```go
package main

import (
	"math"
	"math/rand"
	"time"
)

//var TIME_PER_MOVE float64 = 623.0 //Time Per Move in Milliseconds
var TIME_PER_MOVE float64 = 790.0 //Time Per Move in Milliseconds

func RunMonteCarlo(validBoards []int, board *UltimateBoard) *Move {
	startTime := time.Now()
	var movesToTry []*Move

	for _, validBoard := range validBoards {
		boardMoves := board[validBoard].GetValidMoves(validBoard)
		movesToTry = append(movesToTry, boardMoves...)
	}

	ties := make(map[Move]float64)
	wins := make(map[Move]float64)
	weightedWins := make(map[Move]float64)
	losses := make(map[Move]float64)
	weightedLosses := make(map[Move]float64)
	gamesPlayed := 0

	allValidMoves := make([]Move, len(movesToTry))
	for i := 0; i < len(movesToTry); i++ {
		allValidMoves[i] = *movesToTry[i]
	}

	for {

		for _, move := range movesToTry {
			gamesPlayed += 1

			previousPlayer := PLAYER

			var movesToGameOver float64

			simulatedBoard := board.Copy()
			simulatedMove := move.Copy()
```

```go

            simulatedBoard.ApplyMove(simulatedMove, PLAYER)

            movesToGameOver += 1.0

            for simulatedBoard.GetWinner() == NO_WINNER {
                if len(simulatedBoard.GetValidMoves(simulatedMove)) == 0 {
                    ties[*move] += 1.0
                    break
                }

                movesToGameOver += 1.0

                simulatedMove = MoveRandomizer(simulatedMove, simulatedBoard)
                previousPlayer = 3 - previousPlayer
                simulatedBoard.ApplyMove(simulatedMove, previousPlayer)
            }
            simulatedWinner := simulatedBoard.GetWinner()
            if simulatedWinner == PLAYER {
                wins[*move] += 1.0
                weightedWins[*move] += (1.0 / movesToGameOver)
            } else if simulatedWinner == OPPONENT {
                losses[*move] += 1.0
                weightedLosses[*move] += (1.0 / movesToGameOver)
            }

        }

        end := time.Now()
        if end.Sub(startTime).Seconds()*1000.0 > TIME_PER_MOVE {
            break
        }
    }

    bestScore := math.Inf(-1)
    var bestMove Move

    for _, move := range allValidMoves {
        score := (weightedWins[move] - (weightedLosses[move] * 2.0)) / (wins[move
        if score > bestScore {
            bestScore = score
            bestMove = move
        }
    }

    return &bestMove
    return &bestMove
}

func MoveRandomizer(lastMove *Move, board *UltimateBoard) *Move {

    validMoves := board.GetValidMoves(lastMove)
    randomMoveIndex := rand.Intn(len(validMoves))
    return validMoves[randomMoveIndex]
}
```