Classic TDD meets functional programming with Kotlin

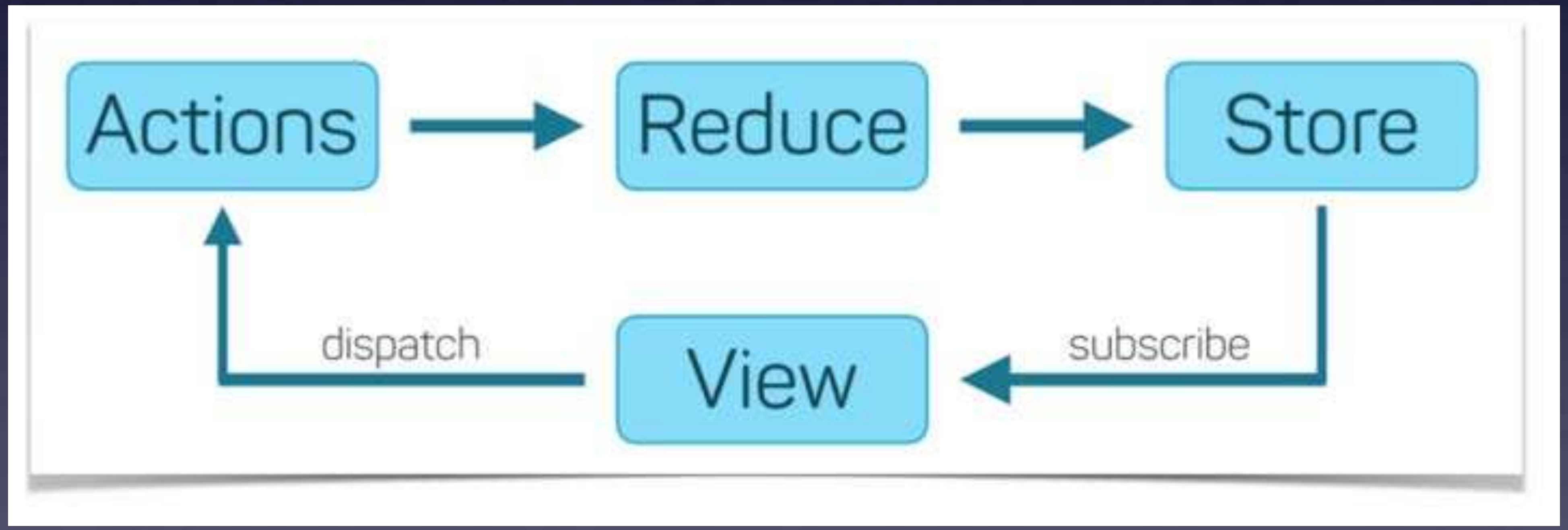
August 17, 2020
Peti Koch

TDD Styles

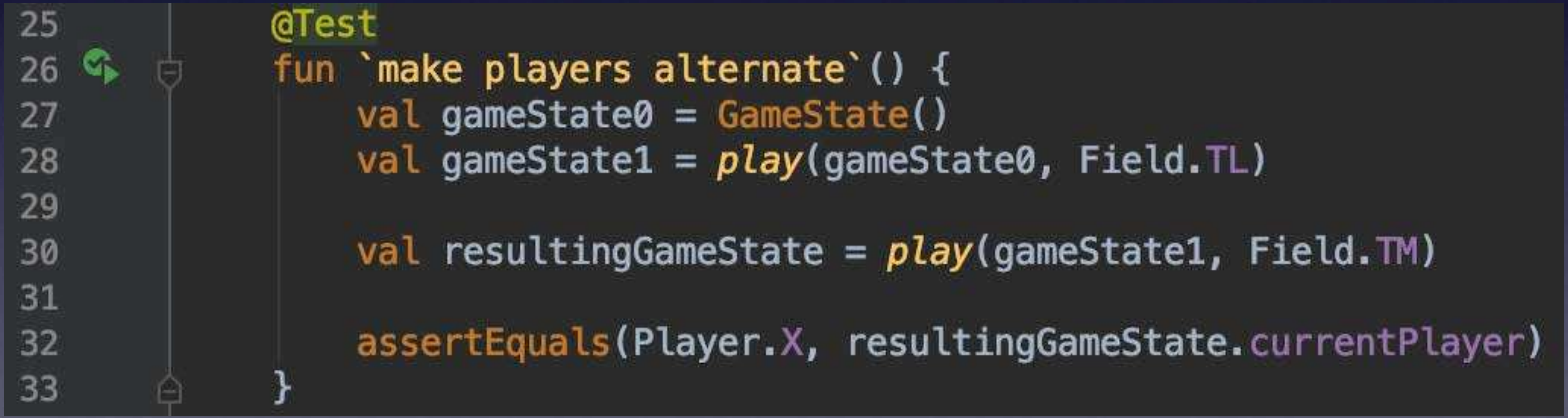
	Classic TDD	Outside-In TDD
Inventors / Origin	Various (e.g. NASA, early 60s)	…
Made „famous“ by	Kent Beck (XP, late 90s)	…
Fits well for	Algorithms	…
…	…	…

Classic TDD for functional programming?

- Let’s try to implement TicTacToe again, with a functional programming style
- Let’s use Kotlin for the Job
- And: I want to actually play it!
- Let’s do it with the unidirectional data flow pattern in mind (Flux, Redux, …)

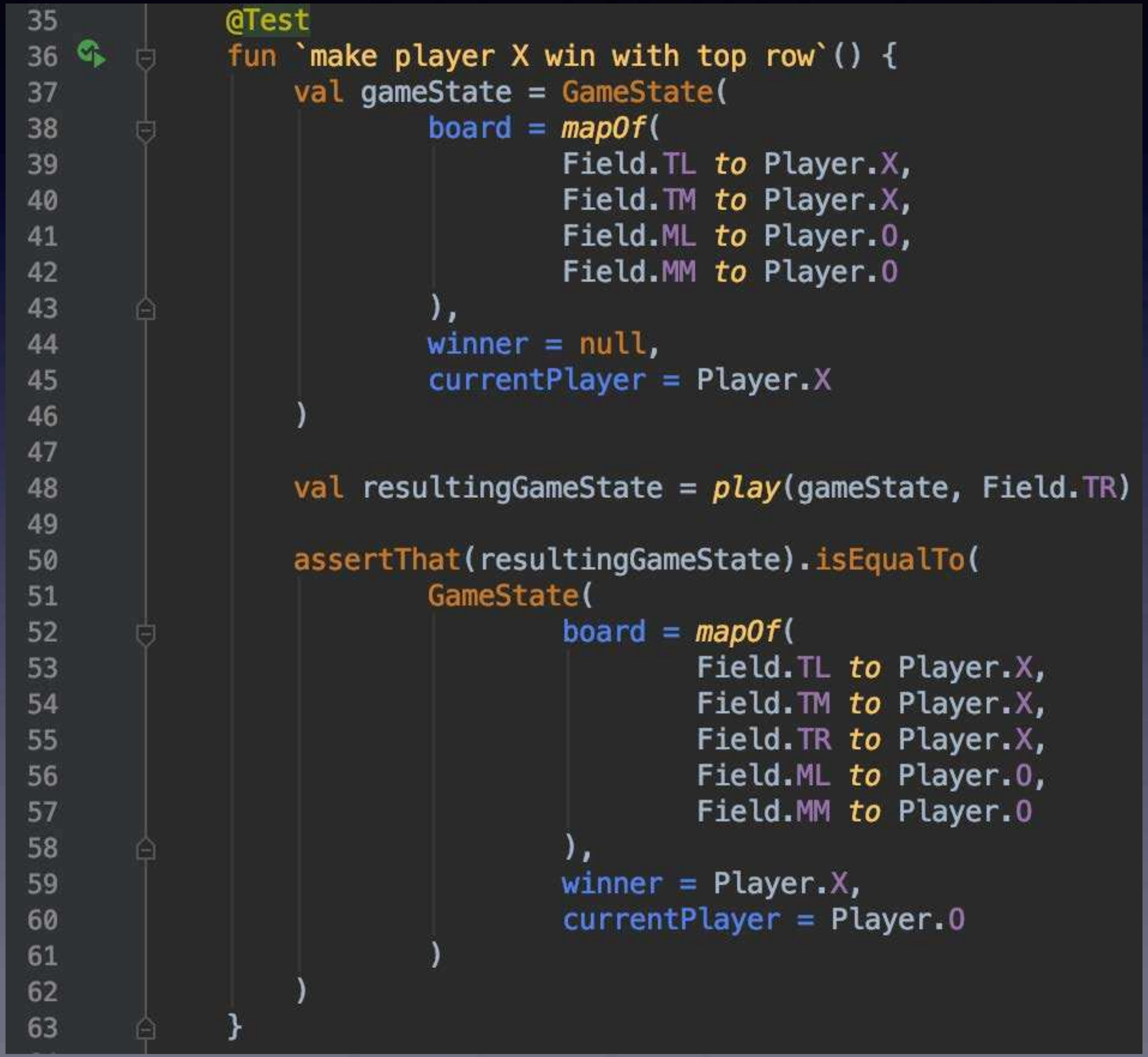


Test structure (1)



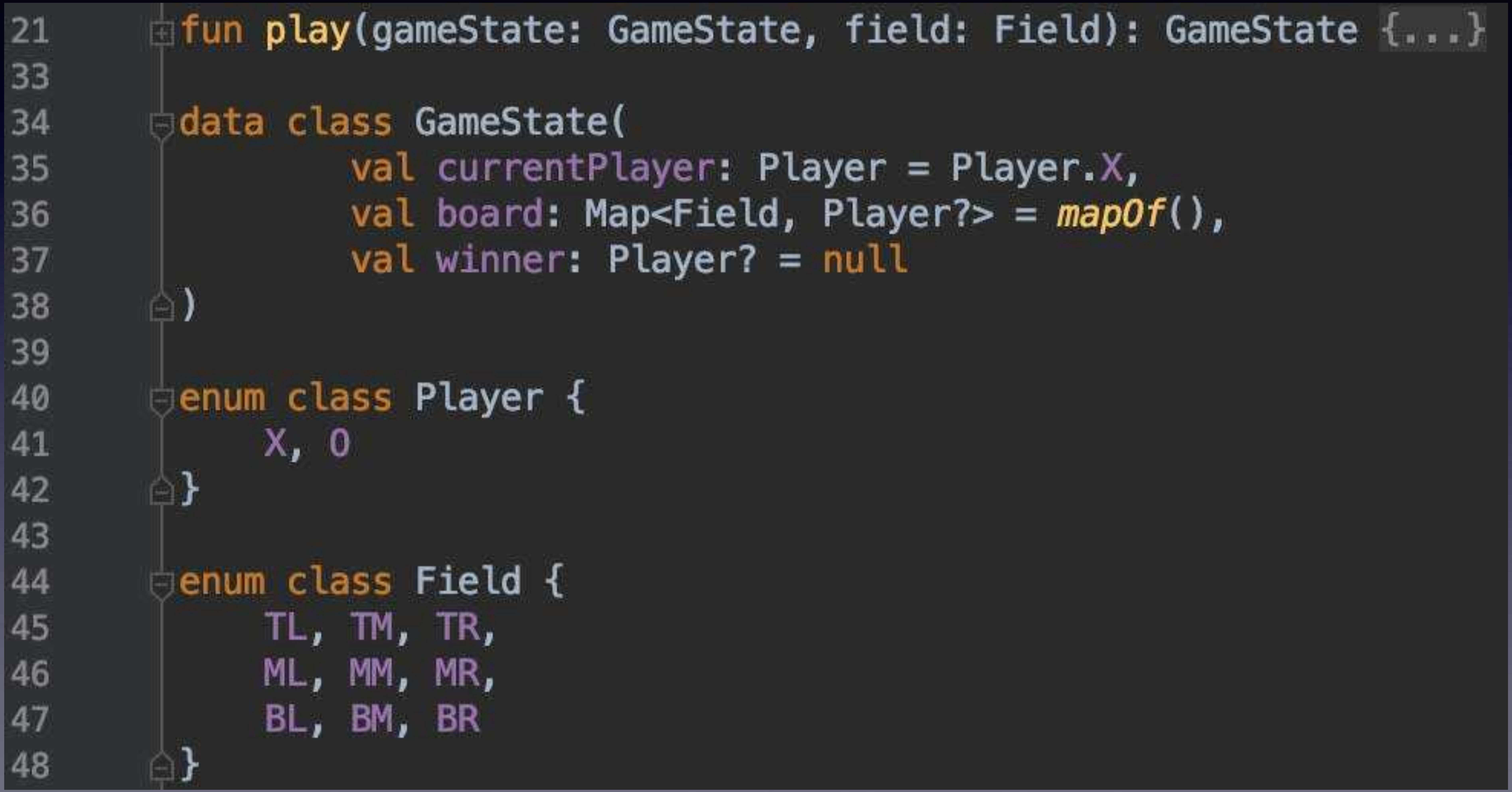
```
@Test
fun `make players alternate`() {
    val gameState0 = GameState()
    val gameState1 = play(gameState0, Field.TL)

    val resultingGameState = play(gameState1, Field.TM)

    assertEquals(Player.X, resultingGameState.currentPlayer)
}
```

Test structure (2)

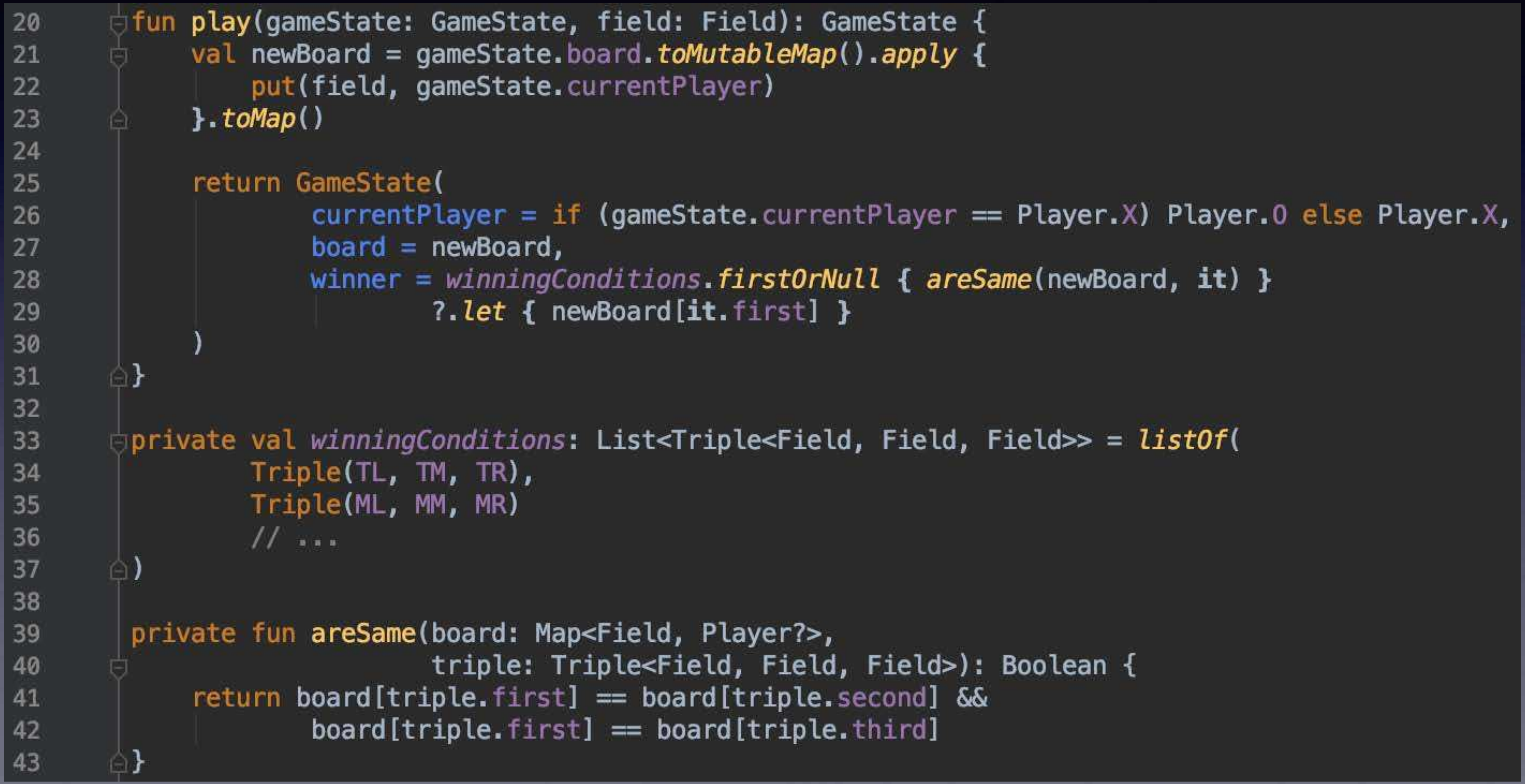


```kotlin
    @Test
    fun `make player X win with top row`() {
        val gameState = GameState(
                board = mapOf(
                        Field.TL to Player.X,
                        Field.TM to Player.X,
                        Field.ML to Player.O,
                        Field.MM to Player.O
                ),
                winner = null,
                currentPlayer = Player.X
        )

        val resultingGameState = play(gameState, Field.TR)

        assertThat(resultingGameState).isEqualTo(
                GameState(
                        board = mapOf(
                                Field.TL to Player.X,
                                Field.TM to Player.X,
                                Field.TR to Player.X,
                                Field.ML to Player.O,
                                Field.MM to Player.O
                        ),
                        winner = Player.X,
                        currentPlayer = Player.O
                )
        )
    }
```

Function signature

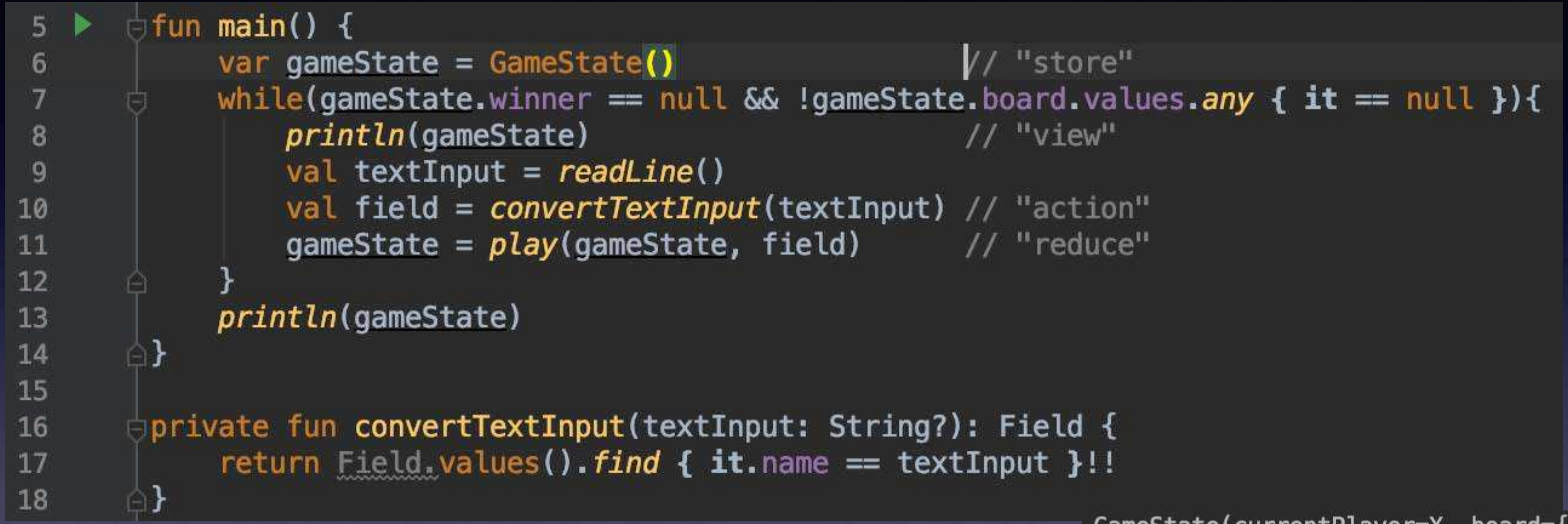

```kotlin
fun play(gameState: GameState, field: Field): GameState {...}

data class GameState(
        val currentPlayer: Player = Player.X,
        val board: Map<Field, Player?> = mapOf(),
        val winner: Player? = null
)

enum class Player {
    X, O
}

enum class Field {
    TL, TM, TR,
    ML, MM, MR,
    BL, BM, BR
}
```

Function implementation

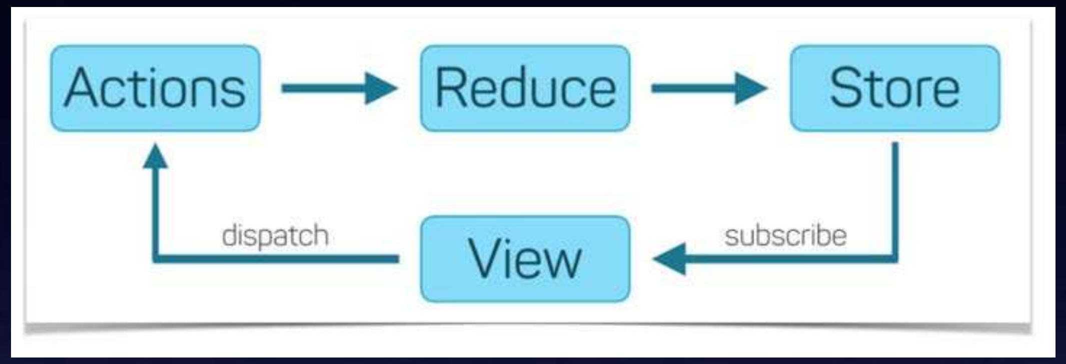


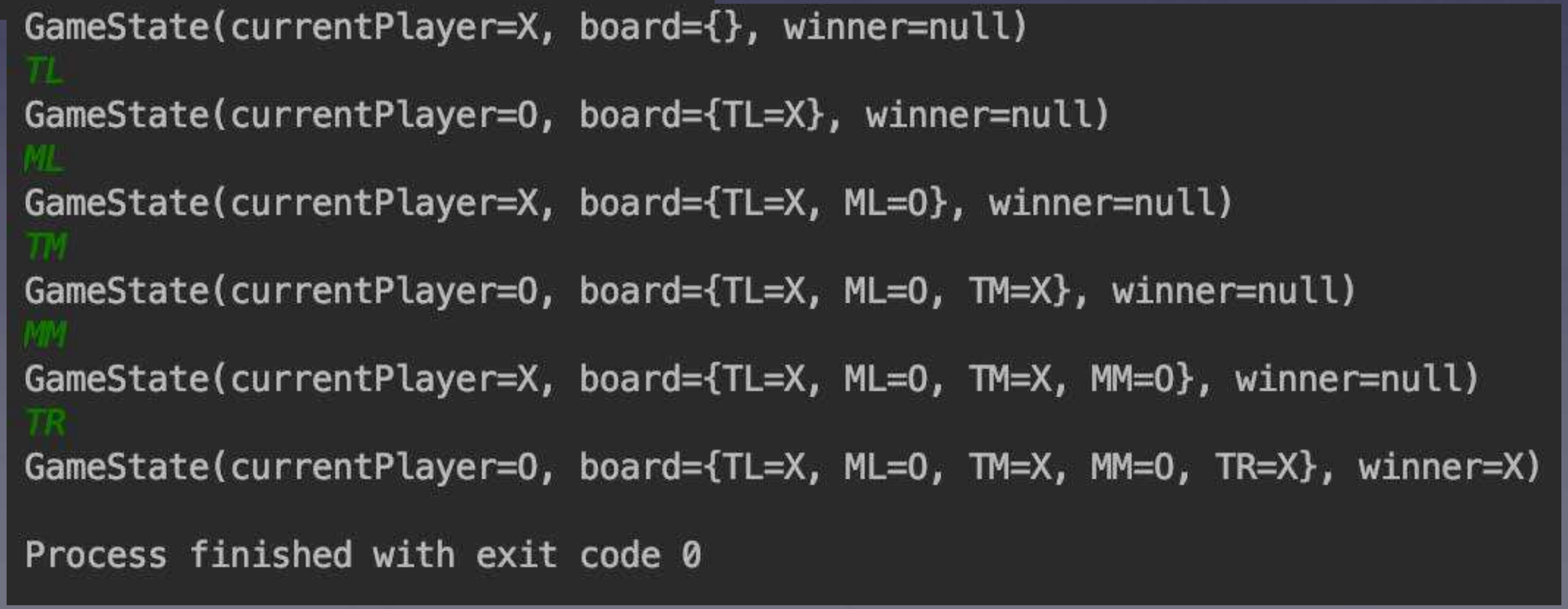
```kotlin
fun play(gameState: GameState, field: Field): GameState {
    val newBoard = gameState.board.toMutableMap().apply {
        put(field, gameState.currentPlayer)
    }.toMap()

    return GameState(
            currentPlayer = if (gameState.currentPlayer == Player.X) Player.O else Player.X,
            board = newBoard,
            winner = winningConditions.firstOrNull { areSame(newBoard, it) }
                    ?.let { newBoard[it.first] }
    )
}

private val winningConditions: List<Triple<Field, Field, Field>> = listOf(
        Triple(TL, TM, TR),
        Triple(ML, MM, MR)
        // ...
)

private fun areSame(board: Map<Field, Player?>,
                    triple: Triple<Field, Field, Field>): Boolean {
    return board[triple.first] == board[triple.second] &&
            board[triple.first] == board[triple.third]
}
```

Game loop



```kotlin
fun main() {
    var gameState = GameState()                 // "store"
    while(gameState.winner == null && !gameState.board.values.any { it == null }){
        println(gameState)                      // "view"
        val textInput = readLine()
        val field = convertTextInput(textInput) // "action"
        gameState = play(gameState, field)      // "reduce"
    }
    println(gameState)
}

private fun convertTextInput(textInput: String?): Field {
    return Field.values().find { it.name == textInput }!!
}
```





```
GameState(currentPlayer=X, board={}, winner=null)
TL
GameState(currentPlayer=O, board={TL=X}, winner=null)
ML
GameState(currentPlayer=X, board={TL=X, ML=O}, winner=null)
TM
GameState(currentPlayer=O, board={TL=X, ML=O, TM=X}, winner=null)
MM
GameState(currentPlayer=X, board={TL=X, ML=O, TM=X, MM=O}, winner=null)
TR
GameState(currentPlayer=O, board={TL=X, ML=O, TM=X, MM=O, TR=X}, winner=X)

Process finished with exit code 0
```