## ΕΙΣΑΓΩΓΗ ΚΑΙ MOTIVATION

- Τι αφορα το project:

Την υλοποίηση του παιχνιδιού Score Four

- Ποιο πρόβλημα λύνει;

Το Score 4 βοηθά στην αντικειμενική και απλή αξιολόγηση των επιδόσεων, με στόχο την αναγνώριση των τομέων που χρειάζονται βέλτιωση και την υποστήριξη της προόδου.

- Γιατί το επιλέξατε ;

Απόδειξη της ανώτερης σκέψης του ανθρώπινου νου έναντι του Η/Υ

## OBJECTIVE & SCOPE

- Κύριοι στόχοι του project:

Αξιολόγηση επιδόσεων:Το σύστημα βαθμολογεί διάφορους τομείς ή παραμέτρους (π.χ απόδοση, ποιότητα , παραγωγικότητα κλπ).

Βελτίωση διαδικασιών: Βοηθάει στον εντοπισμό περιοχών που χρειάζονται βελτίωση, έτσι ώστε να ληφθούν τα κατάλληλα μέτρα.

Αναγνώριση επιτυχιών: Παρέχει έναν τρόπο να αναγνωρίζονται οι επιτυχίες ή οι τομείς που αποδίδουν καλύτερα.

- Δυνατότητες & Λειτουργίες που υποστηρίζει

Διαχείριση Σκορ & Πρόοδου:

Το Score 4 επιτρέπει την παρακολούθηση της προόδου ή της επίδοσης ενός ατόμου ή ομάδας με την πάροδο του χρόνου.

Μπορεί να παρέχει πληροφορίες για την εξέλιξη των επιδόσεων και να εντοπίζει αν υπάρχουν αυξομειώσεις στην απόδοση.

## SYSTEM ARCHITECTURE

- Πως λειτουργεί το σύστημα;

Αποτελείται από δύο παίκτες(Υπολογιστής και Χρήστς).Ο κάθε παίκτης παίζει εναλλάξ με κύριο στόχο την επίτευξη κάθετων,οριζόντιων και διαγώνων τετράδων.Νικητής είναι αυτός που θα κάνει πρώτος την τετράδα.

- Κύριοι components;

  Backend

- Τεχνολογίες που χρησιμοποιήθηκαν & λόγοι επιλογής;

  Visual Studio 2022.

- Κυριοι λόγοι επιλογής:
  1)Καλύτερο προγραμματιστικό περιβάλλον
  2)Δυνατότητα προσθήκης γραφικών

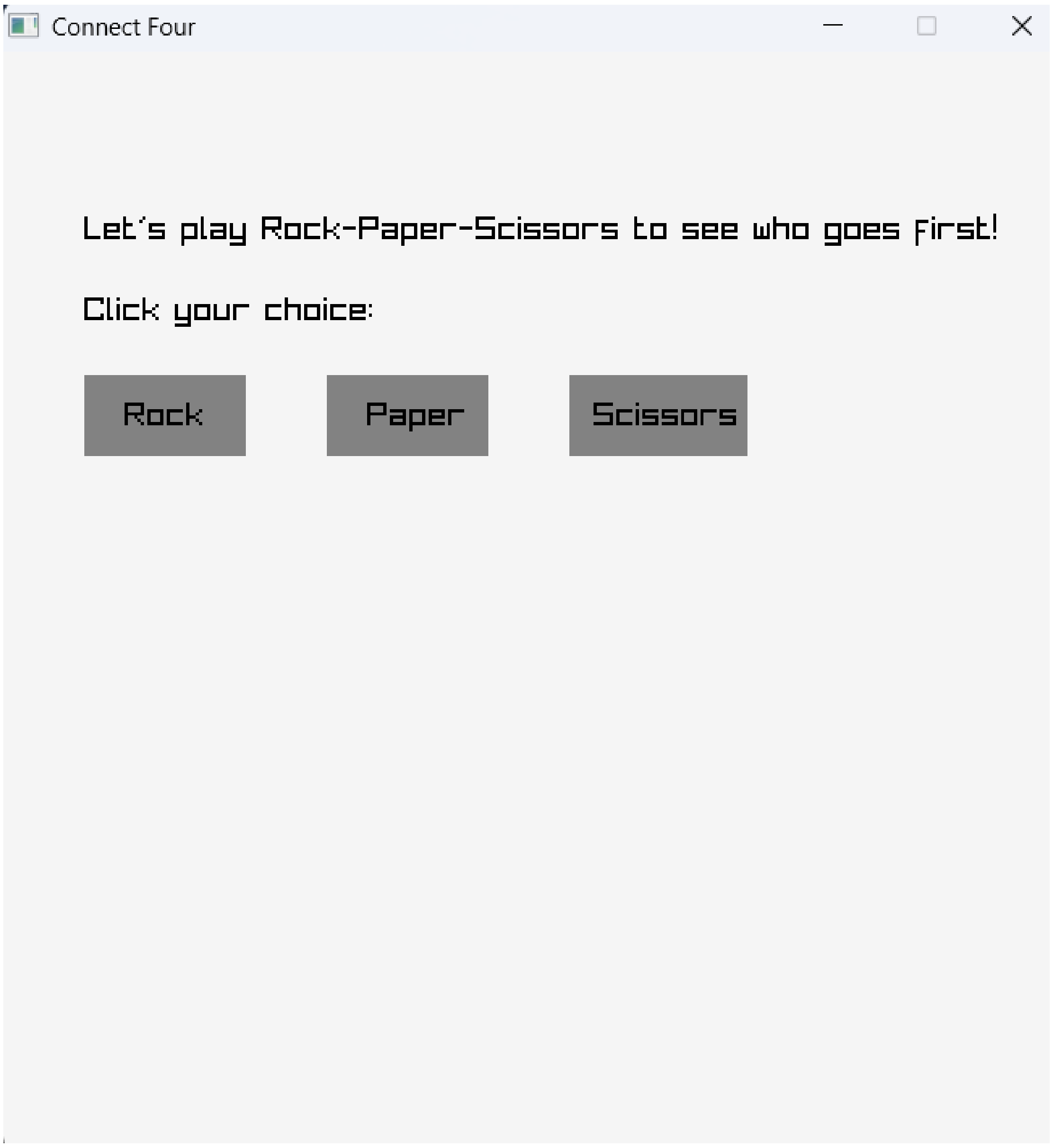
## ΚΩΔΙΚΑΣ & ΥΛΟΠΟΙΗΣΗ

```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <cstdlib>      //dhlwsh bibliouhkwn
#include <ctime>
#include <algorithm>
#include <string>
#include "raylib.h"

using namespace std;

const int ROWS = 6;  //arxikopoihsh grammwn pinaka
const int COLS = 7;  //arxikopoihsh sthlwn pinaka
const int PLAYER = 1;
const int AI = 2;
const int MAX_DEPTH = 5;

// GUI constants
const int CELL_SIZE = 80;
const int BOARD_OFFSET_X = 50;
const int BOARD_OFFSET_Y = 50;  //megethi pinaka
const int SCREEN_WIDTH = BOARD_OFFSET_X * 2 + COLS * CELL_SIZE;
const int SCREEN_HEIGHT = BOARD_OFFSET_Y * 2 + ROWS * CELL_SIZE + 100;

vector<vector<int>> board(ROWS, vector<int>(COLS, 0));
bool gameInProgress = false;
int turn = PLAYER;
int gameOver = 0;
string gameResult = "";

void printBoard() {
    cout << "\n";
    for (int r = 0; r < ROWS; r++) {
        for (int c = 0; c < COLS; c++) {
            if (board[r][c] == 0) cout << ". ";   //emfanish pinaka
            else if (board[r][c] == PLAYER) cout << "P ";
            else cout << "A ";
        }
        cout << "\n";
    }
    cout << "0 1 2 3 4 5 6\n";
}

int isValidMove(int col) {
    if (col < 0 || col >= COLS) return 0;  //elegxos egkyrothtas kinhshs
paikth
    return board[0][col] == 0;
}

int getNextOpenRow(int col) {
    for (int r = ROWS - 1; r >= 0; r--) {  // Epistrefei tin proti dia8esimi
grammi (apo kato pros ta pano) stin dothisa stili.
                                           // An i stili einai gemati,
epistrefei -1.
        if (board[r][col] == 0) return r;
    }
    return -1;
}

void dropPiece(int row, int col, int piece) {
    board[row][col] = piece;        //topothethsh diskoy stis theseis
}

int winningMove(int piece) {
    for (int c = 0; c < COLS - 3; c++) {
        for (int r = 0; r < ROWS; r++) {
            if (board[r][c] == piece && board[r][c+1] == piece &&
                board[r][c+2] == piece && board[r][c+3] == piece)
// Elegxei an yparxei nikifora kinisi gia ton paikti me to sygkekrimeno piece.

// Elegxei opızontia, katheta, kai diagonia (kai pros tis dyo kateuthynseis).

// Epistrefei 1 an yparxei grammi 4 omoion kommatiwn, alliws epistrefei 0.
                return 1;
        }
    }

    for (int c = 0; c < COLS; c++) {
        for (int r = 0; r < ROWS - 3; r++) {
            if (board[r][c] == piece && board[r+1][c] == piece &&
                board[r+2][c] == piece && board[r+3][c] == piece)
                return 1;
        }
    }

    for (int r = 3; r < ROWS; r++) {
        for (int c = 0; c < COLS - 3; c++) {
            if (board[r][c] == piece && board[r-1][c+1] == piece &&
                board[r-2][c+2] == piece && board[r-3][c+3] == piece)
                return 1;
        }
    }
    for (int r = 0; r < ROWS - 3; r++) {
        for (int c = 0; c < COLS - 3; c++) {
            if (board[r][c] == piece && board[r+1][c+1] == piece &&
                board[r+2][c+2] == piece && board[r+3][c+3] == piece)
                return 1;
        }
    }

    return 0;
}

bool isBoardFull() {
    for (int c = 0; c < COLS; c++) {
        if (isValidMove(c)) return false;  //elegxos gia gemato pinaka
    }
    return true;
}

int scorePosition(int piece) {
    if (winningMove(piece)) return 10000;                          //
Dinei skor gia mia thesi sto tamplo me vasi to an yparxei nikifora kinisi.
                                                                   // An o
paixths me to sygkekrimeno kommati kerdizei, epistrefei 10000.
                                                                  // An o
antipalos mporei na kerdisei stin epomeni kinisi, epistrefei -10000.
                                                                 // Alliws
epistrefei 0.
    if (winningMove(piece == PLAYER ? AI : PLAYER)) return -10000;
    return 0;
}

int minimax(int depth, int alpha, int beta, int maximizingPlayer) {
    if (depth == 0 || winningMove(PLAYER) || winningMove(AI) || isBoardFull())
{
        return scorePosition(AI);
    }

    if (maximizingPlayer) {
        int maxEval = numeric_limits<int>::min();
        for (int col = 0; col < COLS; col++) {
            if (isValidMove(col)) {
//algorithmos minmax-AI
                int row = getNextOpenRow(col);
                board[row][col] = AI;
                int eval = minimax(depth - 1, alpha, beta, 0);
                board[row][col] = 0;
                maxEval = max(maxEval, eval);
                alpha = max(alpha, eval);
                if (beta <= alpha) break;
            }
        }
        return maxEval;
    } else {
        int minEval = numeric_limits<int>::max();
        for (int col = 0; col < COLS; col++) {
            if (isValidMove(col)) {
                int row = getNextOpenRow(col);
                board[row][col] = PLAYER;
                int eval = minimax(depth - 1, alpha, beta, 1);
                board[row][col] = 0;
                minEval = min(minEval, eval);
                beta = min(beta, eval);
                if (beta <= alpha) break;
            }
        }
        return minEval;
    }
}

int getBestMove() {
    int bestScore = numeric_limits<int>::min();
    int bestCol = 3;

    for (int col = 0; col < COLS; col++) {                          // Vriskei
tin kaliteri kinisi gia ton AI paixti.
                                                                    // 1)
Elegxei an mporei na kerdisei amesa – epistrefei afti tin kinisi.
                                                                    // 2)
Elegxei an o antipalos mporei na kerdisei stin epomeni – kanei block.
                                                                   // 3) An
oute AI oute o paikths kerdizoun amesa, xrisimopoiei to minimax
                                                                   //    me
alpha-beta pruning gia na vrei tin pio skorpismeni kinisi.
                                                                   //
Epistrefei tin stili me to kalitero score.
        if (isValidMove(col)) {
            int row = getNextOpenRow(col);
            board[row][col] = AI;
            if (winningMove(AI)) {
                board[row][col] = 0;
                return col;
            }
            board[row][col] = 0;
        }
    }

    for (int col = 0; col < COLS; col++) {
        if (isValidMove(col)) {
            int row = getNextOpenRow(col);
            board[row][col] = PLAYER;
            if (winningMove(PLAYER)) {
                board[row][col] = 0;
                return col;
            }
            board[row][col] = 0;
        }
    }

    for (int col = 0; col < COLS; col++) {
        if (isValidMove(col)) {
            int row = getNextOpenRow(col);
            board[row][col] = AI;
            int score = minimax(MAX_DEPTH - 1, numeric_limits<int>::min(),
numeric_limits<int>::max(), 0);
            board[row][col] = 0;
            if (score > bestScore) {
                bestScore = score;
                bestCol = col;
            }
        }
    }

    if (!isValidMove(bestCol)) {
        for (int col = 0; col < COLS; col++) {
            if (isValidMove(col)) {
                bestCol = col;
                break;
            }
        }
    }

    return bestCol;
}

int getRandomChoice() {
    return rand() % 3;       //epilogh tyxaias epiloghs
}

int rockPaperScissors() {
    string choices[] = {"rock", "paper", "scissors"};     //// Paizei Rock-
Paper-Scissors gia na apofasistei poios paizei protos (PLAYER i AI).
    int userChoice = -1;
    bool validChoice = false;

    while (!validChoice) {
        ClearBackground(RAYWHITE); //here
        BeginDrawing();

        DrawText("Let's play Rock-Paper-Scissors to see who goes first!", 50,
100, 20, BLACK);
        DrawText("Click your choice:", 50, 150, 20, BLACK);



        DrawRectangle(50, 200, 100, 50, GRAY);
        DrawText("Rock", 75, 215, 20, BLACK);

//sxediash mplok epilogwn rock,paper,scissors
        DrawRectangle(200, 200, 100, 50, GRAY);
        DrawText("Paper", 225, 215, 20, BLACK);

        DrawRectangle(350, 200, 110, 50, GRAY);
        DrawText("Scissors", 365, 215, 20, BLACK);

        EndDrawing();


        if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
            Vector2 mousePos = GetMousePosition();

            if (mousePos.y >= 200 && mousePos.y <= 250) {
                if (mousePos.x >= 50 && mousePos.x <= 150) {
                    userChoice = 0; // Rock
                    validChoice = true;
                } else if (mousePos.x >= 200 && mousePos.x <= 300) {
                    userChoice = 1; // Paper
                    validChoice = true;
                } else if (mousePos.x >= 350 && mousePos.x <= 450) {
                    userChoice = 2; // Scissors
                    validChoice = true;
                }
            }
        }

        if (WindowShouldClose()) {
            CloseWindow();
            exit(0);
        }
    }

    int aiChoice = getRandomChoice();



    for (int i = 0; i < 60; i++) {
        ClearBackground(RAYWHITE);//here
        BeginDrawing();

        DrawText("Let's play Rock-Paper-Scissors to see who goes first!", 50,
100, 20, BLACK);
        DrawText(("You chose: " + choices[userChoice]).c_str(), 50, 200, 20,
BLACK);
        DrawText(("AI chose: " + choices[aiChoice]).c_str(), 50, 230, 20,
BLACK);
```

```
        EndDrawing();

        if (WindowShouldClose()) {
            CloseWindow();
            exit(0);
        }
    }

    if (userChoice == aiChoice) {
        return rockPaperScissors();
    }

    if ((userChoice == 0 && aiChoice == 2) || // h petra nikaei to xarti
        (userChoice == 1 && aiChoice == 0) || // to xarti nikaei thn petra
        (userChoice == 2 && aiChoice == 1)) { // to psalidi nikaei to xarti
        return PLAYER;
    } else {
        return AI;
    }
}

void resetGame() {

    for (int r = 0; r < ROWS; r++) {
        for (int c = 0; c < COLS; c++) {
            board[r][c] = 0;
        }
    }                                                   // epanafora paixnidou

    turn = rockPaperScissors();
    gameOver = 0;
    gameResult = "";
    gameInProgress = true;
}

void drawBoard() {

    DrawRectangle(BOARD_OFFSET_X - 10, BOARD_OFFSET_Y - 10,
                  COLS * CELL_SIZE + 20, ROWS * CELL_SIZE + 20, BLUE);


//// Zwgrafizei to tamplo kai ta diskakia tou paixnidiou stin othonh.
    for (int r = 0; r < ROWS; r++) {
        for (int c = 0; c < COLS; c++) {
            int x = BOARD_OFFSET_X + c * CELL_SIZE + CELL_SIZE / 2;
            int y = BOARD_OFFSET_Y + r * CELL_SIZE + CELL_SIZE / 2;
            if (board[r][c] == 0) {
                DrawCircle(x, y, CELL_SIZE / 2 - 5, LIGHTGRAY);
            } else if (board[r][c] == PLAYER) {
                DrawCircle(x, y, CELL_SIZE / 2 - 5, RED);
            } else {
                DrawCircle(x, y, CELL_SIZE / 2 - 5, YELLOW);
            }
        }
    }

    for (int c = 0; c < COLS; c++) {
        DrawText(TextFormat("%d", c),
                 BOARD_OFFSET_X + c * CELL_SIZE + CELL_SIZE / 2 - 5,
                 BOARD_OFFSET_Y + ROWS * CELL_SIZE + 20,
                 20, BLACK);
    }

    if (!gameOver && turn == PLAYER) {
        Vector2 mousePos = GetMousePosition();
        if (mousePos.y >= BOARD_OFFSET_Y && mousePos.y <= BOARD_OFFSET_Y +
ROWS * CELL_SIZE) {
            for (int col = 0; col < COLS; col++) {
                int colX = BOARD_OFFSET_X + col * CELL_SIZE;
                if (mousePos.x >= colX && mousePos.x < colX + CELL_SIZE &&
isValidMove(col)) {
                    DrawRectangleLines(colX, BOARD_OFFSET_Y - 10,
                                       CELL_SIZE, ROWS * CELL_SIZE + 20,
                                       GREEN);
                    break;
                }
            }
        }
    }
}

int main() {


    srand(time(0));


    SetConfigFlags(FLAG_MSAA_4X_HINT | FLAG_VSYNC_HINT);
    InitWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Connect Four");
    SetTargetFPS(60);
    SetExitKey(KEY_NULL);

    gameInProgress = false;
    int frameCount = 0;

    while (!WindowShouldClose()) {      // Kentriki epanalhpsh tou
paixnidiou:xeirizetai thn eisodo,thn logiki paixnidiou kai zwgrafisi.
        frameCount++;


        if (!gameInProgress && IsKeyPressed(KEY_SPACE)) {
            resetGame();
        }


        if (gameInProgress && !gameOver) {
            if (turn == PLAYER) {

                if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
                    Vector2 mousePos = GetMousePosition();

                    if (mousePos.y >= BOARD_OFFSET_Y && mousePos.y <=
BOARD_OFFSET_Y + ROWS * CELL_SIZE) {
                        for (int col = 0; col < COLS; col++) {
                            int colX = BOARD_OFFSET_X + col * CELL_SIZE;
                            if (mousePos.x >= colX && mousePos.x < colX +
CELL_SIZE) {
                                if (isValidMove(col)) {
                                    int row = getNextOpenRow(col);
                                    dropPiece(row, col, PLAYER);
                                    printBoard();

                                    if (winningMove(PLAYER)) {
                                        gameResult = "You win!";
                                        gameOver = 1;
                                    } else if (isBoardFull()) {
                                        gameResult = "Draw!";
                                        gameOver = 1;
                                    } else {
                                        turn = AI;
                                        frameCount = 0;
                                    }
                                }
                                break;
                            }
                        }
//minima gia to poios nikaei
                    }
                }
            } else {

                if (frameCount > 30) {
                    frameCount = 0;
                    int col = getBestMove();

                    if (isValidMove(col)) {
                        int row = getNextOpenRow(col);
                        dropPiece(row, col, AI);
                        printBoard();

                        if (winningMove(AI)) {
                            gameResult = "AI wins!";
                            gameOver = 1;
                        } else if (isBoardFull()) {
                            gameResult = "Draw!";
                            gameOver = 1;
                        } else {
                            turn = PLAYER;
                        }
                    } else {


                        for (int c = 0; c < COLS; c++) {
                            if (isValidMove(c)) {
                                col = c;
                                int row = getNextOpenRow(col);
                                dropPiece(row, col, AI);

                                if (winningMove(AI)) {
                                    gameResult = "AI wins!";
                                    gameOver = 1;
                                } else if (isBoardFull()) {
                                    gameResult = "Draw!";
                                    gameOver = 1;
                                } else {
                                    turn = PLAYER;
                                }
                                break;
                            }
                        }
                    }
                }
            }
        }

        // zwgrafisma
        BeginDrawing();
        ClearBackground(RAYWHITE);

        if (!gameInProgress) {

            DrawText("Connect Four", SCREEN_WIDTH / 2 - 120, 100, 40,
DARKBLUE);
            DrawText("Press SPACE to start", SCREEN_WIDTH / 2 - 120, 200, 20,
BLACK);
            DrawText("Press ESC to quit", SCREEN_WIDTH / 2 - 100, 250, 20,
BLACK);
        } else {

            drawBoard();



            if (gameOver) {
                DrawText(gameResult.c_str(), SCREEN_WIDTH / 2 - 100,
BOARD_OFFSET_Y + ROWS * CELL_SIZE + 60, 30, BLACK);
                DrawText("Press R to restart", SCREEN_WIDTH / 2 - 100,
BOARD_OFFSET_Y + ROWS * CELL_SIZE + 90, 20, GRAY);



                if (IsKeyPressed(KEY_R)) {
                    resetGame();
                }
            } else {
                string turnText = (turn == PLAYER) ? "Your turn" : "AI is
thinking...";
                DrawText(turnText.c_str(), SCREEN_WIDTH / 2 - 100,
BOARD_OFFSET_Y + ROWS * CELL_SIZE + 60, 30, BLACK);
            }
        }

        EndDrawing();



        if (WindowShouldClose()) {
            break;
        }
```

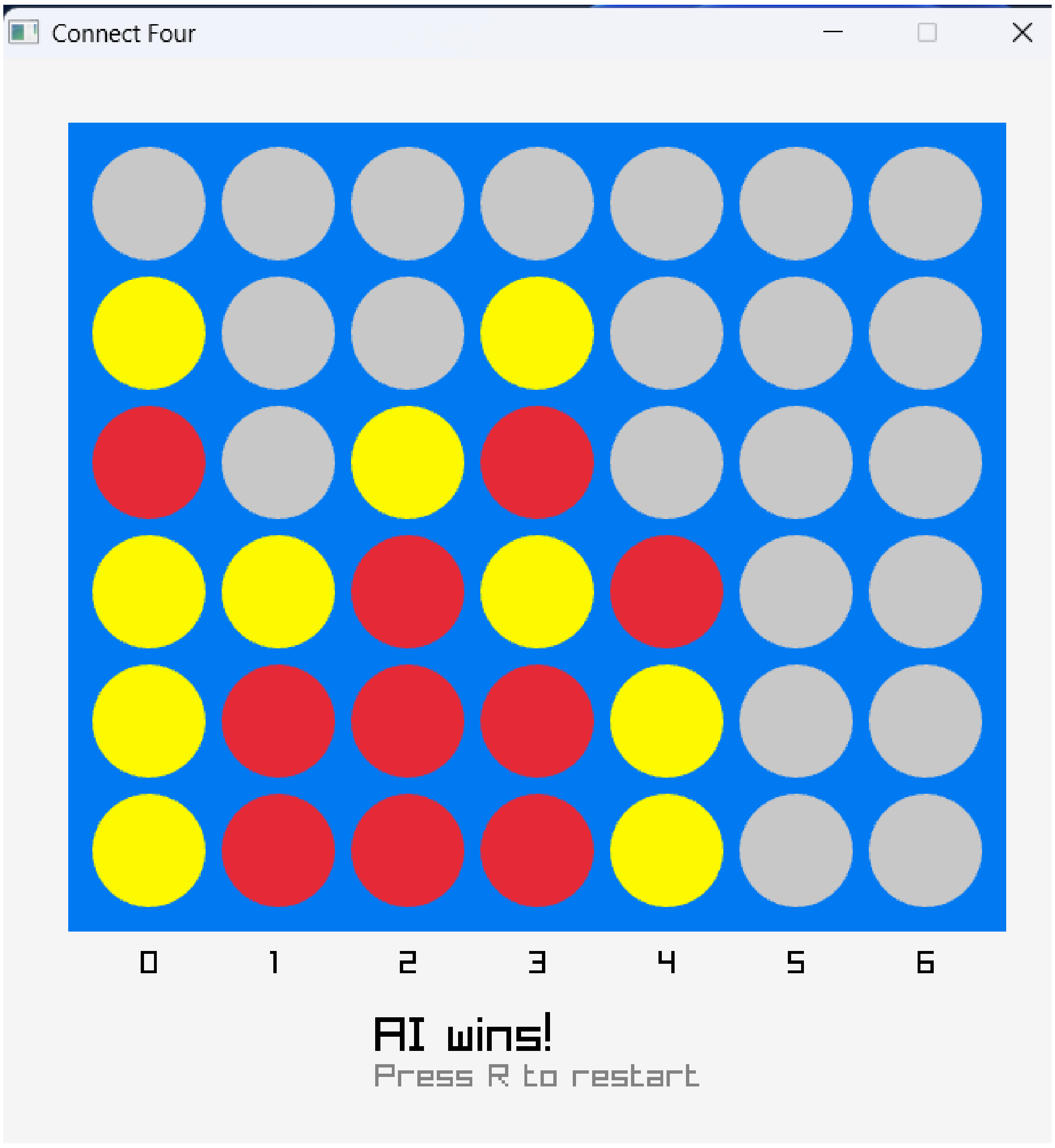
```
    }



    CloseWindow();
    return 0;
}
```

## ΑΠΟΤΕΛΕΣΜΑΤΑ & DEMO


Connect Four

Let's play Rock-Paper-Scissors to see who goes first!

Click your choice:



AI wins!
Press R to restart

ΣΥΓΚΡΙΣΗ ΜΕ AI-GENERATED CODE

```
#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <limits>


using namespace std;
```

```
const int ROWS = 6;

const int COLS = 7;

const int MAX_DEPTH = 4;

char board[ROWS][COLS];


void initializeBoard() {

   for (int i = 0; i < ROWS; ++i)

      for (int j = 0; j < COLS; ++j)

         board[i][j] = '.';

}


void printBoard() {

   cout << "\n";

   for (int i = 0; i < ROWS; ++i) {

      for (int j = 0; j < COLS; ++j)

         cout << board[i][j] << " ";

      cout << "\n";

   }

   for (int j = 1; j <= COLS; ++j)

      cout << j << " ";

   cout << "\n";

}


bool isColumnFull(int col) {

   return board[0][col] != '.';

}


bool placeDisc(int col, char symbol) {
    if (col < 0 || col >= COLS || isColumnFull(col)) return false;
    for (int i = ROWS - 1; i >= 0; --i) {
        if (board[i][col] == '.') {
            board[i][col] = symbol;
            return true;
        }
    }
    return false;
}

void undoMove(int col) {
    for (int i = 0; i < ROWS; ++i) {
        if (board[i][col] != '.') {
            board[i][col] = '.';
            break;
        }
    }
}

bool checkDirection(int row, int col, int dr, int dc, char symbol) {
    int count = 0;
    for (int i = 0; i < 4; ++i) {
        int r = row + dr * i;
        int c = col + dc * i;
        if (r >= 0 && r < ROWS && c >= 0 && c < COLS && board[r][c] == symbol)
            count++;
        else
            break;
    }
    return count == 4;
}

bool checkWin(char symbol) {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            if (board[i][j] == symbol)
                if (checkDirection(i, j, 0, 1, symbol) ||
                    checkDirection(i, j, 1, 0, symbol) ||
                    checkDirection(i, j, 1, 1, symbol) ||
                    checkDirection(i, j, 1, -1, symbol))
                    return true;
    return false;
}

bool isDraw() {
    for (int j = 0; j < COLS; ++j)
        if (board[0][j] == '.') return false;
    return true;
}

int evaluateBoard(char ai, char player) {
    if (checkWin(ai)) return 1000;
    if (checkWin(player)) return -1000;
    return 0;
}
```

```
int minimax(int depth, bool maximizingPlayer, char ai, char player) {
  int score = evaluateBoard(ai, player);
  if (depth == 0 || score == 1000 || score == -1000 || isDraw())
    return score;

  if (maximizingPlayer) {
    int maxEval = numeric_limits<int>::min();
    for (int col = 0; col < COLS; ++col) {
      if (!isColumnFull(col)) {
        placeDisc(col, ai);
        int eval = minimax(depth - 1, false, ai, player);
        undoMove(col);
        maxEval = max(maxEval, eval);
      }
    }
    return maxEval;
  } else {
    int minEval = numeric_limits<int>::max();
    for (int col = 0; col < COLS; ++col) {
      if (!isColumnFull(col)) {
        placeDisc(col, player);
        int eval = minimax(depth - 1, true, ai, player);
        undoMove(col);
        minEval = min(minEval, eval);
      }
    }
    return minEval;
  }
```

```
}

int getBestMove(char ai, char player) {
  int bestScore = numeric_limits<int>::min();
  int bestCol = -1;

  for (int col = 0; col < COLS; ++col) {
    if (!isColumnFull(col)) {
      placeDisc(col, ai);
      int score = minimax(MAX_DEPTH - 1, false, ai, player);
      undoMove(col);
      if (score > bestScore) {
        bestScore = score;
        bestCol = col;
      }
    }
  }

  return bestCol;
}

int main() {
  srand(time(0));
  initializeBoard();
  printBoard();

  char player = 'X';
  char computer = 'O';
```

```
while (true) {
  int col;
  cout << "\nYour turn (1-7): ";
  cin >> col;
  col--;

  if (!placeDisc(col, player)) {
    cout << "Invalid column. Try again.\n";
    continue;
  }

  printBoard();
  if (checkWin(player)) {
    cout << "Congratulations! You win!\n";
    break;
  }
  if (isDraw()) {
    cout << "It's a draw!\n";
    break;
  }

  int compMove = getBestMove(computer, player);
  placeDisc(compMove, computer);
  cout << "\nComputer played column " << (compMove + 1) << "\n";

  printBoard();
  if (checkWin(computer)) {
```

```
            cout << "Computer wins! Better luck next time.\n";
            break;
        }
        if (isDraw()) {
            cout << "It's a draw!\n";
            break;
        }
    }

    return 0;
}
```

ΣΥΜΠΕΡΑΣΜΑΤΑ & LESSONS LEARNED

Τι μάθατε από το project;

1)Μάθαμε να προγραμματίζουμε σε περιβάλλον Visual Code

2)Εξασκήσαμε τις  γνώσεις που μάθαμε στην θεωρία

3)Υλοποίηση του Project μας με κατάλληλη χρήση γραφικών

Τι θα μπορούσε να βελτιωθεί;

1)Εγκατάσταση ηχητικών εφέ

2)Σε περίπτωση ισοπαλίας ο πίνακας θα επεκτείνεται μέχρι να βρεθεί ο νικητής.

- Μελλοντικές προεκτάσεις του project

Μετατροπή από μοναδικό παιχνίδι σε mini game με την χρήση launcher.

- LAUNCHER

Ο όρος launcher αναφέρεται γενικά σε ένα πρόγραμμα ή εργαλείο που χρησιμοποιείται για να εκκινήσει άλλες εφαρμογές ή διαδικασίες σε ένα σύστημα υπολογιστή ή σε μια συσκευή.