

06.06 .2018 Wrocław

PROGRAMOWANIE OBIEKTOWE

INEW0003P

Projekt

Wydział Elektroniki	Kierunek: Informatyka
Grupa zajęciowa (np. Pn7:30):	Semestr: 2017/18 LATO
Nazwisko i Imię: Markuszewski Maciej	Nr indeksu:241258
Nazwisko i Imię: Knaziak Hubert	Nr indeksu: 241320
Nr. grupy projektowej:	1
Prowadzący: mgr.inż. Adam Włodarczyk	

TEMAT:

System rejestracji i logowania - Clicker

OCENA:

PUNKTY:

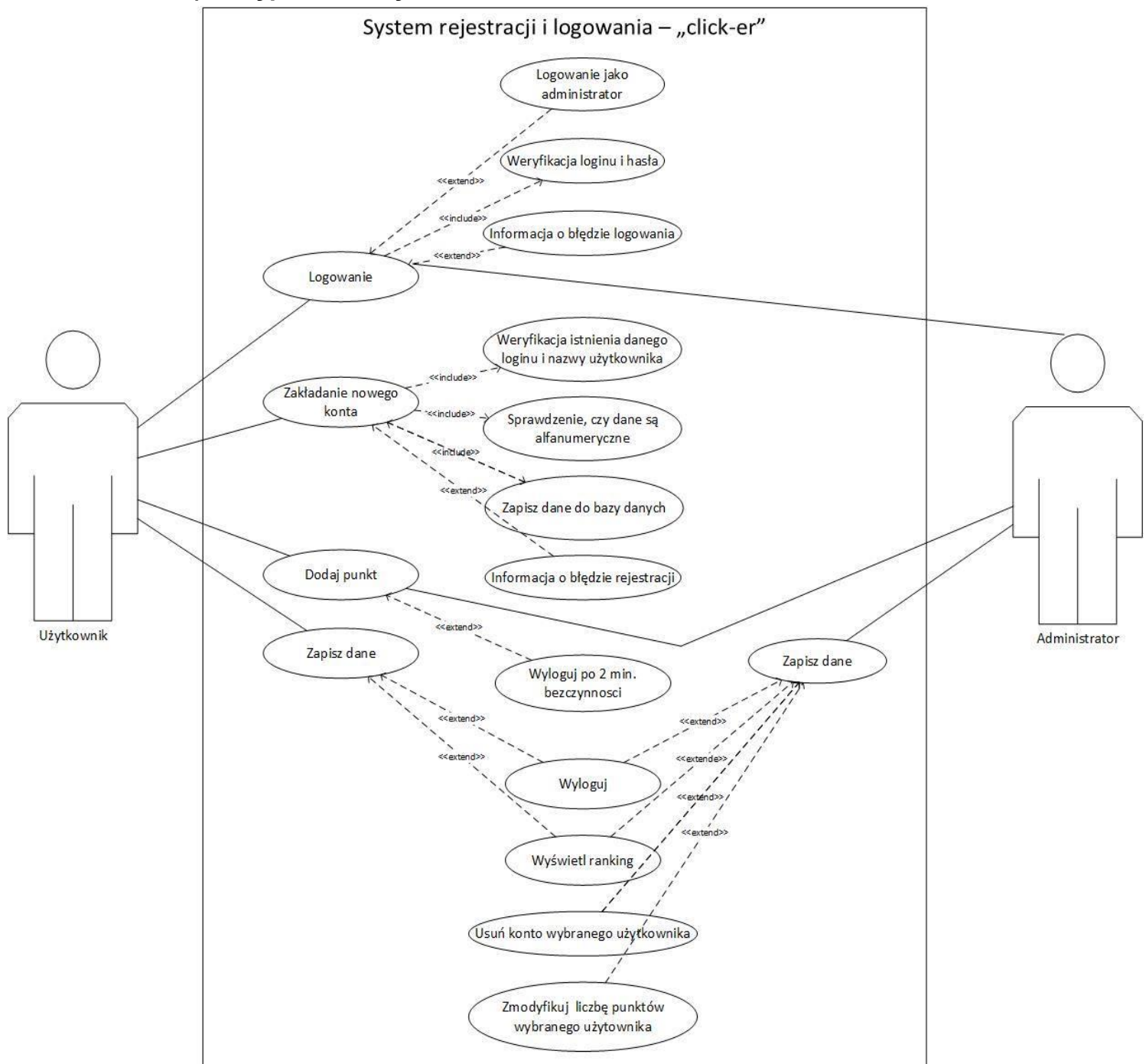
Data:

1. Założenia i opis funkcjonalny programu

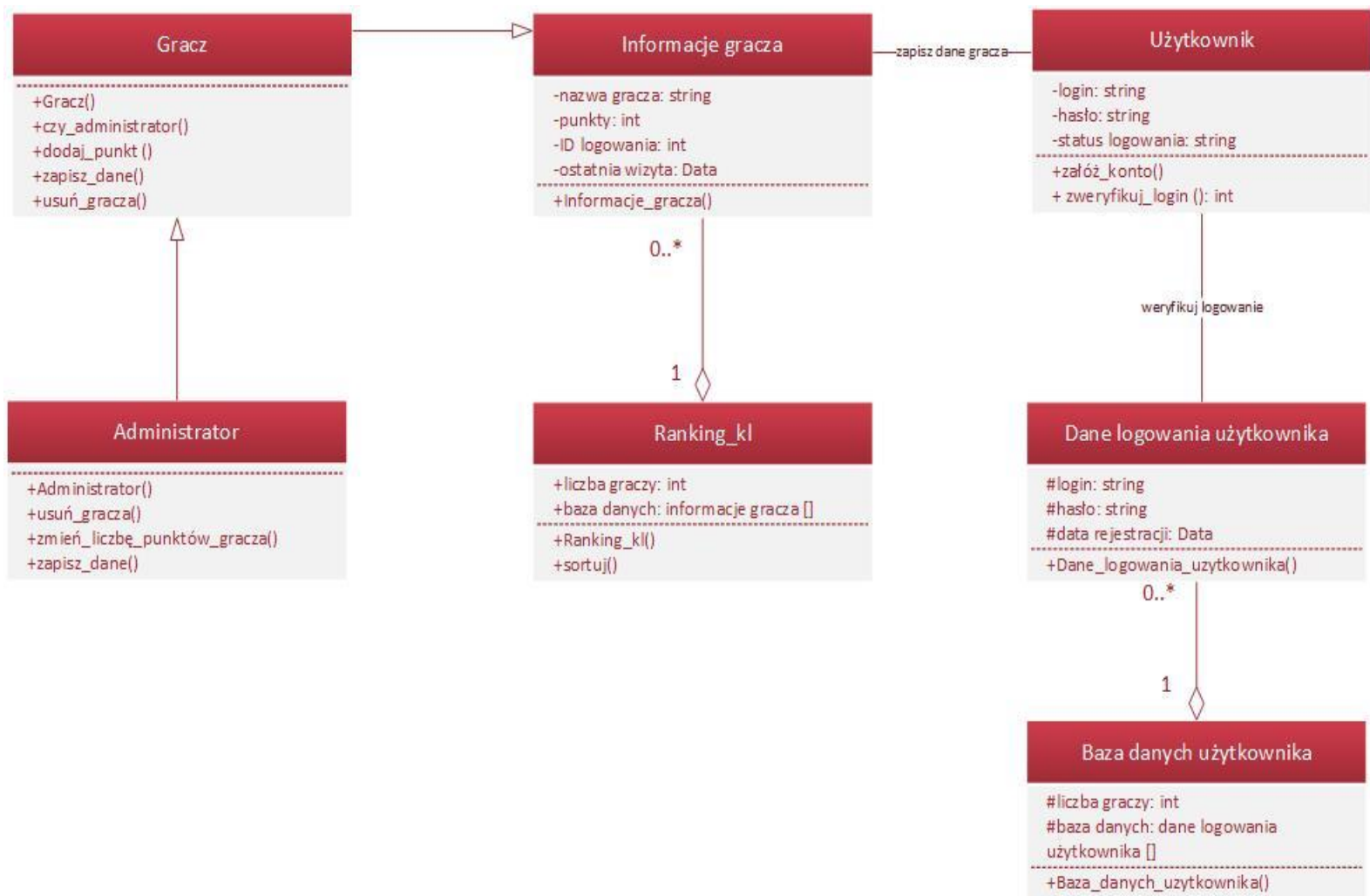
Program napisany przez grupę ma na celu wykorzystanie w swojej budowie możliwie jak największej ilości mechanizmów programowania obiektowego. Użytkownik ma możliwość zakładania nowego konta, lub też logowania się na utworzone wcześniej przy pomocy odpowiedniego loginu oraz hasła. Po zalogowaniu użytkownik zostaje przekierowany do okna, w którym po każdym naciśnięciu odpowiedniego klawisza otrzymuje punkty, które są przypisywane do jego konta. Po upływie 2 minut od pozostawienia programu na tym poziomie w stanie bezczynności automatycznie zapisuje on dane i wylogowuje gracza. Z tego samego poziomu ma on również możliwość otworzenia okna rankingowego, dzięki któremu może z łatwością sprawdzić, na której plasuje się pozycji wśród pozostałych użytkowników programu. Wyświetla ono bowiem nazwę użytkownika (podawaną przy zakładaniu konta), pozycję na liście, ilość zdobytych punktów, oraz datę ostatniego logowania każdego gracza z osobna. Użytkownik ma możliwość powrotu do poprzedniego okna, skąd może on dalej kontynuować rozgrywkę, lub też zapisać dane i wylogować się z programu. Konto posiadające specjalne uprawnienia administratorskie posiada dodatkową możliwość usunięcia dowolnego gracza, oraz modyfikacji posiadanych przez niego punktów.

2. Diagramy UML

a) Przypadków użycia



b) Klas



3. Kod klas C++ / C#

a) C++

```

#include <iostream>
#include <string>
#include <vector>
using namespace std;
  
```

```

class Baza_danych_uzytkownika; // predefinicja klasy
class Gracz;
extern Gracz G; // wykorzystanie obiektu utworzonego w innym pliku programu
extern int ID;
  
```

```

class Uzytkownik
{
    string login;//zmienna tekstowa przechowująca login gracza
    string haslo;//zmienna tekstowa przechowująca hasło gracza
    string status_logowania;//zmienna tekstowa przechowująca status logowania

public:
    bool zaloz_konto(string, string, string); /* funkcja odpowiadająca za utworzenie
    nowego konta i zapisanie danych w pliku o rozszerzeniu .csv */
    int zweryfikuj_login(string, string, Gracz*); /* funkcja sprawdzająca poprawność
    wprowadzonego loginu oraz hasła z istniejącymi już w bazie danych */
};

class Informacje_gracza
{
public:
    string nazwa_gracza;//zmienna tekstowa przechowująca nazwę gracza
    int punkty; //zmienna całkowita przechowująca ilość punktów gracza
    int ID_logowania;//zmienna całkowita przechowująca numer identyfikacyjny
    time_t ostatnia_wizyta;//zmienna typu czas przechowująca datę ostatniej wizyty
public:
    Informacje_gracza();// konstruktor domyślny
    Informacje_gracza(string);// konstruktor przyjmujący za argument nazwę gracza
    Informacje_gracza(string, int);//konstruktor przyjmujący z argument nazwę
    gracza i jego ID */
    friend bool Uzytkownik::zaloz_konto(string, string, string); /* funkcja
    zaprzyjaźniona z tą klasą */
};

class Gracz: public Informacje_gracza
{
public:
    Gracz();// konstruktor domyślny
    Gracz(int);// konstruktor przyjmujący za parametr ID gracza
    void dodaj_punkt();// funkcja odpowiadająca za przyznanie punktu graczowi
    void zapisz_dane();// funkcja odpowiadająca za zapisanie danych z rozgrywki w
    bazie danych */
};

class Ranking
{
public:

```

```

        int liczba_graczy; /* zmienna całkowita przechowująca ilość graczy w bazie
danych */
        vector<Informacje_gracza> dane ; /*vector przechowujący dane zawarte w
obiekcie klasy „informacje gracza” */
        Ranking();// konstruktor
};

```

```

class Dane_logowania_uzytkownika
{
protected:
    string login; /* zmienna tekstowa przechowująca login gracza
    string haslo; /* zmienna tekstowa przechowująca hasło gracza
    time_t data_rejestracji; /* zmienna typu czas przechowująca czas ostatniej
rejestracji */
public:
    friend bool Uzytkownik::zaloz_konto(string, string, string); /* funkcja zaprzyjaźniona
z tą klasą */
    friend int Uzytkownik::zweryfikuj_login(string, string, Gracz*); /* funkcja
zaprzyjaźniona z tą klasą */
    Dane_logowania_uzytkownika(); // konstruktor domyślny
    Dane_logowania_uzytkownika(string); /* konstruktor przyjmujący za parametr
nazwę gracza */
};

```

```

class Baza_danych_uzytkownika
{
public:
    int liczba_graczy; /* zmienna całkowita przechowująca liczbę graczy w bazie
danych */
    vector<Dane_logowania_uzytkownika> dane; /* vector danych zawartych w
obiekcie klasy „Dane_logowania_uzytkownika” */
    Baza_danych_uzytkownika(); // konstruktor
};

```

b) C#

```
public class Ranking_kl
{
    public int liczba_graczy=0; // zmienna całkowita przechowująca ilość graczy
    public List<Informacje_gracza> dane; /* lista przechowująca dane zawarte w
    obiekcie klasy „Informacje gracza” */
    public Ranking_kl(); //konstruktor
    public void Sortuj(int p, int k); /* funkcja odpowiedzialna za sortowanie kolejności
    wyświetlania graczy w oknie rankingowym */
}

class Uzytkownik
{
    private string login; //zmienna tekstowa przechowująca login gracza
    private string haslo; //zmienna tekstowa przechowująca hasło gracza

    public bool zaloz_konto(string l, string n, string h, ref Gracz G); /* funkcja
    odpowiadająca za utworzenie nowego konta i zapisanie danych w pliku o
    rozszerzeniu .csv */
    public int zweryfikuj_login(string l, string h); /* funkcja sprawdzająca poprawność
    wprowadzonego loginu oraz hasła z istniejącymi już w bazie danych */
}

public class Informacje_gracza
{
    public string nazwa_gracza; //zmienna tekstowa przechowująca nazwę gracza
    public int punkty; //zmienna całkowita przechowująca ilość punktów gracza
    public int ID_logowania; /* zmienna całkowita przechowująca numer
    identyfikacyjny */
    public long ostatnia_wizyta; //zmienna przechowująca datę ostatniej wizyty
    public Informacje_gracza(); //konstruktor domyślny
    public Informacje_gracza(string s); /* konstruktor przyjmujący za argument nazwę
    gracza */
    public Informacje_gracza(string n, int id); /* konstruktor przyjmujący z argument
    nazwę gracza i jego ID */
}
```

```

public class Gracz : Informacje_gracza
{
    public Gracz();// konstruktor domyślny
    public Gracz(int id); // konstruktor przyjmujący za parametr ID gracza
    public boolczy_administrator(); /* funkcja sprawdzająca, które konto posiada
uprawnienia administratorskie */
    public booldodaj_punkt(); /* funkcja odpowiadająca za przyznanie punktu
graczowi */.
    public voidzapisz_dane(); /* funkcja odpowiadająca za zapisanie danych z
rozgrywki w bazie danych */
    public voidusun_gracza(); /* funkcja odpowiadająca za usunięcie gracza z bazy
danych */
}

public class Administrator : Gracz
{
    public Administrator(Gracz G); //konstruktor
    public voidusun_gracza(intID,Ranking_kl R); /* funkcja odpowiedzialna za
usuwanie graczy za pomocą konta administrator */
    public voidzmien_liczbe_punktow_gracza(intID,intpunkty,Ranking_kl R);
/* funkcja odpowiadająca za zmianę liczby punktów graczy z poziomu konta
administratora */
    public voidzapisz_dane(Ranking_kl R); /* funkcja odpowiedzialna za zapisywanie
zmian dokonanych w rankingu z poziomu konta administratora */
}

class Dane_logowania_uzytkownika
{
    internalprotected string login; // zmienna tekstowa przechowująca login gracza
    internalprotected string haslo; // zmienna tekstowa przechowująca hasło gracza
    internalprotectedlongdata_rejestracji;/* zmienna przechowująca czas ostatniej
rejestracji*/
    public Dane_logowania_uzytkownika(); //konstruktor
    public Dane_logowania_uzytkownika(string s); /* konstruktor przyjmujący za
parametr nazwę gracza */
}

class Baza_danych_uzytkownika
{
    public intliczba_graczy;/* zmienna całkowita przechowująca liczbę graczy w
bazie danych */
    public List<Dane_logowania_uzytkownika> dane;;/* lista danychzawartych w
obiekcie klasy„Dane_logowania_użytkownika” */
    public Baza_danych_uzytkownika(); // konstruktor

```


}

4. Opis oraz kod własnych funkcji C#

<p>Konstruktor klasy <code>Informacje_gracza</code> przypisuje dane wczytane z pliku odpowiednim atrybutom. Dane są oddzielone od siebie przecinkami. Z tego powodu funkcja sprawdza znaki, aż nie znajdzie przecinka. Wszystkie znaki przed nim zapisuje do odpowiednich atrybutów. W przypadku niektórych z nich, które są zmiennymi liczbowymi należy zapisać znaki między przecinkami do pewnej zmiennej przechowującej znaki, a następnie przekonwertować je na dane liczbowe.</p>	<pre>public Informacje_gracza(string s) { string d = ""; int i = 0; for (; s[i] != ','; i++) { nazwa_gracza += s[i]; } i++; for (; s[i] != ','; i++) { d += s[i]; } i++; punkty = Convert.ToInt32(d); d = ""; for (; s[i] != ','; i++) { d += s[i]; } i++; ID_logowania = Convert.ToInt32(d); d = ""; for (; i < s.Length; i++) { d += s[i]; } ostatnia_wizyta = Convert.ToInt64(d); }</pre>
<p>Metoda klasy <code>gracz</code>, która dodaje punkty za każdym razem, gdy użytkownik kliknie <code>dodaj punkt</code>. Sprawdza ona również, czy gracz był aktywny w ciągu ostatnich 2 minut. Jeśli był, to poprawnie doda punkt i zaktualizuje czas ostatniej wizyty. Jeśli nie to zwróci wartość <code>false</code>, a program zapisze dane gracza oraz się wyłączy.</p>	<pre>public bool dodaj_punkt() { DateTime czas = new DateTime(this.ostatnia_wizyta); if(czas.Minute*60+czas.Second+120> DateTime.Now.Minute*60 +DateTime.Now.Second) { punkty++; this.ostatnia_wizyta = DateTime.Now.Ticks; return true; } else { return false; } }</pre>

<p>Metoda klasy Ranking_kl sortująca graczy względem liczby punktów. Jest ona wywoływana tylko wtedy, gdy na liczbę punktów graczy wpłynął administrator, przez co należy posortować wielu graczy. Używany algorytm, to algorytm quicksort. Jego średnia złożoność to $n \log n$.</p>	<pre> public void Sortuj(int p, int k) { if (p < k) { int x = this.dane[(p + k) / 2].punkty; int i = p, j = k; Informacje_gracza w; while (i <= j) { while (this.dane[j].punkty < x) j--; while (this.dane[i].punkty > x) i++; if (i <= j) { w = this.dane[i]; this.dane[i] = this.dane[j]; this.dane[j] = w; i++; j--; } } this.Sortuj(p, j); this.Sortuj(i + 1, k); } } </pre>
<p>Konstruktor klasy Baza_danych_uzytkownika, który wczytuje dane logowania użytkowników z pliku „.csv”. Pakiety danych są oddzielone znakami końca linii. Funkcja zaczytuje je oraz dodaje do listy danych obiekty Dane_logowania_uzytkownika, które są tworzone na tej samej zasadzie, co opisane wcześniej Informacje_gracza.</p>	<pre> class Baza_danych_uzytkownika { public int liczba_graczy; public List<Dane_logowania_uzytkownika> dane; <small>Odwolania: 5</small> public Baza_danych_uzytkownika() { dane = new List<Dane_logowania_uzytkownika>(); liczba_graczy = 0; StreamReader pin; pin=new StreamReader("baza.csv"); string s; s = pin.ReadLine(); for (int i = 0; s != null && s!=""; i++) { dane.Add(new Dane_logowania_uzytkownika(s)); liczba_graczy++; s = pin.ReadLine(); s = pin.ReadLine(); } pin.Close(); } } </pre>

Metoda klasy gracz, która aktualizuje liczbę punktów, czas ostatniej wizyty oraz pozycję w rankingu gracza w pliku „.csv”. Funkcja załaduje obiekt Ranking_kl. Następnie znajduje w nim gracza, który jest zalogowany i aktualizuje dane o nim, znajduje którą pozycję w rankingu powinien zająć gracz. Dane gracza są wstawione w nowe miejsce, a dane ze starego miejsca są usuwane. Po wykonaniu tych operacji dane ze zmienionego obiektu Ranking_kl są wpisywane do pliku „.csv”

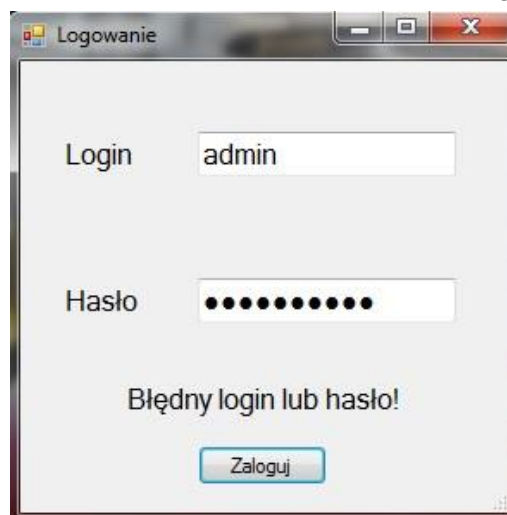
```
public void zapisz_dane()
{
    Ranking_kl R = new Ranking_kl();
    for (int i = 0; i < R.liczba_graczy; i++)
    {
        if (R.dane[i].ID_logowania == ID_logowania)
        {
            R.dane[i].punkty = punkty;
            R.dane[i].ostatnia_wizyta = DateTime.Now.Ticks;
            for (int j = 0; j < i; j++)
            {
                if (R.dane[i].punkty > R.dane[j].punkty)
                {
                    R.dane.Insert(j, R.dane[i]);
                    R.dane.RemoveAt(i + 1);
                }
            }
            break;
        }
    }
    StreamWriter rout;
    rout = new StreamWriter("ranking.csv");
    for (int i = 0; i < R.liczba_graczy; i++)
    {
        rout.WriteLine("{0},{1},{2},{3}\n",
            R.dane[i].nazwa_gracza, R.dane[i].punkty,
            R.dane[i].ID_logowania, R.dane[i].ostatnia_wizyta);
    }
    rout.Close();
}
```

5. Opis użytkowy programu C++ / C#, Java



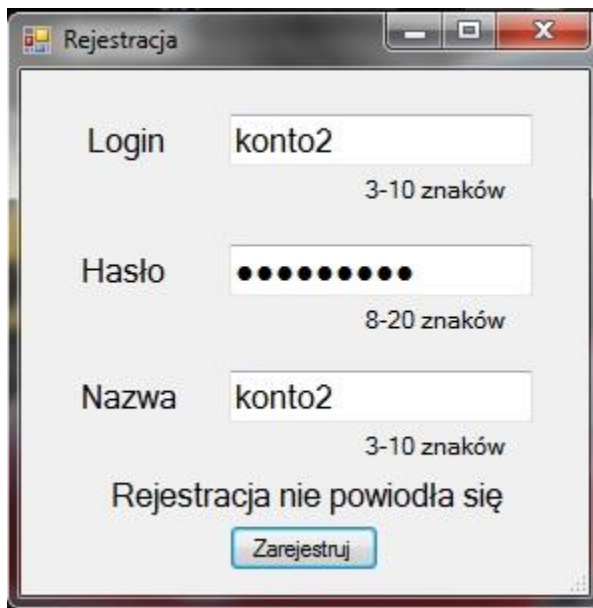
Rys.1 Ekran startowy programu

Na rys.1 widzimy ekran startowy programu. Użytkownik może z pozycji każdego następnego okna, okrywać program na pasku zadań, minimalizować, maksymalizować okno oraz zamknąć aplikację. Funkcje te zlokalizowane są w prawym górnym rogu ekranu. Z tej pozycji istnieje możliwość wyboru opcji „Zaloguj”, która powoduje przeniesienie do okna dialogowego logowania na istniejące konto. Po kliknięciu w przycisk „Założ konto” użytkownik zostanie przekierowany do okna rejestracji, z którego będzie mógł wprowadzić dane konieczne do utworzenia nowego konta.



Rys. 2 Ekran logowania

Na rys.2 widoczny jest ekran logowania na konto istniejące w bazie danych. W polu tekstowym opisanym jako „login” użytkownik wprowadza swój login, a w polu oznaczonym jako „hasło” hasło do konta. Wprowadzane do pola tekstowego jest automatycznie ukrywane przez widoczne kropki, które zastępują znaki w hasle. Po zweryfikowaniu poprawności wprowadzonego loginu oraz hasła z tymi istniejącymi już w bazie danych program przekieruje użytkownika do ekranu rozgrywki. Jeśli jednak dane okażą się niewłaściwe, ukaże się komunikat widoczny u góry przycisku „Zaloguj”.



Rejestracja

Login konto2
3-10 znaków

Hasło
8-20 znaków

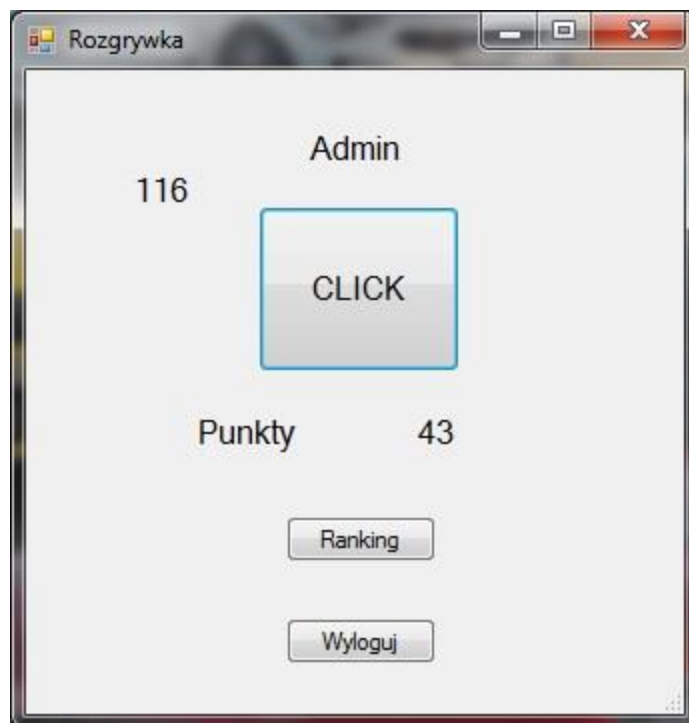
Nazwa konto2
3-10 znaków

Rejestracja nie powiodła się

Zarejestruj

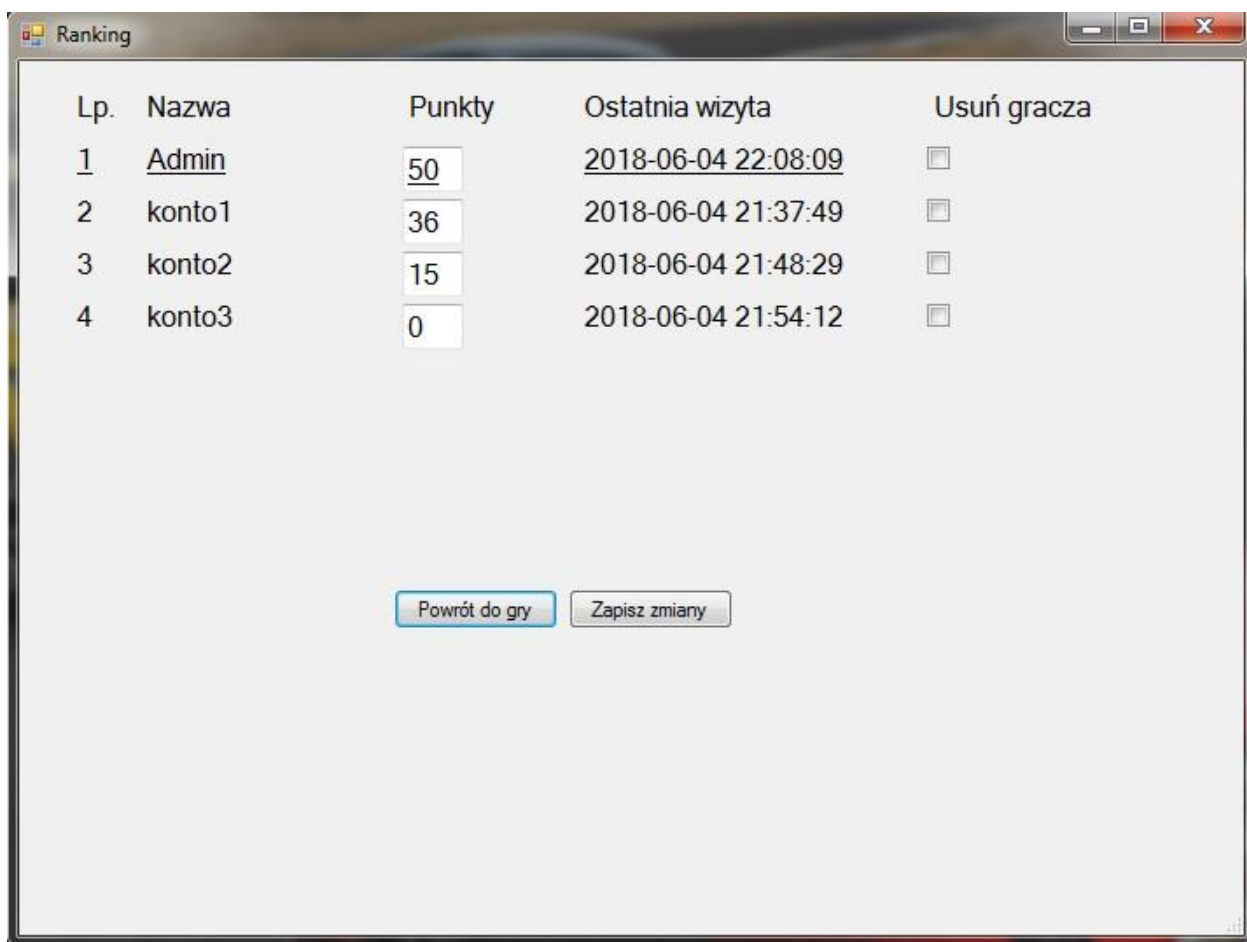
Rys. 3 Ekran rejestracji

Na rys. 3 widoczny jest ekran rejestracji nowego konta do bazy danych. Użytkownik wprowadza kolejno login, hasło, oraz nazwę gracza, która będzie później widoczna w tabeli rankingowej. Po wciśnięciu przycisku „ Zarejestruj” program sprawdzi, czy wprowadzony tekst ma odpowiednią dla danego pola ilość znaków oraz sprawdzi, czy spełniają one warunek alfanumeryczności. Po spełnieniu wszystkich warunków program zapisze informacje w bazie danych oraz przekieruje do okna rozgrywki. Jeżeli jednak który z warunków nie zostanie spełniony, wówczas pojawi się komunikat widoczny powyżej przycisku „Zarejestruj”.



Rys.4 Ekran rozgrywki

Na rys. 4 widzimy ekran rozgrywki. Na samej górze wyświetlana jest nazwa gracza prowadzącego rozgrywkę. Po lewej stronie od nazwy wyświetlany jest „czas sesji” podawany w sekundach pozostały do automatycznego wylogowania się w przypadku nie dokonania interakcji z programem przed jego upływem. Duży przycisk opatrzony napisem „CLICK” odpowiada za doliczanie punktów do konta użytkownika w ilości 1 punkt za każde naciśnięcie guzika. Pod spodem widoczna jest aktualna ilość punktów, jakie użytkownik posiada. Przycisk „Ranking” przekierowuje gracza do okna rankingowego. Poniższy przycisk „Wyloguj” powoduje zapisanie danych do konta użytkownika, a następnie zakończenie działania programu.



Rys. 5 Ekran rankingowy

Na rys. 5 widoczny jest ekran rankingowy. Konta sortowane są ze względu na ilość posiadanych punktów. Lp. wskazuje na pozycję w rankingu danego gracza. Przy każdym użytkowniku widoczne informacje to nazwa gracza, ilość zdobytych przez niego punktów, oraz data jego ostatniej wizyty. Podkreśleniem wskazywane jest konto aktualnie zalogowanego gracza. Tradycyjni użytkownicy mogą z tego poziomu jedynie sprawdzać stan listy rankingowej oraz usunąć swoje własne konto z bazy. Konto administratorskie posiada jednak specjalne uprawnienia. Administrator ma możliwość zmiany ilości punktów przypisanych do dowolnego konta, oraz usuwania dowolnego konta. Wprowadzone zmiany zatwierdzane są poprzez wciśnięcie klawisza „Zapisz zmiany” który po wykonaniu zapisu automatycznie przekieruje gracza do okna rozgrywki. Wciśnięcie przycisku „Powrót do gry” spowoduje jedynie przekserowanie do okna rozgrywki bez zapisywania wprowadzonych zmian.

Wymagania techniczne i uwagi instalacyjne:

Program ten nie posiada szczególnych wymagań sprzętowych. Może on zostać zainstalowany na dowolnym komputerze bądź laptopie wyposażonym w system Microsoft Windows, nie starszy niż Windows 7 oraz posiadający zainstalowaną platformę .NET, konieczną do poprawnego działania programu w wersji o kodzie źródłowym C#. Instalacja polega jedynie na pobraniu na komputer odpowiedniego skompresowanego załącznika w formacie .zip, a następnie odpakowaniu go w dowolnym miejscu na komputerze. Następnie wystarczy jedynie uruchomić plik wykonywalny o rozszerzeniu .exe będący zawartością rozpakowanego załącznika.

6. Listing kodu C# – wraz z komentarzami.

```
public class Ranking_kl
{
    public int liczba_graczy=0;
    public List<Informacje_gracza> dane;
    public Ranking_kl(){// Konstruktor klasy Ranking_kl. Wywoływany za każdym razem, gdy
inne funkcje odwołują się do Rankingu, a nie mają go podanego w argumencie
        dane = new List<Informacje_gracza>();
    liczba_graczy = 0;
    StreamReader rin;
    rin=new StreamReader("ranking.csv");
        string s;
        s = rin.ReadLine();
        for (int i = 0; s != null&& s != ""; i++){
    dane.Add(new Informacje_gracza(s));
    liczba_graczy++;
        s = rin.ReadLine();
        s = rin.ReadLine();
        }
    rin.Close();
    }
    public void Sortuj(int p, int k)// Metoda sortująca ranking algorytmem quicksort.
Wywoływana, gdy administrator zmieni liczbę punktów graczy
    {
    if (p < k){
    int x = this.dane[(p + k) / 2].punkty;
    int i = p, j = k;
    Informacje_gracza w;
    while (i<=j){
    while (this.dane[j].punkty < x)
        j--;
    while (this.dane[i].punkty > x)
        i++;
    if (i <= j) {
        w = this.dane[i];
    this.dane[i] = this.dane[j];
    this.dane[j] = w;
        i++;
        j--;
    this.Sortuj(p, j);
    this.Sortuj(i + 1, k);
    }
    }
    }
```

```

class Uzytkownik
{
private string login;
private string haslo;

    public bool zaloz_konto(string l, string n, string h, ref Gracz G)//Metoda sprawdzająca, czy
    podane przez użytkownika dane do nowego konta nie powtórzyły się. Jeśli nie, to nowe konto
    jest zakładane i zapisywane do bazy danych
    {
this.login = l;
this.haslo = h;
bool ok = true;
Baza_danych_uzytkownika B =new Baza_danych_uzytkownika();
        for (int i = 0; i <B.liczba_graczy; i++)
        {
if (login == B.dane[i].login)
        {
            ok = false;
break;
        }
        }
Ranking_kl R = new Ranking_kl();
        for (int i = 0; i <R.liczba_graczy; i++)
        {
if (n == R.dane[i].nazwa_gracza)
        {
            ok = false;
        }
        }
if (ok)
        {
Dane_logowania_uzytkownika d = new Dane_logowania_uzytkownika();
d.login = login;
d.haslo = haslo;
d.data_rejestracji = DateTime.Now.Ticks ;
Informacje_graczaig=new Informacje_gracza(n, B.liczba_graczy);
G.nazwa_gracza = ig.nazwa_gracza;
G.punkty = ig.punkty;
G.ID_logowania = ig.ID_logowania;
G.ostatnia_wizyta = ig.ostatnia_wizyta;
B.dane.Add(d);
B.liczba_graczy++;
R.dane.Add(ig);
R.liczba_graczy++;
        }
    }
}

```

```

StreamWriter pout, rout;
pout = new StreamWriter("baza.csv");
rout = new StreamWriter("ranking.csv");
    for (int i = 0; i < B.liczba_graczy; i++)
    {
pout.WriteLine("{0},{1},{2}\n", B.dane[i].login, B.dane[i].haslo, B.dane[i].data_rejestracji);
    }
    for (int i = 0; i < R.liczba_graczy; i++)
    {
rout.WriteLine("{0},{1},{2},{3}\n", R.dane[i].nazwa_gracza, R.dane[i].punkty,
R.dane[i].ID_logowania, R.dane[i].ostatnia_wizyta);
    }
pout.Close();
rout.Close();
    return true;
}
else{
    return false;
}
}
    public int zweryfikuj_login(string l, string h) // Metoda sprawdzająca poprawność loginu i
hasła podczas logowania
    {
Baza_danych_uzytkownika B=new Baza_danych_uzytkownika();
Ranking_kl R=new Ranking_kl();
    login = l;
haslo = h;
    for (int i = 0; i < B.liczba_graczy; i++)
    {
if (login == B.dane[i].login && haslo == B.dane[i].haslo)
    {
        for (int j = 0; j < R.liczba_graczy; j++)
        {
if (i == R.dane[j].ID_logowania)
            {
                return i;
            }
        }
    }
}
    }
    return -1;
}
}

```

```

public class Informacje_gracza
{
    public string nazwa_gracza;
    public int punkty;
    public int ID_logowania;
    public long ostatnia_wizyta;
    public Informacje_gracza()
    {
        //puste
    }
    public Informacje_gracza(string s)//Konstruktor klasy Informacje_gracza. Wywołany za
    każdym razem, gdy wczytywany jest ranking z bazy danych
    {
        string d = "";
int i = 0;
        for (; s[i] != ','; i++){
nazwa_gracza += s[i];
        }
        i++;
        for (; s[i] != ','; i++){
            d += s[i];
        }
        i++;
        punkty = Convert.ToInt32(d);
        d = "";
        for (; s[i] != ','; i++){
            d += s[i];
        }
        i++;
        ID_logowania = Convert.ToInt32(d);
        d = "";
        for (; i < s.Length; i++){
            d += s[i];
        }
        ostatnia_wizyta = Convert.ToInt64(d);
    }
    public Informacje_gracza(string n, int id)
    {
nazwa_gracza = n;
        punkty = 0;
        ID_logowania = id;
        ostatnia_wizyta = DateTime.Now.Ticks;
    }
}

```

```

public class Gracz : Informacje_gracza
{
    public Gracz()//Konstruktor domyślny klasy Gracz. Wywołany podczas rejestracji
    {
        //puste
    }
    public Gracz(int id)//Konstruktor klasy Gracz. Wywołany po pozytywnej odpowiedzi
    metody sprawdzającej poprawność loginu i hasła
    {
        Ranking_kl R = new Ranking_kl();
        for (int i = 0; i < R.liczba_graczy; i++)
        {
            if (id == R.dane[i].ID_logowania)
            {
                nazwa_gracza = R.dane[i].nazwa_gracza;
                punkty = R.dane[i].punkty;
                ID_logowania = R.dane[i].ID_logowania;
                ostatnia_wizyta = DateTime.Now.Ticks;
            }
        }
        public bool czy_administrator()//Metoda sprawdzająca, czy gracz, który się zalogował nie
        jest administratorem
        {
            if (this.nazwa_gracza == "Admin" && this.ID_logowania == 0)
            {
                return true;
            }
            else return false;
        }
        public bool dodaj_punkt()//Metoda dodająca punkty graczowi, jeśli ten klika w przycisk
        dodający punkt
        {
            DateTime czas = new
            DateTime(this.ostatnia_wizyta);
            if(czas.Minute*60+czas.Second+120>
            DateTime.Now.Minute*60 +DateTime.Now.Second) {
                punkty++;
                this.ostatnia_wizyta = DateTime.Now.Ticks;
                return true;
            }
            else{
                return false;
            }
        }
    }
}

```

```

        public void zapisz_dane()// Metoda zapisująca dane. Wywoływana, gdy gracz chce się
        wylogować, lub przeglądać ranking
        {
            Ranking_kl R = new Ranking_kl();
            for (int i = 0; i < R.liczba_graczy; i++)
            {
                if (R.dane[i].ID_logowania == ID_logowania)
                {
                    R.dane[i].punkty = punkty;
                    R.dane[i].ostatnia_wizyta = DateTime.Now.Ticks;
                    for (int j = 0; j < i; j++)
                    {
                        if (R.dane[i].punkty > R.dane[j].punkty)
                        {
                            R.dane.Insert(j, R.dane[i]);
                            R.dane.RemoveAt(i + 1);
                        }
                    }
                    break;
                }
            }
            StreamWriter rout;
            rout = new StreamWriter("ranking.csv");
            for (int i = 0; i < R.liczba_graczy; i++)
            {
                rout.WriteLine("{0},{1},{2},{3}\n",
                    R.dane[i].nazwa_gracza, R.dane[i].punkty,
                    R.dane[i].ID_logowania, R.dane[i].ostatnia_wizyta);
            }
            rout.Close();
        }

        public void usun_gracza()// Metoda usuwająca gracza, gdy ten się na to zdecyduje
        {
            Ranking_kl R=new Ranking_kl();
            Baza_danych_uzytkownika B = new Baza_danych_uzytkownika();
            for(int i = 0; i < R.liczba_graczy; i++)
            {
                if (R.dane[i].ID_logowania==this.ID_logowania)
                {
                    R.dane.RemoveAt(i);
                    B.dane.RemoveAt(this.ID_logowania);
                    R.liczba_graczy--;
                    B.liczba_graczy--;
                    break;
                }
            }
        }
    }

```

```

    }
    }
    for(int i = 0; i < R.liczba_graczy; i++)
    {
if (R.dane[i].ID_logowania>this.ID_logowania)
    {
R.dane[i].ID_logowania--;
    }
    }
    StreamWriter rout;
    rout = new StreamWriter("ranking.csv");
    for (int i = 0; i < R.liczba_graczy; i++)
    {
rout.WriteLine("{0},{1},{2},{3}\n", R.dane[i].nazwa_gracza, R.dane[i].punkty,
R.dane[i].ID_logowania, R.dane[i].ostatnia_wizyta);
    }
    rout.Close();
    StreamWriter pout;
    pout = new StreamWriter("baza.csv");
    for (int i = 0; i < B.liczba_graczy; i++)
    {
pout.WriteLine("{0},{1},{2}\n", B.dane[i].login, B.dane[i].haslo, B.dane[i].data_rejestracji);
    }
    pout.Close();
    }
    }
    public class Administrator : Gracz
    {
        public Administrator(Gracz G)//Konstruktor klasy Administrator. Wywołuje się po
pozytywnej odpowiedzi funkcji sprawdzającej, czy gracz jest administratorem
        {
            this.ID_logowania = G.ID_logowania;
            this.nazwa_gracza = G.nazwa_gracza;
            this.punkty = G.punkty;
            this.ostatnia_wizyta = G.ostatnia_wizyta;
        }

        public void usun_gracza(int ID, Ranking_kl R)//Metoda usuwająca graczy z decyzji
administratora
        {
            Baza_danych_uzytkownika B = new Baza_danych_uzytkownika();
            int ID_logowania = R.dane[ID].ID_logowania;
            B.dane.RemoveAt(ID_logowania);
            R.dane.RemoveAt(ID);
            R.liczba_graczy--;

```

```

B.liczba_graczy--;
    for (int i = 0; i < R.liczba_graczy; i++){
    if (R.dane[i].ID_logowania > ID_logowania) {
    R.dane[i].ID_logowania--;
        }
    }
StreamWriter pout;
pout = new StreamWriter("baza.csv");
    for (int i = 0; i < B.liczba_graczy; i++){
pout.WriteLine("{0},{1},{2}\n", B.dane[i].login, B.dane[i].haslo, B.dane[i].data_rejestracji);
    }
pout.Close();
    }
    public void zmien_liczbe_punktow_gracza(int ID,int punkty,Ranking_kl R)
    {
R.dane[ID].punkty = punkty;
if (R.dane[ID].ID_logowania == this.ID_logowania) {
this.punkty = punkty;
    }
    }
    public void zapisz_dane(Ranking_kl R)//Metoda zapisująca dane do rankingu
modyfikowanego przez administratora
    {
Baza_danych_uzytkownika B = new Baza_danych_uzytkownika();
StreamWriter rout;
rout = new StreamWriter("ranking.csv");
    for (int i = 0; i < R.liczba_graczy; i++)
    {
rout.WriteLine("{0},{1},{2},{3}\n", R.dane[i].nazwa_gracza, R.dane[i].punkty,
R.dane[i].ID_logowania, R.dane[i].ostatnia_wizyta);
    }
rout.Close();
StreamWriter pout;
pout = new StreamWriter("baza.csv");
    for (int i = 0; i < B.liczba_graczy; i++)
    {
pout.WriteLine("{0},{1},{2}\n", B.dane[i].login, B.dane[i].haslo, B.dane[i].data_rejestracji);
    }
pout.Close();
    }
}
Class Dane_logowania_uzytkownika
{
    Internal protected string login;

```



```

Internal protected string haslo;
Internal protected long data_rejestracji;
    public Dane_logowania_uzytkownika()//Konstruktor domyślny klasy
Dane_logowania_uzytkownika. Wywołany podczas rejestracji
    {
        //puste
    }
    public Dane_logowania_uzytkownika(string s)//Konstruktor klasy
Dane_logowania_uzytkownika. Wywołany za każdym razem, gdy wczytywana jest
Baza_danych_uzytkownika z bazy danych
    {
        string d = "";
int i = 0;
        for (; s[i] != ','; i++)
        {
            login += s[i];
        }
        i++;
        for (; s[i] != ','; i++)
        {
            haslo += s[i];
        }
        i++;
        for (; i < s.Length; i++)
        {
            d += s[i];
        }
        data_rejestracji = Convert.ToInt64(d);
    }

class Baza_danych_uzytkownika
{
    public int liczba_graczy;
    public List<Dane_logowania_uzytkownika> dane;
    public Baza_danych_uzytkownika()// Konstruktor klasy Baza_danych_uzytkownika.
Wywołany za każdym razem, gdy inne funkcje odwołują się do bazy danych, a nie mają go
podanego w argumencie
    {
        dane = new List<Dane_logowania_uzytkownika>();
        liczba_graczy = 0;
        StreamReader pin;
        pin=new StreamReader("baza.csv");
        string s;
        s = pin.ReadLine();
    }
}

```

```

        for (int i = 0; s != null&& s!=""; i++)
        {
dane.Add(new Dane_logowania_uzytkownika(s));
liczba_graczy++;
            s = pin.ReadLine();
            s = pin.ReadLine();
        }
pin.Close();
    }
}

```

7. Wnioski

W wersji programu pisanej w języku C# ostatecznie udało się osiągnąć wszystkie założenia funkcjonalne z diagramu Use Case. Z kolei w wersji pierwszej, pisanej w języku C++ nie udało się oprogramować rzeczy takich jak konto administratora oraz wyświetlanie rankingu. Problemem okazała się różnica w konstrukcji obu języków. Pomimo tego, że ich składnia jest do siebie bardzo podobna, to jednak język C# jest o wiele mocniej orientowany obiektowo niż C++. W projekcie opartym na koncepcji programowania obiektowego język C# okazał się być znacznie lepszym do osiągnięcia założonych celów. W języku C++ problemem okazało się również przeniesienie funkcjonowania programu do środowiska graficznego GUI. O wiele trudniejszym było połączenie ze sobą GUI i kodu pisanego w języku C++ niż w C#. Przykładowo przekonwertowanie danych wpisywanych przez użytkownika do elementu „TextBox” na zmienną typu string w języku C++ wymagało użycia tzw. „marshall”, którego użycie przysparzało wielu kłopotów. Z kolei w języku C# tę samą czynność można było przeprowadzić przy użyciu prostej oraz intuicyjnej metody „Convert”.

dokumentacja:

Gr1_Proj_KnaziakH_MarkuszeowskiM.pdf

np. Gr2_Proj_KowalskaJ_NowakP.pdf

katalog programu (bez kompresji)

Gr1_Proj_KnaziakH_MarkuszeowskiM