DomsPlayer = Hand -> Board -> (Domino,End)

Take the hand and the board. And find the correct domino with each player’s characteristics which can be played on the board on L or R.

Data Structure

Board = [Domino]

Hand = [Domino]

Domino = (Int, Int)

End = L | R deriving (Eq, Show)

Ordered(list of all the dominoes) ::[Domino]

DomsPlayer = Hand -> Board -> (Domino,End)

hsdPlayer :: DomsPlayer

Try every domino on the board and find the domino which will go with highest score returned. If nothing can be played, return any domino that can’t be played

simplePlayer :: DomsPlayer

Find the first domino which can go on the board. If nothing is found, return absurd domino value.

scoreN :: Board -> Domino-> (Int, End)

if domino can be played on the board, then play it and then get the score with End type. If it can’t be played, return minus score. So later I can find any domino that can be played at least with 0 score. If nothing can be played hsdPlayer will return the domino that couldn’t be played in this scoreN function

playDom :: Domino->Board->End->Maybe Board

If the domino can be played on given end, then return the updated board with the domino. If played P is false then play the domino.

isJust :: (Maybe a) -> Bool

If maybe fns have any value, returns true, otherwise false. To check the domino can be played or not

resMaybe :: (Maybe a)->a

If the maybe fns can return some value, get that value. It is for updating the board to get the score

playedP :: Domino->Board-> Bool

For checking the domino is played on the board twice or not.

scoreBoard :: Board -> Int

Calculate the score of the given board. It is for finding the domino with highest score

robby :: DomsPlayer->DomsPlayer->Board->Int->Int->[(Int,Int)]

This function takes two integers. One for seed number and the other for keep calling each player. First make two hand sets. Hand1 is the first 9 dominoes that are took out from the randomized list of dominoes. Hand2 is the next 9 dominoes. The rest dominoes will not be used. Player1 be called, if the repeat integer is 1 and if it can find the domino that can be played on the board. If not, playDom will return nothing. And this will be the end of the game, so return (0,0) which have no effect on the score. But if the playDom return the maybe board updated, then get the score and store the score on the left side of the tuple(which will be returned finally) with 0 in the right side. Additionally the initial board will be set as zero. Next, call the player2 by updating the board and changing the repeat integer to 2. If player2 have any domino that will go on the updated board then play it and get and store the sore on the right side of the tuple with 0 on the left side. Finally, this function will return the list of tuples. Odd numbers of the list will have the scores only on the left side for player1. Even numbers will have the score only on the right side just for player2. The last one will be always (0,0)

playDomsRound :: DomsPlayer->DomsPlayer->Int->(Int, Int)

By using robby function (When calling this function, set the board as [] for the first go), get the list of tuples. By summing up all the first values of the tuples in the list, the total score for the player1 can be made. As same, summing up all the second values of the tuples in the list will return the total score for the player2. Make one final tuple as a result. Left side will be total score for the player1 and the right side will be the total score for player2.

shuffleDoms :: Int -> Hand

Return thelist of unordered dominoes(28). Take the seed number and make the ordered set of dominoes as randomized one.