Politechnika Świętokrzyska w Kielcach Wydział Elektroniki, Automatyki i Informatyki	
Programowanie Obiektowe - Projekt Informatyka - I rok, II semestr, Rok akademicki - 2023/2024	
Temat projektu: **Gra kółko i krzyżyk**	Wykonali: **Adrian Borzęcki** **Bartłomiej Baczyński** **Piotr Kozłowski** **Adam Kozuń** Grupa: **1ID11A**

Realizacja projektu gry pt. “Kółko i krzyżyk”

Podział pracy w zespole i procentowy udział

I) Adrian Borzęcki **(27%)**
- Stworzenie ogólnej logiki gry (mechaniki gry, grafik na planszy i logiki bota, możliwość przegranej, wygranej i remisu),
- implementacja kodu minmax do działania bota,
- działanie wczytywania i wypisywania wyników i pseudonimów graczy oraz ich czasu gry,
- stworzenie sprawozdania.

II) Bartłomiej Baczyński **(27%)**
- Wspólne zaprojektowanie grafik,
- pozbycie się niepotrzebnych rozwiązań w kodzie,
- działanie responsywności myszki (interfejs UI, ogólna komunikacja z graczem za pomocą myszy i klawiatury),
- stworzenie sprawozdania.

III) Piotr Kozłowski **(27%)**
- Stworzenie ogólnej logiki gry (mechaniki gry, grafik na planszy i logiki bota, możliwość przegranej, wygranej i

remisu),
- poprawienie kodu minmax do działania bota (konkretnie poprawienie działania funkcji sztucznaInteligencja i ruchKomputera),
- wyświetlanie czasu gry,
- stworzenie sprawozdania.

IV) Adam Kozuń **(19%)**
- Wspólne zaprojektowanie grafik,
- poprawienie optymalizacji kodu,
- pomoc przy inicjalizacji myszki i jej responsywności,
- stworzenie sprawozdania.

Spis treści

a) cel i zakres koncepcji projektu
b) przegląd innych podobnych rozwiązań
c) specyfikacja Projektowa zawierająca: opis funkcjonalności oraz uzasadnienie wyboru konkretnych technologii
d) techniczna implementacja (opis listingu)
e) wnioski zawierające podsumowanie wykonania działań i możliwości rozwoju aplikacji w przyszłości
f) oświadczenie o samodzielności projektu

Biblioteki:

- #include <iostream>
- #include <vector>
- #include <deque>
- #include <fstream>
- #include <string>
- #include <cstdlib>
- #include <ctime>
- #include <allegro5/allegro.h>
- #include <allegro5/allegro_primitives.h>
- #include <allegro5/allegro_font.h>
- #include <allegro5/allegro_ttf.h>
- #include <allegro5/allegro_native_dialog.h>
- #include <allegro5/allegro_image.h>

Funkcje:

- **void inicjalizuj**(vector<Pole>& plansza, vector<Kolizja>& kolizje)

- **void wyswietl**(vector<Pole>& plansza)
- **bool planszaPelna**(const vector<Pole>& plansza)
- **bool sprawdzWygrana**(char gracz, const vector<Pole>& plansza)
- **int sztucznaInteligencja**(char gracz, vector<Pole>& plansza, bool OptymalnyRuch)
- **void ruchKomputera**(vector<Pole>& plansza)
- **void sprawdzKolizje**(vector<Kolizja> kolizje, vector<Pole>& plansza, int mouseX, int mouseY)
- bool okno()
- **void zapiszWynik**(const string& wynik, const string& nick, const double& czas)
- **deque<string>** czytajOstatnieWyniki(const string& filePath, int n)
- **void wyswietlWyniki**(const deque<string>& wyniki)
- **string getNickname()**
- **int main()**

Metody klasy kolizja:

- konstruktor - **Kolizja()**
- konstruktor z parametrami - **Kolizja(int _x, int _y, int _size)**

Legenda:

- Opisy tego koloru to komentarze do kodu

Funkcjonalność projektu

a) cel i zakres koncepcji projektu:

Projekt gry pt. “kółko i krzyżyk” dotyczył utworzenia gry strategicznej, w której to celem gracza jest pokonać przeciwnika, w tym przypadku sztuczną inteligencje (bota).
Gra odbywa się na polu o wymiarach 3x3, a w ogólnym założeniu gry, Gracze obejmują pola na przemian dążąc do objęcia trzech pól w jednej linii, przy jednoczesnym uniemożliwieniu tego samego przeciwnikowi. Pole może być objęte przez jednego gracza i nie zmienia swego właściciela przez cały przebieg gry.
Wynik gracza zapisywany jest w postaci podawanego przez gracza pseudonimu gry, wtedy zapisany plik, obejmuje:

- wpisany pseudonim gracza (nick),
- czy gracz wygrał, przegrał bądź zremisował z sztuczną inteligencją,
- czas gry, jaki gracz poświęcił od początku do końca rozgrywki.

Wynik gracza zostaje zapisany w pliku pt. “wyniki.txt”, a następnie jest on uwidoczniony w końcowym, nowym oknie gry (400x400), a następnie program kończy swoje działanie po 5 sekundach.

b) przegląd innych podobnych rozwiązań

W zapoznaniu z działaniem innych gier typu “kółko i krzyżyk”, doszliśmy do wniosku, że gry tej kategorii cechują się podobnym, a wręcz identycznym działaniem gry. Jednak w celu stworzenia podobnej gry, używane były:

- inne biblioteki, gdzie w naszym przypadku została użyta biblioteka allegro5,
- inne koncepcje działania sztucznej inteligencji gry, tutaj użyty przez nas został algorytm minmax,
- w niektórych przypadkach użyty nawet został silnik, przykładowo unity, do stworzenia podobnej gry.

Ogólny i krótki opis algorytmu minmax:

Algorytm minimax to metoda stosowana w teorii gier i sztucznej inteligencji do optymalizacji decyzji w grach o sumie zerowej, takich jak szachy czy kółko i krzyżyk. Polega na minimalizacji maksymalnej możliwej straty (minimalizacja maksimum) lub maksymalizacji minimalnej możliwej wygranej (maksymalizacja minimum).

Czemu użyliśmy biblioteki allegro5:

W celu realizacji projektu, użyta została przez nas podana biblioteka, ponieważ już w poprzednim projekcie związanym z projektem gry “Gra Snake” i “Gra Space Invaders”, używaliśmy biblioteki allegro, przez co znaliśmy już od samego początku tworzenia projektu daną bibliotekę i nie mieliśmy problemów z implementacją rzeczy takich jak mysz, klawiatura czy okna gry i grafiki.

c) specyfikacja Projektowa zawierająca: opis funkcjonalności oraz uzasadnienie wyboru konkretnych technologii

Opis funkcjonalności znajduje się niżej, jednak co do wyboru konkretnych technologii:

Ogólny i krótki opis algorytmu minmax:
Algorytm minimax to metoda stosowana w teorii gier i sztucznej inteligencji do optymalizacji decyzji w grach o sumie zerowej, takich jak szachy czy kółko i krzyżyk. Polega na minimalizacji maksymalnej możliwej straty (minimalizacja maksimum) lub maksymalizacji minimalnej możliwej wygranej (maksymalizacja minimum).

Czemu użyliśmy biblioteki allegro5:
W celu realizacji projektu, użyta została przez nas podana biblioteka, ponieważ już w poprzednim projekcie związanym z projektem gry “Gra Snake” i “Gra Space Invaders”, używaliśmy biblioteki allegro, przez co znaliśmy już od samego początku tworzenia projektu daną bibliotekę i nie mieliśmy problemów z implementacją rzeczy takich jak mysz, klawiatura czy okna gry i grafiki.

d) Techniczna implementacja (opis listingu):

```
/// Standardowe operacje wejścia/wyjścia
#include <iostream>
/// Dynamiczne tablice do przechowywania planszy
#include <vector>
/// Przechowywanie ostatnich wyników z pliku (deque<string>)
#include <deque>
/// Operacje na plikach (czytanie i zapisywanie wyników)
#include <fstream>
/// Operacje na łańcuchach znaków(std::string itp.)
#include <string>
/// Funkcje standardowej biblioteki, np. rand() i srand()
#include <cstdlib>
/// Inicjalizacja generatora losowego srand(time(NULL))
#include <ctime>
/// Podstawowe funkcje Allegro 5
#include <allegro5/allegro.h>
/// Rysowanie prymitywów (linii do siatki, była używana
wcześniej)
#include <allegro5/allegro_primitives.h>
/// Podstawowe funkcje czcionek Allegro 5
#include <allegro5/allegro_font.h>
/// Wczytywanie czcionek TrueType
#include <allegro5/allegro_ttf.h>
/// Pokazywanie natywnych okien dialogowych (komunikaty o
błędach)
#include <allegro5/allegro_native_dialog.h>
/// Ładowanie i operacje na obrazach
#include <allegro5/allegro_image.h>

using namespace std;
```

```
/// Definiuje wymiary okna gry (szerokość, x)
const int szerokosc = 15;
/// Definiuje wymiary okna gry (wysokość, y)
const int wysokosc = 15;
/// Definiuje położenie planszy gry (położenie wzgledem osi x)
const int szerokosc_s = 3;
/// Definiuje położenie planszy gry (położenie względem osi y)
const int wysokosc_s = 3;
/// Definiuje rozmiar okna gry (pod względem ogólnym)
const int rozmiar = 60;
/// Definiuje rozmiar gry pod względem generowania kółka i
krzyżyka na planszy
const int rozmiar2 = 135;
/// Pomaga w dokładnym ustawieniu położenia responsywności
myszki (jej poprawnej responsywności względem planszy)
const int xoffset = 41;

/// Flaga wskazująca, czy gra się zakończyła (true jeżeli tak)
bool koniecGry = false;
/// Wskaźnik do kolejki zdarzeń Allegro, kolejka zdarzeń jest
używana do przechwytywania i obsługi zdarzeń takich jak
naciśnięcia klawiszy myszy, czy klawiatury
ALLEGRO_EVENT_QUEUE* queue = NULL;
/// Wskaźnik do wyświetlania okna w Allegro, używane jest do
określania położenia np. stopera gry względem okna
ALLEGRO_DISPLAY* display = NULL;
/// Wskaźnik do bitmapy tła gry, używane do wyświetlania
tlo.png lub tlo2.png, zależnie od tego które tło zostało
wylosowane
ALLEGRO_BITMAP* bitmap;
/// Wskaźnik do bitmapy krzyżyka (X), obraz reprezentujący
ruch osoby, która gra w grę
ALLEGRO_BITMAP* cross;
```

```
/// Wskaźnik do bitmapy kółka (O), w skrócie obraz reprezentujący ruch bota
ALLEGRO_BITMAP* circle;
/// Wskaźnik do bitmapy ekranu końca gry, jest to obraz który jest wyświetlany przy zwycięstwie, przegranej lub remisie gracza
ALLEGRO_BITMAP* gameOverImage;
/// Wskaźnik do czcionki Allegro, czcionka jest używana do rysowania tekstu na ekranie
ALLEGRO_FONT* font = NULL;
/// Wskaźnik do timera Allegro, timer jest używany do pokazania czasu gry gracza
ALLEGRO_TIMER* timer = NULL;
/// Używane do odliczania czasu który upływa w sekundach
double elapsedTime = 0;
/// Flaga wskazująca, czy stoper ma być uruchomiony (odliczanie czasu)
bool startTimer = false;

/// Deklaracja klasy, używana jest do wykrywania, czy kliknięcie myszy znajduje się wewnątrz określonego obszaru (planszy gry). Używana jest do sprawdzania kliknięć na różnych polach planszy gry (planszy 3x3)

class Kolizja
{
public:
/// Inicjalizuje pola x, y i size i ustawia ich domyślną wartość na 0
    Kolizja()
    {
        x = 0;
        y = 0;
```

```
        size = 0;
    }

    /// Inicjalizuje pola x, y i size, przypisuje te wartości
odpowiednim polom
    Kolizja(int _x, int _y, int _size) : x(_x), y(_y), size(_size) {}
    bool isMouseInside(int mouseX, int mouseY);
private:
    int x;
    int y;
    int size;

};

bool Kolizja::isMouseInside(int mouseX, int mouseY)
{
    /// Jeśli współrzędna x myszy jest mniejsza niż x kwadratu, to
punkt znajduje się na lewo od kwadratu i zwraca false
    if (mouseX < x) return false;
    /// Jeśli współrzędna x myszy jest większa niż prawa
krawędź kwadratu (x + size), to punkt znajduje się na prawo od
kwadratu i zwraca false
    if (mouseX > (x + size)) return false;
    /// Jeśli współrzędna y myszy jest mniejsza niż y kwadratu, to
punkt znajduje się powyżej kwadratu i zwraca false
    if (mouseY < y) return false;
    /// Jeśli współrzędna y myszy jest większa niż dolna krawędź
kwadratu (y + size), to punkt znajduje się poniżej kwadratu i
zwraca false
    if (mouseY > (y + size)) return false;

    /// Jeśli żadna z powyższych sytuacji nie zachodzi, to mysz
znajduje się wewnątrz kwadratu i zwracana jest wartość true
```

```
    return true;
}

struct Pole {
    /// Przechowuje wartość pola, może być ' ', 'X', lub 'O'
    char wartosc;
    /// Inicjuje pole wartosc na ' ', pole jest początkowo puste
    Pole() : wartosc(' ') {}
};

/// Inicjalizuje planszę oraz obiekty kolizji
///
/// Inicjalizuje timery, kolejkę zdarzeń oraz czcionki.
/// Tworzy obiekty kolizji na planszy
/// Plansza - wektor obiektów typu Pole, reprezentujących
planszę
/// Kolizje - wektor obiektów typu Kolizja, reprezentujących
obszary kolizji
void inicjalizuj(vector<Pole>& plansza, vector<Kolizja>&
kolizje) {
    plansza.assign(9, Pole());

    /// Inicjalizacja timera
    ///
   /// Ustawia timer na 60 FPS
    timer = al_create_timer(1.0);



    /// Tworzenie kolejki zdarzeń
    queue = al_create_event_queue();
    /// Inicjalizacja komunikatów błędów
    ///
```

```
    /// Wyświetlenie komunikatów błędów w przypadku
niepowodzenia inicjalizacji konkretnych rzeczy w kodzie
    if (!queue) {
        al_show_native_message_box(NULL, NULL, NULL,
"Failed to create event queue!", NULL, NULL);
        al_destroy_display(display);
        return;
    }
    if (!al_init_font_addon()) {
        cout << "Could not init the font addon.\n";
    }
    if (!al_init_ttf_addon()) {
        cout << "Could not init the ttf addon.\n";
    }
    font = al_load_ttf_font("C:\\Windows\\Fonts\\Arial.ttf", 100, 0);
    if (!font) {
        cout << "Failed to load font!" << endl;
        return;
    }

    //Rejestracja źródeł zdarzeń w kolejce
    al_register_event_source(queue,
al_get_display_event_source(display));
    al_register_event_source(queue,
al_get_keyboard_event_source());
    al_register_event_source(queue,
al_get_mouse_event_source());

    ///Poprzedni kod odpowiedzialny za działanie timera
    // Inicjalizacja timera (60 klatek na sekundę)
  //  timer = al_create_timer(1.0 / 60);
//    if (!timer) {
//        cerr << "Failed to create timer!" << endl;
```

```
//        return;
 //    }
    al_register_event_source(queue,
al_get_timer_event_source(timer));

    /// Inicjalizacja planszy i kolizji
    int planszaX = (al_get_display_width(display) - szerokosc_s
* rozmiar2) / 2 - xoffset;
    int planszaY = (al_get_display_height(display) - wysokosc_s
* rozmiar2) / 2;

    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            kolizje.push_back(Kolizja(planszaX + j * rozmiar2,
planszaY + i * rozmiar2, rozmiar2));
        }
    }

    /// Rozpoczęcie timera
    al_start_timer(timer);
    al_get_timer_count(timer);
}

/// Wyświetla planszę oraz znaki na ekranie
///
/// Wyświetla planszę, siatkę, znaki (X i O) oraz stoper.
/// Plansza - wektor obiektów typu Pole, reprezentujących
planszę
void wyswietl(vector<Pole>& plansza) {
    /// Czyści ekran, ustawiając kolor tła na czarny
    al_clear_to_color(al_map_rgb(0, 0, 0));

    ///Ustawiało kolor siatki na biały (gdy było używane)
```

```
    ALLEGRO_COLOR color2 = al_map_rgb(255, 255, 255);

    // Oblicz współrzędne początkowe dla planszy, aby była na środku ekranu
    int planszaX = (al_get_display_width(display) - szerokosc_s * rozmiar2) / 2;
    int planszaY = (al_get_display_height(display) - wysokosc_s * rozmiar2) / 2;

///Rysowanie siatki, używane wcześniej gdy nie mieliśmy grafik
    /*for (int i = 0; i < szerokosc_s + 1; i++) {
        al_draw_line(planszaX + i * rozmiar2, planszaY, planszaX + i * rozmiar2, planszaY + wysokosc_s * rozmiar2, color2, 3);
    }
    for (int i = 0; i < wysokosc_s + 1; i++) {
        al_draw_line(planszaX, planszaY + i * rozmiar2, planszaX + szerokosc_s * rozmiar2, planszaY + i * rozmiar2, color2, 3);
    }*/


    al_draw_bitmap(bitmap, 0, 0, 0); // Rysowanie bitmapy tła



    // Rysowanie znaków na planszy
    for (int i = 0; i < 9; i++)
    {
        int x = planszaX + (i % 3) * rozmiar2 + rozmiar2 / 2;
        int y = planszaY + (i / 3) * rozmiar2 + rozmiar2 / 2;

        if (plansza[i].wartosc == 'X')
        {
```

```
            //70 to offset w pikselach, moze sie zmieniac w
zaleznosci od rozmiaru zdjecia, aktualna tekstura miała rozmiar
140px x 140px, ale już nie ma :P
// Rysowanie krzyżyka na bitmapie
            al_draw_bitmap(cross, x - 112, y - 60, 0);
        }
        else if (plansza[i].wartosc == 'O')
        {
// Rysowanie kółka na bitmapie
            al_draw_bitmap(circle, x - 112, y - 60, 0);
        }
    }

    // Rysowanie stopera
    int stoperX = al_get_display_width(display) - 250;
    int stoperY = al_get_display_height(display) - 100;
    al_draw_textf(font, al_map_rgb(255, 255, 255), stoperX,
stoperY, ALLEGRO_ALIGN_RIGHT, "Czas: %.0f s",
elapsedTime);

    /*
    for (int i = 0; i < 9; ++i) {
        int x = planszaX + (i % 3) * rozmiar2 + rozmiar2 / 2;
        int y = planszaY + (i / 3) * rozmiar2 + rozmiar2 / 2;
        al_draw_textf(font, al_map_rgb(255, 255, 255), x, y - 55,
ALLEGRO_ALIGN_CENTER, "%c", plansza[i].wartosc);
    }
    */


    al_flip_display();
    //al_destroy_font(font);
}
```

```
/// Sprawdza, czy plansza jest pełna
///
/// Przechodzi przez wszystkie pola planszy i sprawdza, czy są puste.
/// plansza - wektor obiektów typu Pole, reprezentujących planszę.
/// return true, jeśli plansza jest pełna, w przeciwnym razie false
bool planszaPelna(const vector<Pole>& plansza) {
    for (int i = 0; i < 9; ++i) {
/// Sprawdzenie, czy pole jest puste
        if (plansza[i].wartosc == ' ') {
/// Jeśli jest puste, plansza nie jest pełna
            return false;
        }
    }
/// Jeśli żadne pole nie jest puste, plansza jest pełna
    return true;
}

/// Sprawdza, czy dany gracz wygrał
///
/// Sprawdza wszystkie możliwe linie wygrywające na planszy
/// gracz - znak gracza ('X' lub 'O')
/// plansza - wektor obiektów typu Pole, reprezentujących planszę
/// return true, jeśli gracz wygrał, w przeciwnym razie false
bool sprawdzWygrana(char gracz, const vector<Pole>& plansza) {
/// Sprawdzenie wygranej w wierszach
    for (int i = 0; i < 9; i += 3) {
```

```
        if (plansza[i].wartosc == gracz && plansza[i + 1].wartosc
== gracz && plansza[i + 2].wartosc == gracz) {
            return true;
        }
    }
/// Sprawdzenie wygranej w kolumnach
    for (int i = 0; i < 3; ++i) {
        if (plansza[i].wartosc == gracz && plansza[i + 3].wartosc
== gracz && plansza[i + 6].wartosc == gracz) {
            return true;
        }
    }
/// Sprawdzenie wygranej na przekątnych
    if ((plansza[0].wartosc == gracz && plansza[4].wartosc ==
gracz && plansza[8].wartosc == gracz) ||
        (plansza[2].wartosc == gracz && plansza[4].wartosc ==
gracz && plansza[6].wartosc == gracz)) {
        return true;
    }
/// Brak wygranej
    return false;
}

int sztucznaInteligencja(char gracz, vector<Pole>& plansza,
bool OptymalnyRuch) {
/// Sprawdzenie, czy gracz 'X' wygrał
    if (sprawdzWygrana('X', plansza)) return -1;
/// Sprawdzenie, czy gracz 'O' wygrał
    if (sprawdzWygrana('O', plansza)) return 1;
/// Sprawdzenie, czy plansza jest pełna
    if (planszaPelna(plansza)) return 0;

    int najlepszyRuch = -1;
```

```
    int najlepszaWartosc = (gracz == 'O') ? -10 - 1 : 10;

    for (int i = 0; i < 9; ++i) {
/// Jeśli pole jest puste, wykonaj ruch
        if (plansza[i].wartosc == ' ') {
            plansza[i].wartosc = gracz;
            int wartoscRuchu = sztucznaInteligencja((gracz == 'X')
? 'O' : 'X', plansza, OptymalnyRuch);
/// Wykonuje najlepszą wartość i ruch na podstawie gracza
            if ((gracz == 'O' && wartoscRuchu > najlepszaWartosc)
|| (gracz == 'X' && wartoscRuchu < najlepszaWartosc)) {
                najlepszaWartosc = wartoscRuchu;
                najlepszyRuch = i;
            }
            plansza[i].wartosc = ' ';
        }
    }
/// Jeśli brak ruchu, zwraca 0
    if (najlepszyRuch == -1) return 0;
    return (gracz == 'O') ? najlepszaWartosc : najlepszaWartosc;
}


void ruchKomputera(vector<Pole>& plansza) {
    int najlepszyRuch = -1;
    int najlepszaWartosc = -10 - 1;

/// 2/3 szansy na optymalny ruch
    bool OptymalnyRuch = (rand() % 3 != 0);

    for (int i = 0; i < 9; ++i) {
/// Jeśli pole jest puste, wykonuje ruch
        if (plansza[i].wartosc == ' ') {
```

```
            plansza[i].wartosc = 'O';
/// Wywołuje funkcję
            int wartoscRuchu = sztucznaInteligencja('X', plansza,
OptymalnyRuch);
            plansza[i].wartosc = ' ';

// Najlepsza wartość i ruch, jeśli wartość ruchu jest większa
            if (wartoscRuchu > najlepszaWartosc) {
                najlepszaWartosc = wartoscRuchu;
                najlepszyRuch = i;
            }
        }
    }

    if (!OptymalnyRuch && najlepszyRuch != -1) {
        /// Wybiera losowo zamiast tego, by od razu podjął
najlepszy ruch
        vector<int> MozliweRuchy;
        for (int i = 0; i < 9; ++i) {
            if (plansza[i].wartosc == ' ') {
                MozliweRuchy.push_back(i);
            }
        }
/// Wybiera losowy ruch spośród dostępnych
        najlepszyRuch = MozliweRuchy[rand() %
MozliweRuchy.size()];
    }
// Komputer wykonuje najlepszy (lub losowy) ruch na planszy
    plansza[najlepszyRuch].wartosc = 'O';
}

void sprawdzKolizje(vector<Kolizja> kolizje, vector<Pole>&
plansza, int mouseX, int mouseY)
```

```
{
/// przejście przez wszystkie obiekty kolizji
    for (int i = 0; i < kolizje.size(); i++)
    {
/// Sprawdzenie, czy myszka znajduje się w obszarze kolizji
        if (kolizje[i].isMouseInside(mouseX, mouseY))
        {
/// Jeśli pole jest puste, gracz umieszcza znak 'X'
            if (plansza[i].wartosc == ' ') {
                plansza[i].wartosc = 'X';
/// Sprawdza, czy gracz 'X' wygrał
                if (sprawdzWygrana('X', plansza)) {
/// Wyświetlenie planszy
                    wyswietl(plansza);
                    koniecGry = true;
                }
/// Jeśli plansza nie jest pełna, komputer wykonuje ruch
                else if (!planszaPelna(plansza)) {
/// Ruch komputera
                    ruchKomputera(plansza);
/// Sprawdzenie, czy komputer 'O' wygrał
                    if (sprawdzWygrana('O', plansza)) {
/// Wyświetlenie planszy
                        wyswietl(plansza);
                        koniecGry = true;
                    }
                }
/// Jeśli plansza jest pełna, gra kończy się remisem
                else {
                    koniecGry = true;
                }
            }
            break;
```

```
        }
    }
}

bool okno() {
/// Inicjalizacja biblioteki Allegro 5
    if (!al_init()) {
        al_show_native_message_box(NULL, NULL, NULL,
"Failed to initialize Allegro 5", NULL, NULL);
        return false;
    }
/// Inicjalizacja natywnych dialogów
    if (!al_init_native_dialog_addon()) {
        cout << "Could not init the native dialog addon.\n";
        return false;
    }
/// Inicjalizacja prymitywów graficznych
    if (!al_init_primitives_addon()) {
        cout << "Could not init the primitives addon.\n";
        return false;
    }
/// Inicjalizacja obrazów
    if (!al_init_image_addon())
    {
        cout << "Could not init the image addon.\n";
        return false;
    }
/// Inicjalizacja klawiatury
    if (!al_install_keyboard()) {
        al_show_native_message_box(NULL, NULL, NULL,
"Failed to initialize the keyboard!", NULL, NULL);
        return false;
    }
```

```
/// Inicjalizacja myszy
    if (!al_install_mouse()) {
        al_show_native_message_box(NULL, NULL, NULL,
"Failed to initialize the mouse!", NULL, NULL);
        return false;
    }
/// Tworzenie displayu o określonych rozmiarach
    display = al_create_display(szerokosc * rozmiar, wysokosc *
rozmiar);
    if (!display) {
        al_show_native_message_box(NULL, NULL, NULL,
"Failed to create Allegro 5 display", NULL, NULL);
        return false;
    }
/// Tworzenie kolejki zdarzeń
    queue = al_create_event_queue();
    if (!queue) {
        al_show_native_message_box(NULL, NULL, NULL,
"Failed to create event queue!", NULL, NULL);
        al_destroy_display(display);
        return false;
    }

/// Rejestracja źródeł zdarzeń dla display, klawiatury i myszy
    al_register_event_source(queue,
al_get_display_event_source(display));
    al_register_event_source(queue,
al_get_keyboard_event_source());
    al_register_event_source(queue,
al_get_mouse_event_source());
    return true;
}
```

```
void zapiszWynik(const string& wynik, const string& nick,
const double& czas) {
/// Otwarcie pliku do zapisu w trybie dopisywania
    ofstream file("wyniki.txt", ios::app);
    if (file.is_open()) {
/// Zapis wyniku w formacie: nick: wynik: czas s
        file << nick << ": " << wynik << ": " << czas << " s" << endl;
        file.close(); // Zamknięcie pliku
    }
    else {
        cout << "Nie można otworzyć pliku do zapisu!" << endl;
    }
}

deque<string> czytajOstatnieWyniki(const string& filePath, int
n) {
/// Kolejka do przechowywania linii wyników
    deque<string> lines;
/// Otwarcie pliku do odczytu
    ifstream file(filePath);
    string line;
    if (file.is_open()) {
/// Odczyt linii z pliku
        while (getline(file, line)) {
            if (lines.size() == n) {
/// Usunięcie najstarszej linii
                lines.pop_front();
            }
/// Dodanie nowej linii do kolejki
            lines.push_back(line);
        }
/// Zamknięcie pliku
        file.close();
```

```
    }
    else {
        cerr << "Nie można otworzyć pliku do odczytu!" << endl;
    }
/// Zwrócenie wyników
    return lines;
}

void wyswietlWyniki(const deque<string>& wyniki)
{
/// Tworzenie nowego displaya o rozmiarach 400x400 pikseli
    ALLEGRO_DISPLAY* resultsDisplay =
al_create_display(400, 400);
/// Wyświetlanie komunikatu w przypadku niepowodzenia
tworzenia wyświetlacza
    if (!resultsDisplay) {
        al_show_native_message_box(NULL, NULL, NULL, "Nie
udało się utworzyć wyświetlacza wyników", NULL, NULL);
        return;
    }

/// Inicjalizacja dodatków fontów i ttf
    if (!al_init_font_addon() || !al_init_ttf_addon()) {
        cout << "Nie udało się zainicjować dodatków fontów lub
ttf." << endl;
        return;
    }
/// Ładowanie fontu z pliku TTF
    ALLEGRO_FONT* font =
al_load_ttf_font("C:\\Windows\\Fonts\\Arial.ttf", 16, 0);
    if (!font) {
        cout << "Nie udało się załadować fontu!" << endl;
        return;
```

```
    }
/// Inicjalizacja pozycji Y dla pierwszego wyniku
    int y = 50;
/// Rysowanie tekstu wyniku na ekranie, wycentrowanego na
środku ekranu
    for (const auto& wynik : wyniki) {
        al_draw_text(font, al_map_rgb(255, 255, 255), 200, y,
ALLEGRO_ALIGN_CENTER, wynik.c_str());
/// Przesunięcie pozycji Y dla kolejnego wyniku
        y += 30;
    }
/// Odświeżenie wyświetlacza
    al_flip_display();
    al_rest(5.0);

/// Zwolnienie fontu
    al_destroy_font(font);
/// Zniszczenie displaya
    al_destroy_display(resultsDisplay);
}

string getNickname() {
/// Zmienna do przechowywania wprowadzonego nicku
    string nickname;
/// Zmienna do przechowywania zdarzeń
    ALLEGRO_EVENT event;
/// Ładowanie fontu
    ALLEGRO_FONT* font =
al_load_ttf_font("C:\\Windows\\Fonts\\Arial.ttf", 32, 0);

/// Czyszczenie ekranu na czarny kolor
    al_clear_to_color(al_map_rgb(0, 0, 0));
// Wyświetlanie nicku wprowadzonego przez gracza
```

```
    al_draw_text(font, al_map_rgb(255, 255, 255), 400, 300,
ALLEGRO_ALIGN_CENTER, "Enter your nickname:");
/// Odświeżenie displaya
    al_flip_display();

/// Oczekiwanie na zdarzenie
    while (true) {
        al_wait_for_event(queue, &event);

/// Sprawdzenie, czy naciśnięto klawisz
        if (event.type == ALLEGRO_EVENT_KEY_CHAR)
 {
/// Pozwala na wprowadzenie klawisza klawiatury
            if (event.keyboard.unichar == '\r') {
                break;
            }
/// Backspace
            else if (event.keyboard.unichar == '\b') {
                if (!nickname.empty()) {
/// Usunięcie ostatniego znaku w nicku
                    nickname.pop_back();
                }
            }
            else {
/// Dodanie znaku do nicku
                nickname += event.keyboard.unichar;
            }
        }
/// Ponowne czyszczenie ekranu i wyświetlanie
zaktualizowanego nicku
        al_clear_to_color(al_map_rgb(0, 0, 0));
/// Wywołanie tła “gameOverImage”
        al_draw_bitmap(gameOverImage, 0, 0, 0);
```

```
        al_draw_text(font, al_map_rgb(160, 82, 45), 450, 325,
ALLEGRO_ALIGN_CENTER, nickname.c_str());
/// Odświeżenie displayu
        al_flip_display();
    }

/// Zwalnianie czcionki
    al_destroy_font(font);
/// Zwracanie nicku gracza
    return nickname;
}

/// Główna funkcja programu
///
/// Tworzy okno gry, inicjalizuje planszę, obsługuje zdarzenia i
logikę gry
/// return Kod wyjścia programu.
int main() {
/// Inicjalizacja generowania liczb pseudolosowych
    srand(time(NULL));
/// Inicjalizacja okna gry, jeśli się nie powiedzie, zwraca -1
    if (!okno()) {
        return -1;
    }

/// Zmienna przechowująca współrzędną X kursora myszy
    int mouseX = 0;
/// Zmienna przechowująca współrzędną Y kursora myszy
    int mouseY = 0;

/// Inicjalizacja wektora pól planszy z 9 elementami
    vector<Pole> plansza(9);
/// Inicjalizacja wektora kolizji
```

```
    vector<Kolizja> kolizje;
/// Wywołanie funkcji inicjalizującej planszę i kolizje
    inicjalizuj(plansza, kolizje);

/// Losowanie liczby (tła) 0 lub 1
    int los = rand() % 2;

    if (los == 1)
    {
/// Ładowanie tła "tlo2.png" w przypadku wyniku 1
        bitmap = al_load_bitmap("..//grafiki//tlo2.png");
    }
    else
    {
/// Ładowanie tła "tlo.png" w przypadku wyniku 0
        bitmap = al_load_bitmap("..//grafiki//tlo.png");
    }

/// Ładowanie grafiki krzyżyka
    cross = al_load_bitmap("..//grafiki//krzyzyk.png");
// Ładowanie grafiki kółka
    circle = al_load_bitmap("..//grafiki//kolko.png");
/// Ładowanie grafiki końca gry
    gameOverImage =
al_load_bitmap("..//grafiki//gameover.png");
/// Wywołanie funkcji wyświetlającej planszę
    wyswietl(plansza);
/// Główna pętla gry, która działa dopóki gra się nie skończy
    while (!koniecGry)
    {
        ALLEGRO_EVENT ev;
        while (!koniecGry) {
/// Oczekiwanie na zdarzenie
```

```
        al_wait_for_event(queue, &ev);

/// Sprawdzenie, czy użytkownik zamknął okno gry
        if (ev.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
            koniecGry = true;
        }
        else if (ev.type == ALLEGRO_EVENT_TIMER) {
/// Zwiększenie czasu o 1 sekundę
            elapsedTime += 1.0;
        }
/// Sprawdzenie ruchu kursora myszy
        else if (ev.type == ALLEGRO_EVENT_MOUSE_AXES)
 {
            mouseX = ev.mouse.x;
            mouseY = ev.mouse.y;
        }
/// Sprawdzenie kolizji i aktualizacja planszy
        else if (ev.type ==
ALLEGRO_EVENT_MOUSE_BUTTON_DOWN) {
            sprawdzKolizje(kolizje, plansza, mouseX, mouseY);
        }
        else if (ev.type == ALLEGRO_EVENT_KEY_DOWN) {
            if (ev.keyboard.keycode >= ALLEGRO_KEY_1 &&
ev.keyboard.keycode <= ALLEGRO_KEY_9) {
                int pole = ev.keyboard.keycode -
ALLEGRO_KEY_1;
            }
        }
/// Wyświetlenie zaktualizowanej planszy
        wyswietl(plansza);
    }
  }
```

```
/// Pobranie nazwy użytkownika
    string nick = getNickname();

/// Sprawdzenie, kto wygrał i zapisanie wyniku
    if (sprawdzWygrana('X', plansza)) {
        zapiszWynik("Wygrana gracza", nick, elapsedTime);
    }
    else if (sprawdzWygrana('O', plansza)) {
        zapiszWynik("Wygrana komputera", nick, elapsedTime);
    }
    else {
        zapiszWynik("Remis", nick, elapsedTime);
    }

/// Odczytanie 10 ostatnich wyników z pliku
    auto wyniki = czytajOstatnieWyniki("wyniki.txt", 10);
/// Wyświetlenie wyników
    wyswietlWyniki(wyniki);

/// Zwalnianie wyświetlacza
    al_destroy_display(display);
/// Zwalnianie kolejki zdarzeń
    al_destroy_event_queue(queue);
    return 0;
}
```

e) Wnioski zawierające podsumowanie wykonania działań i możliwości rozwoju aplikacji w przyszłości

Czego nie udało się zrealizować:
Wszystkie nasze cele realizacji rozwoju projektu zostały osiągnięte, wszystko udało się zrealizować

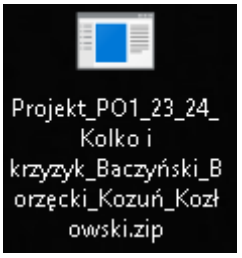
Co można poprawić:
Istnieje możliwość poprawy logiki sztucznej inteligencji, by działała on lepiej

Możliwe kierunki rozwoju projektu:
Przykładowym kierunkiem rozwoju projektu, byłoby dodanie możliwości loopowania gry (ciągłego grania). Ewentualnie, dalszą możliwością rozwoju projektu, byłoby dodanie większej ilości losowych grafik i wyświetlanie poprzednich wyników przy starcie gry.

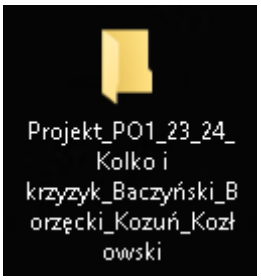
f) Załączniki projektowe (instrukcja instalacji obsługi aplikacji)

Opis funkcjonalności projektu i instrukcja kompilacji/uruchomienia oraz wyniki testów:

1. Wypakuj zip z gotowym projektem (nazwa zipa “Projekt_PO1_23_24_ Kolko i krzyzyk_Baczyński_Borzęcki_Kozuń_Kozłowski.zip”) **do dysku C**, jest to konieczne dla pewnego zadziałania gry,



2. Aby uruchomić grę, należy wejść do folderu o nazwie “Projekt_PO1_23_24_ Kolko i krzyzyk_Baczyński_Borzęcki_Kozuń_Kozłowski”,

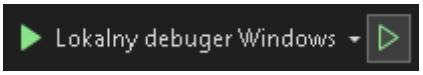


3. Następnie należy przejść do folderu “część praktyczna”,

4. Trzeba wejść następnie do folderu “kolko_krzyzyk_game”,

5. z kolei potem, należy uruchomić plik o nazwie “kolko_krzyzyk_game.sln”,

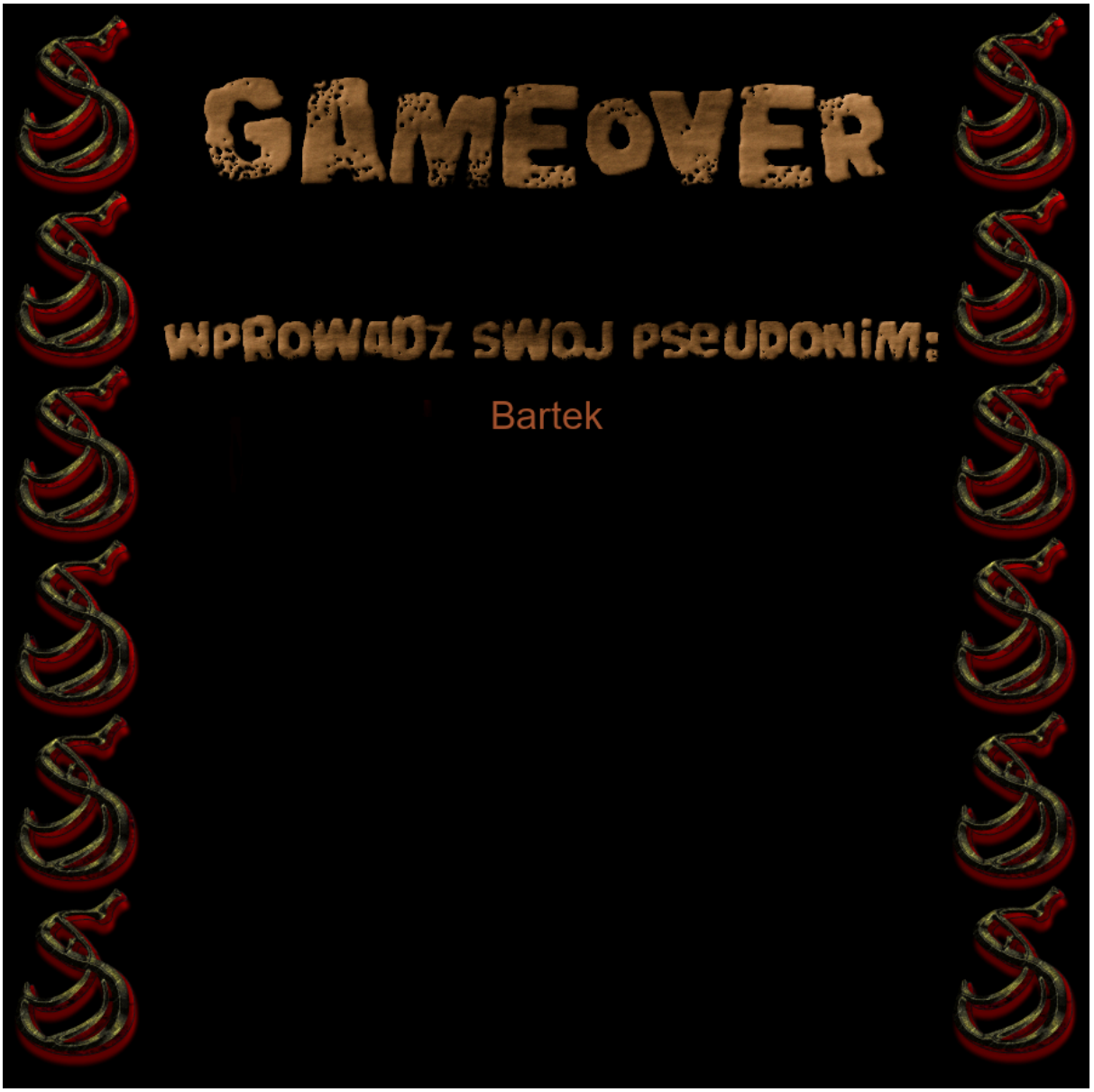
6. By gra się uruchomiła (skompilowała), należy kliknąć pusty zielony trójkąt na górze programu, lub zielony trójkąt pełnego koloru,



7. Gdy gra zostanie uruchomiona, pojawi się ekran gry, uwidoczniony na zdjęciu poniżej (istnieje możliwość wyświetlenia jednej z dwóch grafik),

8. Aby przejść grę, należy stosować się do zasad gry kółka i krzyżyka, czyli wypełnić całą planszę po skosie, pionowo lub poziomo grę krzyżykami,

9. Gdy gracz zakończy grę, zostanie uwidoczniony “gameOverImage” z możliwością zapisu swojego pseudonimu i wyniku,



10. Podany pseudonim zostanie zapisany i wyświetlona zostanie tabela wyników graczy, gra po 5 sekundach wyłączy się i zakończy swoje działanie.

g) Wymagania aplikacji, w tym niezbędne oprogramowanie do jej uruchomienia, kody źródłowe

Użyty język programowania, biblioteki, IDE, OS…:

I) Użyty język programowania: **język C++**

II) Użyta biblioteka do graficznej reprezentacji gry: **Allegro 5**

III) Użyte środowisko programistyczne: **Visual Studio 2022**

Wymagania sprzętowe:

Dla poprawnej funkcjonalności gry, wystarczy komputer z systemem operacyjnym windows 10 wzwyż, bez konieczności konkretnych podzespołów. Do uruchomienia gry i jej poprawnego działania, wystarczy komputer lub laptop ze specyfikacjami komputerów lub laptopów służbowych, czyli ze zintegrowaną kartą graficzną i procesorem wczesnej generacji. Gra nie jest ani trochę wymagająca pod względem sprzętowym.