**Connect 4 Robot Report**

**Stream 4 - Group 6**

Daniel Rolinsky               - 21095495

Anthony Hoalga              - 21085218

Brandon Narine               - 21087557

Vicente III Hernandez     - 21092350

December 5 2023

Forematter

**Summary** - tony

**Acknowledgements** - vicente & dj big danny d

**Table of Contents -** every one

**List of Tables and Figures -** brandon and tony

**Introduction -** Brandon

**Scope  -** brandon

**Constraints and Criteria -** Daniel

**Mechanical Design and Implementation -** king danny and tony baloney

**Software Design and Implementation -** vicente and b - dawg

**Verification**- vicente

**Project Plan** - vicente

**Conclusions -** tony

**Recommendations -** tony

Back Matter

Format - brandon

**References -** brandon

**Appendices –** everyone

Appendix A

*Official Code used on demonstration day*

#pragma config(Motor, motorA, ,

tmotorEV3\_Large, PIDControl, reversed, encoder)

//Constants

const int BOARD\_ROWS = 6;

const int BOARD\_COLUMNS = 7;

const int DEFAULT\_DISPLAY\_LINE = 3;

const int HUMAN\_TOKEN\_TYPE = 1;

const int ROBOT\_TOKEN\_TYPE = 2;

const int MIDDLE\_COLUMN = 4;

const int TWO\_LINE = 2;

const int THREE\_LINE = 5;

const int OPPO\_THREE\_LINE = 4;

const int FOUR\_LINE = 900;

const int colAngle[7] = {0, -70, -145, -220, -300, -400, -528};

//Data Structs

typedef struct

{

int score, columnOfMove;

} minimaxReturns ;

//Function Prototyping

int horizontalCheck(int row, int column, int playerToken,

int oppoToken, bool oppoCheck = false);

int verticalCheck(int row, int column, int playerToken, int oppoToken, bool oppoCheck = false);

int positiveSlopeCheck(int row, int column, int playerToken,

int oppoToken, bool oppoCheck = false);

int negativeSlopeCheck(int row, int column, int playerToken,

int oppoToken, bool oppoCheck = false);

void addTokenToArray(int \*columnHeights, int row, int column,

int tokenType);

void removeTokenInArray(int \*columnHeights, int row, int column);

int robotMove(int \*columnHeights);

void minimaxAlg(int \*columnHeights, int depth, bool maxPlayer, minimaxReturns &values, int columnOfMove = 0);

int scoreBoard(int columnOfMove, int playerToken, int oppoToken);

int scorePoints(int sum, bool oppoCheck);

int max(int num1, int num2);

int min(int num1, int num2);

void sensorConfig();

void configureMotors();

void waitButton(TEV3Buttons buttonName);

void motorHold(int column);

void playGame(int & currentPlayer, bool playerTwo,

bool & exitProgram);

void dropToken(int \*columnHeights, int choiceCol,

bool isHumanPlaying);

void sortTokens();

void humanMove(int \*columnHeights, int currentCol, int currentPlayer, bool & exitProgram);

void HumanMove2(int \*columnHeights, int currentCol, int currentPlayer, bool & exitProgram);

int gameWon(int currentPlayer);

void resetSpinner();

int moveSelect(int currentCol, int currentPlayer, bool & exitProgram);

void spinnerMotor(bool isHumanPlaying);

bool legalCheck(int choiceCol);

//2D Array

int boardArray[BOARD\_ROWS][BOARD\_COLUMNS] =

{

{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0}

};

task main()

{

bool playAgain = true;

bool exitProgram = false;

while(playAgain == true && exitProgram == false)

{

MSMMUXinit();

SensorType[S4] = sensorI2CCustom;

int currentPlayer = 0;

sensorConfig();

configureMotors();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Put Cartridges");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "On Top!");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "Then Press");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 9, "Enter To Play!");

waitButton(buttonEnter);

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Select Mode:");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "Up Button: PvP");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "Down Button:");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 9, "PvCPU");

bool playerTwo = false;

bool modeSelected = false;

while(!modeSelected){

if(getButtonPress(buttonDown))

{

while(getButtonPress(buttonDown)){}

playerTwo = true;

modeSelected = true;

}

else if(getButtonPress(buttonUp))

{

while(getButtonPress(buttonUp)){}

playerTwo = false;

modeSelected = true;

}

}

playGame(currentPlayer, playerTwo, exitProgram);

if(!exitProgram){

eraseDisplay();

if(gameWon(currentPlayer) == 3)

{

displayBigTextLine(DEFAULT\_DISPLAY\_LINE , "Game Has Ended");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3 , "In A Tie");

}

else if(gameWon(currentPlayer) == 1)

{

displayBigTextLine(DEFAULT\_DISPLAY\_LINE , "Congratulations!");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "Player 1");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "You've Won!");

}

else if(playerTwo == false)

{

displayBigTextLine(DEFAULT\_DISPLAY\_LINE , "Congratulations!");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "Player 2");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "You've Won!");

}

else if(playerTwo == true)

{

displayBigTextLine(DEFAULT\_DISPLAY\_LINE , "GAME OVER!");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "The Robot Has");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "Beaten You :(");

}

wait1Msec(4000);

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Put Cartridges");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "At The Bottom");

wait1Msec(5000);

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE , "Please Swap The");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3 ,"Motor on Top For");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6,"The Motor On The");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 9,"Bottom");

wait1Msec(5000);

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE , "Please Press The");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3 ,"Touch Sensor");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6,"When Ready To");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 9,"Sort The Tokens");

while(SensorValue[S1] == 0)

{}

eraseDisplay();

resetSpinner();

sortTokens();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Put Cartridges ");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 3, "At The Top!");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "Press Enter");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 9, "Once Finished");

waitButton(buttonEnter);

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Press Up To");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +3 , "Play Again");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 6, "Or Press");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE + 9, "Down To Exit");

bool choiceSelected = false;

while(!choiceSelected){

if(getButtonPress(buttonUp))

{

while(getButtonPress(buttonUp)){}

playAgain = true;

choiceSelected = true;

for(int row = 0; row < BOARD\_ROWS; row ++)

{

for(int col = 0; col < BOARD\_COLUMNS; col ++)

{

boardArray[row][col] = 0;

}

}

}

else if(getButtonPress(buttonDown))

{

while(getButtonPress(buttonDown)){}

playAgain = false;

choiceSelected = true;

}

}

}

}

eraseDisplay();

time1[T2] =0;

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Thank You");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +3 , "For Playing!");

while(time1[T2] < 7000){}

eraseDisplay();

}

int gameWon(int currentPlayer)

// assumption: the red = 1 = human, yellow = 2 = robot

{

int draw = 3, notWon = 0, win = 0;

if(boardArray[0][0] != 0 && boardArray[0][1] != 0 && boardArray[0][2] != 0 && boardArray[0][3] != 0

&& boardArray[0][4] != 0 && boardArray[0][5] != 0 && boardArray[0][6] != 0)

{

return draw;

}

if(currentPlayer == 1)

{

win = 1; // human won

}

else if(currentPlayer == 2)

{

win = 2; // robot won

}

for(int row = 5; row >= 0; row--)

{

for(int col = 0; col < 7; col++)

{

int row1 = row - 1;

int row2 = row - 2;

int row3 = row - 3;

if(row > 2)

{

if(boardArray[row][col] == currentPlayer

&& boardArray[row1][col] == currentPlayer

&& boardArray[row2][col] == currentPlayer

&& boardArray[row3][col] == currentPlayer)

{

return win;

} // checks vertical

if(col < 4)

{

if(boardArray[row][col] == currentPlayer

&& boardArray[row][col + 1] == currentPlayer

&& boardArray[row][col + 2] == currentPlayer

&& boardArray[row][col + 3] == currentPlayer)

{

return win;

} // checks horizontal

if(boardArray[row][col] == currentPlayer

&& boardArray[row1][col + 1] == currentPlayer

&& boardArray[row2][col + 2] == currentPlayer

&& boardArray[row3][col + 3] == currentPlayer)

{

return win;

} // checks +ve slope

}

}

else if(col < 4)

{

if(boardArray[row][col] == currentPlayer

&& boardArray[row][col + 1] == currentPlayer

&& boardArray[row][col + 2] == currentPlayer

&& boardArray[row][col + 3] == currentPlayer)

{

return win;

} // checks horizontal

if(boardArray[row][col] == currentPlayer

&& boardArray[row + 1][col + 1] == currentPlayer

&& boardArray[row + 2][col + 2] == currentPlayer

&& boardArray[row + 3][col + 3] == currentPlayer)

{

return win;

} // checks -ve slope

}

}

}

return notWon; //not won if neither the robot nor human has a connect 4

}

void playGame(int &currentPlayer, bool playerTwo, bool &exitProgram){

currentPlayer = 1;

int columnHeights[BOARD\_COLUMNS] = {0,0,0,0,0,0,0};

int currentCol = 1;

while(gameWon(currentPlayer) == 0 && exitProgram == false)

{

currentPlayer = 1;

humanMove(columnHeights, currentCol, currentPlayer, exitProgram);

if(gameWon(currentPlayer) == 0 && exitProgram == false)

{

if(playerTwo == true)

{

currentPlayer = 2;

robotMove(columnHeights);

}

else

{

currentPlayer = 2;

HumanMove2(columnHeights, currentCol, currentPlayer, exitProgram);

}

}

}

return;

}

void humanMove(int \*columnHeights, int currentCol, int currentPlayer, bool &exitProgram)

{

int choiceCol = 1;

do {

choiceCol = moveSelect(choiceCol, currentPlayer, exitProgram);

} while(!legalCheck(choiceCol) && exitProgram == false);

if(exitProgram == true)

return;

dropToken(columnHeights, choiceCol, true); // means human is playing

return;

}

int robotMove(int \*columnHeights)

{

minimaxReturns values;

minimaxAlg(columnHeights, 2, true, values);

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Robot Move");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +3 , "Current Column:");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +6 , "%d", values.columnOfMove);

dropToken(columnHeights, values.columnOfMove, false);

return values.columnOfMove;

}

void HumanMove2(int \*columnHeights, int currentCol, int currentPlayer, bool &exitProgram){

int choiceCol = 1;

do {

choiceCol = moveSelect(currentCol, currentPlayer, exitProgram);

} while(!legalCheck(choiceCol) && exitProgram == false);

if(exitProgram == true)

return;

dropToken(columnHeights, choiceCol, false); // means human is playing

return;

}

int moveSelect(int currentCol, int currentPlayer, bool &exitProgram)

{

int selectCol = currentCol;

time1[T1] = 0;

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Player: %d", currentPlayer);

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +3, "Current Column:");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +6 , "%d", selectCol);

while(!getButtonPress(buttonEnter) && !exitProgram)

{

if(SensorValue[S1] == 1 && time1[T1] > 1500){

exitProgram = true;

}

else if(SensorValue[S1] == 0){

time1[T1] = 0;

}

if(getButtonPress(buttonUp))

{

while(getButtonPress(buttonUp))

{}

if(selectCol == 7)

{

selectCol = 1;

}

else

{

selectCol++;

}

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Player: %d", currentPlayer);

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +3 , "Current Column:");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +6 , "%d", selectCol);

wait1Msec(500);

}

else if(getButtonPress(buttonDown))

{

while(getButtonPress(buttonDown))

{}

if(selectCol == 1)

{

selectCol = 7;

}

else

{

selectCol--;

}

eraseDisplay();

displayBigTextLine(DEFAULT\_DISPLAY\_LINE, "Player: %d", currentPlayer);

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +3 , "Current Column:");

displayBigTextLine(DEFAULT\_DISPLAY\_LINE +6 , "%d", selectCol);

wait1Msec(500);

}

}

while(getButtonPress(buttonEnter))

{}

return selectCol;

}

bool legalCheck(int choiceCol)

{

if(boardArray[0][choiceCol - 1] == 0 && choiceCol >= 1 && choiceCol <= 7)

{

return true;

}

return false;

}

void spinnerMotor(bool isHumanPlaying){

if(isHumanPlaying){

motor[motorC] = 10;

while (nMotorEncoder[motorC] < 95){}

motor[motorC] = 0;

}

else

{

motor[motorC] = -10;

while (nMotorEncoder[motorC] > -5){}

motor[motorC] = 0;

}

return;

}

void configureMotors()

{

nMotorEncoder[motorA] = nMotorEncoder[motorB] = nMotorEncoder[motorC] = nMotorEncoder[motorD] = 0;

wait1Msec(50);

motor[motorA] = motor[motorB] = motor[motorC] = motor[motorD]= 0 ;

wait1Msec(50);

MSMMotorEncoderReset(mmotor\_S4\_1);

}

void sensorConfig() {

SensorType[S1] = sensorEV3\_Touch;

wait1Msec(50);

SensorType[S2] = sensorEV3\_Color;

wait1Msec(50);

SensorMode[S2] = modeEV3Color\_Color;

wait1Msec(50);

}

void dropToken(int \*columnHeights, int choiceCol, bool isHumanPlaying)

{

choiceCol -= 1;

motor[motorA] = 15;

motor[motorD] = 15;

while(nMotorEncoder[motorA] >= colAngle[choiceCol])

{}

clearTimer(T4);

bool motorActivated = false;

while(time1[T4] < 4000)

{

motorHold(choiceCol);

if(time1[T4] > 1000 && !motorActivated)

{

spinnerMotor(isHumanPlaying);

motorActivated = true;

}

}

motor[motorA] = motor[motorD] = 0;

wait1Msec(500);

int tokenType = HUMAN\_TOKEN\_TYPE;

if(!isHumanPlaying)

{

tokenType = ROBOT\_TOKEN\_TYPE;

}

int emptyTokenRow = (BOARD\_ROWS - 1) - columnHeights[choiceCol];

boardArray[emptyTokenRow][choiceCol] = tokenType;

columnHeights[choiceCol] += 1;

motor[motorA] = -15;

setMotorBrakeMode(motorD, motorCoast);

while(nMotorEncoder[motorA] <= 0)

{}

setMotorBrakeMode(motorD, motorBrake);

motor[motorA] = 0;

motor[motorD] = 0;

clearTimer(T4);

while(time1(T4) < 500)

{

motorHold(0);

}

motor[motorA] = 0;

motor[motorD] = 0;

return;

}

void motorHold(int column)

{

while(nMotorEncoder[motorA] < colAngle[column])

{

motor[motorD] = -2;

motor[motorA] = -3;

}

while(nMotorEncoder[motorA] > colAngle[column])

{

motor[motorD] = 3;

motor[motorA] = 2;

}

motor[motorD] = 2;

motor[motorA] = -2;

}

void sortTokens(){

motor[motorC] = -10;

time1[T3] = 0;

int previousMotor = 100;

while(SensorValue[S1] == 0){

if(SensorValue[S2] == 4) {

motor[motorB] = -35;

while(nMotorEncoder[motorB] > -55){}

motor[motorB] = 0;

}

else if(SensorValue[S2] == 5) {

wait1Msec(500);

motor[motorB] = 35;

while(nMotorEncoder[motorB] < 0){}

motor[motorB] = 0;

}

if(time1[T3] >= 500){

if(previousMotor == nMotorEncoder[motorC])

{

motor[motorC] = 10;

wait1Msec(1000);

motor[motorC] = -10;

}

previousMotor = nMotorEncoder[motorC];

time1[T3] = 0;

}

}

motor[motorC] = 0;

motor[motorB] = 35;

while(nMotorEncoder[motorB] < 0){}

motor[motorB] = 0;

}

void waitButton(TEV3Buttons buttonName)

{

while(!getButtonPress(buttonName))

{}

while(getButtonPress(buttonName))

{}

return;

}

void resetSpinner()

{

MSMMotor(mmotor\_S4\_2,2);

while(MSMMotorEncoder(mmotor\_S4\_2) < 360)

{}

MSMotorStop(mmotor\_S4\_2);

}

int scorePoints(int sum, bool oppoCheck)

{

if(!oppoCheck)

{

if(sum == 2)

{

return TWO\_LINE;

}

else if(sum == 3)

{

return THREE\_LINE;

}

}

else

{

if(sum == 3)

{

return -1 \* OPPO\_THREE\_LINE;

}

}

return 0;

}

int horizontalCheck(int row, int column, int playerToken,

int oppoToken, bool oppoCheck)

{

int score = 0;

const int START\_ROW = row;

const int START\_COLUMN = column;

int sum = 1;

for(int colOffset = 1; colOffset < 4; colOffset++)

{

int currentToken = boardArray[START\_ROW][START\_COLUMN + colOffset];

if(currentToken == oppoToken || START\_COLUMN + colOffset == BOARD\_COLUMNS)

{

return 0;

}

else if(currentToken == playerToken)

{

sum++;

}

}

score += scorePoints(sum, oppoCheck);

return score;

}

int verticalCheck(int row, int column, int playerToken, int oppoToken, bool oppoCheck)

{

int score = 0;

const int START\_ROW = row;

const int START\_COLUMN = column;

int sum = 1;

for(int rowOffset = 1; rowOffset < 4; rowOffset++)

{

int currentToken = boardArray[START\_ROW - rowOffset][START\_COLUMN];

if(currentToken == oppoToken || START\_ROW - rowOffset == -1)

{

return 0;

}

else if(currentToken == playerToken)

{

sum++;

}

}

score += scorePoints(sum, oppoCheck);

return score;

}

int positiveSlopeCheck(int row, int column, int playerToken,

int oppoToken, bool oppoCheck)

{

int score = 0;

const int START\_ROW = row;

const int START\_COLUMN = column;

int sum = 1;

for(int slopeOffset = 1; slopeOffset < 4; slopeOffset++)

{

int currentToken = boardArray[START\_ROW - slopeOffset][START\_COLUMN + slopeOffset];

if(currentToken == oppoToken || START\_ROW - slopeOffset == -1 || START\_COLUMN + slopeOffset == BOARD\_COLUMNS)

{

return 0;

}

else if(currentToken == playerToken)

{

sum++;

}

}

score += scorePoints(sum, oppoCheck);

return score;

}

int negativeSlopeCheck(int row, int column, int playerToken,

int oppoToken, bool oppoCheck)

{

int score = 0;

const int START\_ROW = row;

const int START\_COLUMN = column;

int sum = 1;

for(int slopeOffset = 1; slopeOffset < 4; slopeOffset++)

{

int currentToken = boardArray[START\_ROW + slopeOffset][START\_COLUMN + slopeOffset];

if(currentToken == oppoToken || START\_ROW + slopeOffset == BOARD\_ROWS || START\_COLUMN + slopeOffset == BOARD\_COLUMNS)

{

return 0;

}

else if(currentToken == playerToken)

{

sum++;

}

}

score += scorePoints(sum, oppoCheck);

return score;

}

int scoreBoard(int columnOfMove, int playerToken, int oppoToken)

{

int score = 0;

int scoreCoefficient = 1;

if(playerToken == HUMAN\_TOKEN\_TYPE)

{

scoreCoefficient = -1;

}

if(columnOfMove == 3)

{

score += MIDDLE\_COLUMN;

}

for(int row = 5; row >= 0 ; row--)

{

for(int col = 0; col < 7; col++)

{

int currentToken = boardArray[row][col];

if(currentToken != 0 && row > 2)

{

//check Vertical

if(currentToken == playerToken)

{

score += verticalCheck(row, col, playerToken, oppoToken);

}

else

{

score += verticalCheck(row, col, oppoToken, playerToken, true);

}

if(col < 4)

{

//check Horizontal and +Slope

if(currentToken == playerToken)

{

score += horizontalCheck(row, col, playerToken, oppoToken);

score += positiveSlopeCheck(row, col, playerToken, oppoToken);

}

else

{

score += horizontalCheck(row, col, oppoToken, playerToken, true);

score += positiveSlopeCheck(row, col, oppoToken, playerToken, true);

}

}

}

else if(currentToken != 0 && col < 4)

{

//check Horizontal and -Slope

if(currentToken == playerToken)

{

score += horizontalCheck(row, col, playerToken, oppoToken);

score += negativeSlopeCheck(row, col, playerToken, oppoToken);

}

else

{

score += horizontalCheck(row, col, oppoToken, playerToken, true);

score += negativeSlopeCheck(row, col, oppoToken, playerToken, true);

}

}

}

}

score \*= scoreCoefficient;

return score;

}

//Robot Player Functions

void minimaxAlg(int \*columnHeights, int depth, bool maxPlayer, minimaxReturns &values, int columnOfMove)

{

const int wonRobot = gameWon(ROBOT\_TOKEN\_TYPE);

const int wonHuman = gameWon(HUMAN\_TOKEN\_TYPE);

if(depth == 0 || wonRobot || wonHuman)

{

minimaxReturns scoreValue;

if(maxPlayer)

{

if(wonRobot == ROBOT\_TOKEN\_TYPE)

{

values.score = FOUR\_LINE;

values.columnOfMove = scoreValue.columnOfMove;

return;

}

else if(wonHuman == HUMAN\_TOKEN\_TYPE)

{

values.score = -1 \* FOUR\_LINE;

values.columnOfMove = scoreValue.columnOfMove;

return;

}

scoreValue.score = scoreBoard(columnOfMove, HUMAN\_TOKEN\_TYPE, ROBOT\_TOKEN\_TYPE);

}

else

{

if(wonRobot == ROBOT\_TOKEN\_TYPE)

{

values.score = FOUR\_LINE;

values.columnOfMove = scoreValue.columnOfMove;

return;

}

else if(wonHuman == HUMAN\_TOKEN\_TYPE)

{

values.score = -1 \* FOUR\_LINE;

values.columnOfMove = scoreValue.columnOfMove;

return;

}

scoreValue.score = scoreBoard(columnOfMove, ROBOT\_TOKEN\_TYPE, HUMAN\_TOKEN\_TYPE);

}

values.score = scoreValue.score;

values.columnOfMove = scoreValue.columnOfMove;

return;

}

if(maxPlayer)

{

minimaxReturns maxValues;

maxValues.score = -9999;

maxValues.columnOfMove = 1;

for(int colDropIndex = 0; colDropIndex < BOARD\_COLUMNS; colDropIndex++) //Drops a token in each column to score that potential move

{

const int emptyTokenRow = (BOARD\_ROWS - 1) - columnHeights[colDropIndex]; //Checks if a column is not full

if(emptyTokenRow > -1)

{

addTokenToArray(columnHeights, emptyTokenRow, colDropIndex, ROBOT\_TOKEN\_TYPE);

minimaxReturns possibleMoveScore;

minimaxAlg(columnHeights, depth - 1, false, possibleMoveScore, colDropIndex);

if(max(maxValues.score, possibleMoveScore.score) == possibleMoveScore.score

&& possibleMoveScore.score != maxValues.score)

{

maxValues.score = possibleMoveScore.score;

maxValues.columnOfMove = colDropIndex + 1;

}

removeTokenInArray(columnHeights, emptyTokenRow, colDropIndex);

}

}

values.score = maxValues.score;

values.columnOfMove = maxValues.columnOfMove;

return;

}

else

{

minimaxReturns minValues;

minValues.score = 9999;

minValues.columnOfMove = 1;

for(int colDropIndex = 0; colDropIndex < BOARD\_COLUMNS; colDropIndex++) //Drops a token in each column to score that potential move

{

const int emptyTokenRow = (BOARD\_ROWS - 1) - columnHeights[colDropIndex]; //Checks if a column is not full

if(emptyTokenRow > -1)

{

addTokenToArray(columnHeights, emptyTokenRow, colDropIndex, HUMAN\_TOKEN\_TYPE);

minimaxReturns possibleMoveScore;

minimaxAlg(columnHeights, depth - 1, true, possibleMoveScore, colDropIndex);

if(min(minValues.score, possibleMoveScore.score) == possibleMoveScore.score && possibleMoveScore.score != minValues.score)

{

minValues.score = possibleMoveScore.score;

minValues.columnOfMove = colDropIndex + 1;

}

removeTokenInArray( columnHeights, emptyTokenRow, colDropIndex);

}

}

values.score = minValues.score;

values.columnOfMove = minValues.columnOfMove;

return;

}

}

void addTokenToArray(int \*columnHeights, int row, int column, int tokenType)

{

boardArray[row][column] = tokenType;

columnHeights[column] += 1;

return;

}

void removeTokenInArray(int \*columnHeights, int row, int column)

{

boardArray[row][column] = 0;

columnHeights[column] -= 1;

return;

}

int max(int num1, int num2)

{

if(num1 > num2) {

return num1;

}

return num2;

}

int min(int num1, int num2)

{

if(num1 < num2) {

return num1;

}

return num2;

}

**Daniel - github**