```go
package main

//Constant for no win board state
var NO_WINNER int = 0

//
var iWin = [8][3]int{
    {0, 1, 2},
    {3, 4, 5},
    {6, 7, 8},
    {0, 3, 6},
    {1, 4, 7},
    {2, 5, 8},
    {2, 4, 6},
    {0, 4, 8}}

type SubBoard [9]int

type UltimateBoard [9]SubBoard

func (board *UltimateBoard) Copy() *UltimateBoard {
    originalBoard := *board
    boardCopy := originalBoard
    return &boardCopy
}

type Move struct {
    board  int
    square int
}

func (move *Move) Copy() *Move {
    return &Move{move.board, move.square}
}

/**
 * Returns the integer value of the winner, if it exists
 * Returns 0 if NO Winner
 */
func (board *SubBoard) GetWinner() int {
    //Indeces to check if same
    //0 1 2
    //3 4 5
```

```
    //6 7 8
    //
    //0 3 6
    //1 4 7
    //2 5 8
    //
    //2 4 6
    //0 4 8

    //Iterate over all the possible combinations of winning indices
    for i, _ := range iWin{
        square1 := board[iWin[i][0]]
        square2 := board[iWin[i][1]]
        square3 := board[iWin[i][2]]

        if square1 != 0 &&              //Value must be played on
            square1 == square2 &&      //Check if 1 == 2
            square2 == square3{        //Check if 2 == 3
            return square1            //Return the player that won
        }
    }
    return NO_WINNER                                        //No Winner
}

func (board *SubBoard) Clear() {

}

/**
 * Return the integer value of winner of full game
 * 0 if No Winner
 */
func (board *UltimateBoard) GetWinner() int {
    for i, _ := range iWin{
        subboard1 := board[iWin[i][0]]
        subboard2 := board[iWin[i][1]]
        subboard3 := board[iWin[i][2]]

        square1 := subboard1.GetWinner()
        square2 := subboard2.GetWinner()
        square3 := subboard3.GetWinner()

        if square1 != 0 &&
            square1 == square2 &&
            square2 == square3{
            return square1
```

```
            return square
        }
    }

    return NO_WINNER
}

func (board *UltimateBoard) Clear() {

}

func (board *SubBoard) GetValidMoves(boardIndex int) []*Move {
    moves := make([]*Move, 0, 9)
    if board.GetWinner() == 0 {
        for index, square := range board {
            // append available moves to moves array
            if square == 0 {
                newMove := &Move{boardIndex, index}
                moves = append(moves, newMove)
            }
        }
    }
    return moves
}

/**
 * Get Valid Moves
 */
func (uboard *UltimateBoard) GetValidMoves(lastMove *Move) []*Move {
    moves := make([]*Move, 0)
    if uboard.GetWinner() == 0 {
        nextSubBoardIndex := lastMove.square
        nextSubBoard := uboard[nextSubBoardIndex]
        // next SubBoard is available
        if nextSubBoard.GetWinner() == 0 {
            // return moves in next subboard
            moves = append(moves, nextSubBoard.GetValidMoves(nextSubBoardIndex)..
        } else {
            // return moves in all available subboards
            for index, board := range uboard {
                if board.GetWinner() == 0 {
                    moves = append(moves, board.GetValidMoves(index)...)
                }
            }
        }
    }
```

```
    return moves
}

/**
 * Apply a move to the ultimate board of Player index
 */
func (board *UltimateBoard) ApplyMove(move *Move, playerId int) {
    board[move.board][move.square] = playerId
}
```