//

// TicTacToeModelTests.swift

// TicTacToeTests

//

// Created by Satyamboss1 on 17-02-2023.

// Copyright © 2023 Satyamboss1. All rights reserved.

//

Import Combine

Import XCTest

@testable import TicTacToe

Class TicTacToeModelTests: XCTestCase {

Private var dependecy: Dependency!

Override func setUp() {

Self.dependecy = Dependency()

}

Func test\_initial\_states() {

XCTAssertNil(dependecy.store.winner)

XCTAssertEqual(dependecy.store.currentPlayer, .o)

XCTAssertFalse(dependecy.store.isGameEnded)

Let lines = dependecy.extra.lines

Let expectedBoard = (0..<lines).map { section in

(0..<lines).map { Address(id: $0 + section \* lines, player: nil) }

}

XCTAssertEqual(dependecy.store.board, expectedBoard)

}

Func test\_currentPlayer\_changes\_when\_empty\_address\_tapped() {

Let address = dependecy.store.board[0][0]

XCTAssertNil(address.player)

XCTAssertEqual(dependecy.store.currentPlayer, .o)

Dependecy.testTarget.input.handleAddress.send(address)

XCTAssertEqual(dependecy.store.board[0][0].player, .o)

XCTAssertEqual(dependecy.store.currentPlayer, .x)

}

Func test\_currentPlayer\_does\_not\_change\_when\_non\_empty\_address\_tapped() {

Var board = dependecy.store.board

Let address = Address(id: 0, player: .o)

Board[0][0] = address

Dependecy.store.board = board

XCTAssertEqual(dependecy.store.currentPlayer, .o)

Dependecy.testTarget.input.handleAddress.send(address)

XCTAssertEqual(dependecy.store.board, board)

XCTAssertEqual(dependecy.store.currentPlayer, .o)

}

Func test\_winner\_is\_o() {

Dependecy.extra.winPatterns.forEach { pattern in

Dependecy.testTarget.input.startNewGame.send()

XCTAssertNil(dependecy.store.winner)

XCTAssertFalse(dependecy.store.isGameEnded)

Var board = dependecy.store.board

Dependecy.idAndIndexPaths(from: pattern).forEach {

Board[$1.section][$1.row] = Address(id: $0, player: .o)

}

Dependecy.store.board = board

XCTAssertEqual(dependecy.store.winner, .o)

XCTAssertTrue(dependecy.store.isGameEnded)

}

}

Func test\_winner\_is\_x() {

Dependecy.extra.winPatterns.forEach { pattern in

Dependecy.testTarget.input.startNewGame.send()

XCTAssertNil(dependecy.store.winner)

XCTAssertFalse(dependecy.store.isGameEnded)

Var board = dependecy.store.board

Dependecy.idAndIndexPaths(from: pattern).forEach {

Board[$1.section][$1.row] = Address(id: $0, player: .x)

}

Dependecy.store.board = board

XCTAssertEqual(dependecy.store.winner, .x)

XCTAssertTrue(dependecy.store.isGameEnded)

}

}

Func test\_draw() {

XCTAssertNil(dependecy.store.winner)

XCTAssertFalse(dependecy.store.isGameEnded)

Var board = dependecy.store.board

Board[0][0] = Address(id: 0, player: .o)

Board[0][1] = Address(id: 1, player: .o)

Board[0][2] = Address(id: 2, player: .x)

Board[1][0] = Address(id: 3, player: .x)

Board[1][1] = Address(id: 4, player: .x)

Board[1][2] = Address(id: 5, player: .o)

Board[2][0] = Address(id: 6, player: .o)

Board[2][1] = Address(id: 7, player: .o)

Board[2][2] = Address(id: 8, player: .x)

Dependecy.store.board = board

XCTAssertNil(dependecy.store.winner)

XCTAssertTrue(dependecy.store.isGameEnded)

}

}

Extension TicTacToeModelTests {

Private typealias Address = TicTacToeModel.Address

Private struct Dependency {

Let store = TicTacToeModel.Store()

Let extra = TicTacToeModel.Extra()

Let testTarget: TicTacToeModel

Init() {

Self.testTarget = TicTacToeModel(input: .init(), store: store, extra: extra)

}

Func idAndIndexPaths(from array: [Int]) -> [(Int, IndexPath)] {

Let lines = extra.lines

Return array.map {

Let section = $0 / lines

Return ($0, IndexPath(row: $0 – (section \* lines), section: section))

}

}

}

}