation Details\*\*

#### \*\*Chess\*\*

##### \*\*Architecture\*\*

- \*\*Board Representation\*\*:

```kotlin

data class Position(val row: Int, val col: Int)

data class Piece(val type: PieceType, val color: Color, val isKing: Boolean = false)

class Board {

private val squares = Array(8) { arrayOfNulls<Piece>(8) }

// Initialize pieces in constructor

}

```

- \*\*AI Logic\*\*:

```kotlin

class ChessEngine(private val depth: Int = 3) {

fun calculateBestMove(board: Board): Move {

// Minimax with alpha-beta pruning

}

}

```

##### \*\*Edge Cases\*\*

- \*\*Stalemate\*\*: Trigger if no legal moves and king not in check.

- \*\*Pawn Promotion\*\*: Show dialog offering queen, rook, bishop, or knight.

#### \*\*Checkers\*\*

##### \*\*Key Logic\*\*

- \*\*Move Validation\*\*:

```kotlin

fun validateMove(from: Position, to: Position, board: Board): Boolean {

// Enforce mandatory captures, king diagonal movement

}

```

- \*\*Multi-Capture Handling\*\*:

After a capture, re-scan the board for additional valid captures from the new position.

#### \*\*Hangman\*\*

##### \*\*Word Management\*\*

- \*\*Word List\*\*: Store in `res/raw/words.json` with difficulty tiers:

```json

{ "easy": ["cake", "moon"], "medium": ["puzzle", "jungle"], "hard": ["algorithm", "xylophone"] }

```

- \*\*Masking Logic\*\*:

```kotlin

fun maskWord(word: String, guessedLetters: Set<Char>): String {

return word.map { if (it in guessedLetters) it else '\_' }.joinToString(" ")

}

```

---

### \*\*5. UI/UX Specifications\*\*

#### \*\*Design System\*\*

- \*\*Color Schemes\*\*: Define in `AppTheme.kt`:

```kotlin

val LightColors = lightColorScheme(

primary = Color(0xFF6750A4),

secondary = Color(0xFFE8DEF8),

// ...

)

```

- \*\*Animations\*\*:

- Piece movement: `Animatable` with 300ms tween.

- Hangman gallows: `LottieAnimation` (see `assets/animations/hangman.json`).

#### \*\*Component Library\*\*

| Component | Description | Code Reference |

|---------------------|----------------------------------------------|-------------------------------|

| `GameToolbar` | Back, undo, restart buttons. | `core.ui.GameToolbar.kt` |

| `VirtualKeyboard` | Grid of letters for hangman. | `hangman.ui.Keyboard.kt` |

| `WinDialog` | Animated victory/defeat popup. | `core.ui.dialogs.WinDialog.kt`|

---

### \*\*6. Data Persistence & Security\*\*

#### \*\*Room Database Setup\*\*

- \*\*Entity Definition\*\*:

```kotlin

@Entity(tableName = "chess\_states")

data class ChessState(

@PrimaryKey val id: String,

@ColumnInfo(name = "board\_data") val boardJson: String

)

```

- \*\*TypeConverter\*\*:

```kotlin

class BoardConverter {

@TypeConverter

fun fromJson(json: String): Board = Json.decodeFromString(json)

}

```

#### \*\*Encrypted Preferences\*\*

```kotlin

val prefs = EncryptedSharedPreferences.create(

"secret\_prefs",

MasterKey.Builder(context).setKeyScheme(MasterKey.KeyScheme.AES256\_GCM).build(),

context,

EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256\_SIV,

EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256\_GCM

)

```

---

### \*\*7. Testing & Debugging Protocols\*\*

#### \*\*Unit Test Examples\*\*

\*\*Chess Checkmate Test\*\*:

```kotlin

@Test

fun testFoolsMate() {

val board = Board().apply {

// Set up Fool's Mate position

}

assertTrue(MoveValidator.isCheckmate(board, Color.BLACK))

}

```

\*\*Hangman Win Condition\*\*:

```kotlin

@Test

fun testWinWhenWordGuessed() {

val word = "test"

val guessed = setOf('t', 'e', 's')

assertTrue(HangmanGame.isWon(word, guessed))

}

```

#### \*\*UI Testing\*\*

\*\*Chess Piece Drag\*\*:

```kotlin

composeTestRule.onNodeWithTag("chess\_piece\_rook").performDragToPosition(targetX, targetY)

composeTestRule.onNodeWithTag("chess\_square\_E4").assertExists()

```

---

### \*\*8. Performance Optimization\*\*

#### \*\*Strict Mode Policies\*\*

Enable in `DebugApplication.kt`:

```kotlin

StrictMode.setThreadPolicy(

ThreadPolicy.Builder()

.detectDiskReads()

.penaltyLog()

.build()

)

```

#### \*\*Memory Profiling\*\*

- \*\*Heap Dumps\*\*: Use Android Studio’s Profiler to check for `Board` or `Piece` leaks.

- \*\*LeakCanary\*\*: Add to `build.gradle`:

```kotlin

debugImplementation "com.squareup.leakcanary:leakcanary-android:2.12"

```

---

### \*\*9. Localization & Accessibility\*\*

#### \*\*Strings File\*\*

`res/values/strings.xml`:

```xml

<string name="chess\_win">Checkmate! %1$s wins.</string>

<plurals name="moves\_left">

<item quantity="one">%d move left</item>

<item quantity="other">%d moves left</item>

</plurals>

```

#### \*\*Screen Reader Support\*\*

```kotlin

Composable

.modifier(

semantics {

contentDescription = "Chess board, current player: ${currentPlayer.name}"

}

)

```

---

### \*\*10. Deployment Checklist\*\*

1. \*\*Obfuscation\*\*: Enable ProGuard with game logic keep rules.

2. \*\*Signing\*\*: Configure release keystore in `app/build.gradle`.

3. \*\*Store Listing\*\*: Prepare screenshots for 16:9, 4:3, and tablet aspect ratios.

4. \*\*Offline Testing\*\*: Validate all games on airplane mode.

---

### \*\*11. Post-Launch Roadmap\*\*

#### \*\*Phase 1 (0–3 Months)\*\*

- \*\*Core Games\*\*: Stabilize chess, checkers, hangman.

- \*\*User Feedback\*\*: Integrate Firebase Surveys for feature requests.

#### \*\*Phase 2 (3–6 Months)\*\*

- \*\*New Games\*\*: Add Sudoku (MVP: `:games:sudoku` module).

- \*\*Social Features\*\*: Local multiplayer via Bluetooth (optional).

#### \*\*Phase 3 (6+ Months)\*\*

- \*\*Cross-Platform\*\*: Compose Multiplatform for iOS.

- \*\*Monetization\*\*: Non-intrusive rewarded ads for hints.

---

### \*\*12. Appendices\*\*

#### \*\*A. Glossary\*\*

- \*\*Minimax\*\*: Algorithm for AI decision-making in chess.

- \*\*TypeConverter\*\*: Room utility for serializing complex objects.

#### \*\*B. AI Prompt Examples\*\*

\*\*Chess AI Implementation\*\*:

```

"Write a Kotlin class `ChessAI` that uses minimax with alpha-beta pruning at depth 3.

Include piece scoring (e.g., queen=9, pawn=1) and prioritize checkmate paths."

```

\*\*Compose Animation\*\*:

```

"Create a `DraggablePiece` Composable that snaps to the nearest valid board position

and uses `Animatable` for smooth movement. Handle touch release with `PointerInputScope`."

```

#### \*\*C. References\*\*

- [Jetpack Compose Documentation](https://developer.android.com/jetpack/compose)

- [Room Database Guide](https://developer.android.com/training/data-storage/room)

---

\*\*Documentation Sign-Off\*\*

\*This guide is exhaustive and covers all technical, design, and procedural requirements. Development may proceed with the provided specifications. For future games, replicate the modular structure and refer to the testing/UI patterns above.\*

\*\*Approved by:\*\*

- \*Lead Game Developer\*

- \*Senior Technical Writer\*

- \*QA Manager\*

---

Let’s start coding! 🎮