ДИПЛОМНА РОБОТА

ДР.ПІс – 21.00.000 ПЗ

Група ПІс-2015

Спірін М.В.

2019

ПВНЗ Університет Короля Данила

Кафедра Інформаційних технологій та програмної інженерії

УДК 004

ДИПЛОМНА РОБОТА

Тема ***Розробка програми для організації спортивних турнірів за Швейцарською системою засобами C#***

Напрям підготовки ***6.050103 – «Програмна інженерія»***
(код і назва спеціальності)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДР.ПІс – 21.00.000 ПЗ
(позначення)

Студент

Спірін М.В.
(підпис) (дата) (розшифрування підпису)

Керівник проекту
д.т.н.,доц. Мельничук С.І.
(посада) (підпис) (дата) (розшифрування підпису)

Нормоконтроль
к.т.н. Мануляк І.З.
(посада) (підпис) (дата) (розшифрування підпису)

Допускається до захисту
Завідувач кафедри
д.т.н.,доц. Мельничук С.І.
(посада) (підпис) (дата) (розшифрування підпису)

2019

ПВНЗ УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет *Інформаційних технологій*
Кафедра *Інформаційних технологій та програмної інженерії*
Напрям підготовки *6.050103 «Програмна інженерія»*

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІТПІ
_____ *С.І. Мельничук*
"____" _____ 2019 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Студенту *Спіріну Михайлу Вікторовичу*

1. Тема проекту (роботи) *Розробка програми для організації спортивних турнірів за Швейцарською системою засобами C#*

Затверджена наказом ректора Університету Короля Данила від *15.11.2018 р. № 20/4*
2. Термін здачі студентом закінченого проекту (роботи) *10.06.2019р.*
3. Вихідні дані до проекту (роботи) *C#, Visual Studio, .NET*

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити) *1. Опис наявних аналогів, алгоритмів роботи та опис використаних технологій. 2. Розробка алгоритмічного та програмного забезпечення. 3. Розробка програмного забезпечення та інтерфейсу програми.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Швейцарська система, структура бази даних, інтерфейс користувача, розробка програмного забезпечення.*

6. Консультанти з проекту (роботи), із зазначенням розділів проекту, що стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 30.10.2018

Керівник Мельничук С.І.

Завдання прийняв до виконання Спірін М.В.

КАЛЕНДАРНИЙ ПЛАН

Пор №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1.	Опис наявних аналогів, алгоритми роботи та опис використаних технологій	30.11.2018	
2.	Розробка алгоритмічного та програмного забезпечення	25.12.2018	
3.	Розробка програмного забезпечення та інтерфейсу програми	21.02.2019	
4.	Оформлення пояснювальної записки	31.05.2019	
5.	Оформлення графічного матеріалу та підготовка до захисту роботи	04.06.2019	

Студент-дипломник *Спірін М.В.*
(підпис) (розшифровка підпису)

Керівник проекту *Мельничук С.І.*
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Дипломна робота присвячена розробці інтерфейсу користувача та програмного забезпечення для проведення турнір за Швейцарською системою. В даній роботі реалізовано інтерфейс для коректного управління програмою, базу даних, внесення команд, жеребкування та плей-офф.

Через швидкий розвиток кібер-спорту та просто спорту в Україні стрімко розвивається, потреба в програмах для організаторів спортивних подій і не тільки. Тому головним завданням даної роботи є розроблення інтерфейсу користувача та програмного забезпечення для зручного та оптимізованого управління програмою.

SUMMARY

The thesis is devoted to the development of user interface and software for the tournament under the Swiss system. In this work an interfaces are implemented for correct control of the program, data base, team introduction, draw and playoffs.

Due to the rapid development of cyber-sport and simple sports in Ukraine, it is rapidly developing, the need for programs for organizers of sports events, and not only. Therefore, the main task of this work is to develop a user interface and software for convenient and optimized management of the program.

РЕФЕРАТ

Розрахунково-пояснювальна записка: 78 сторінок, 36 рисунків, 1 додаток.

Об’єктом дослідження виступає організація спортивних подій, а саме Швейцарська система проведення турнірів засобами C#.

Метою роботи є розробка програмного забезпечення та інтерфейсу програми на основі C# в редакторі Visual Studio для організації спортивних турнірі за Швейцарською системою.

Відповідно до поставленого завдання в роботі використовувалися наукові досягнення в областях розробки інформаційних систем і програмного забезпечення. В роботі використані технології для розробки програмного забезпечення використовувалася мова програмування C#, мультиредактор Visual Studio в якому за допомогою windows forms був створений інтерфейс користувача.

C#, VISUAL STUDIO, .NET FREAMWORK, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, ПРОГРАМА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Т.Д. – так далі

wind. forms – windows froms

CLR – Common Language Runtime

Sql – Structured query language

БД – База даних

ПЗ – програмне забезпечення

CRUD – Create Read Update Delete

DBMS – Database Management System

ВСТУП

Актуальність теми. Кожна спортивна подія включає в себе регламент завдяки якому утворюються певні правила на час проведення різних турнірів. Є багато різних систем проведення спортивних подій тай не тільки спортивних але в яких і є дух змагання і мова піде про Швейцарську систему.

Швейцарська система - один із способів (систем) організації спортивних змагань. Особливо часто її застосовують при проведенні турнірів з інтелектуальних видів ігор, таким як шашки, шахи, го і подібні. Особливість цієї системи в тому, що змагання проводяться без вибування гравців. Це дозволяє приймати в них участь необмеженому числу спортсменів і визначати переможця при невеликій кількості проведених турів. Вперше таку систему змагань застосували на шаховому турнірі, який відбувся в 1895 році в Цюріху (Швейцарія). Саме тому вона і отримала назву швейцарської.

Умови проведення – традиційно прийнято проводити спортивні турніри за коловою системою, що дозволяє отримати найбільш об'єктивний результат. При такій організації турніру кожен гравець повинен провести не менше однієї партії з кожним з учасників. За сумою набраних очок визначають переможця. Але при такій системі з кожним новим гравцем багаторазово зростає кількість проведених зустрічей. Якщо ж число учасників турніру перевалює за 30, то провести таку кількість партій стає неможливо. У турнірах, організованих за швейцарською системою, часом бере участь більше сотні гравців, і для визначення переможця вистачає 500 зустрічей в 10 турах. У коловою системою при такій кількості учасників треба було б провести близько 5000 партій.

Переваги – швейцарська система приваблива тим, що дозволяє скоротити час на проведення змагань. При її застосуванні кількість проведених турів визначається заздалегідь, згідно з положенням про турнір. У кожному турі система підбору пар організована так, що в результаті дозволяє забезпечити чіткий розподіл місць за кількістю набраних очок. У швейцарською системою

прийнято вважати, що для визначення переможця досить стільки турів, скільки потрібно ступенів для виявлення переможця за нокаут-системою при такій же кількості гравців.

Організація турніру за швейцарською системою та порядок проведення - у турнірі бере участь парна кількість гравців. У першому турі підбір пар виробляється або випадкової жеребкуванням, або згідно з рейтингом учасників. При другому варіанті складання пар всіх учасників ділять на групи найсильніших гравців і слабких і складають пари з урахуванням рейтингу. При цьому найсильнішого учасника з першої групи ставлять в пару до найсильнішого з другої, другого за силою гравця з першої групи - до другого з другої. Наприклад, якщо в турнірі беруть участь 60 спортсменів, то перший гравець (за рейтингом) буде грати з 31-м, другий – з 32-м і так далі. Якщо число учасників виявилося непарних, то гравець під останнім номером в першому турі отримує очко без гри.

Починаючи з другого туру гравців ділять на групи з рівною кількістю отриманих очок. За підсумками першого туру вийде три групи учасників: виграли, які зіграли внічию і програвши. Якщо в якійсь групі виявиться непарна кількість учасників, то одного з них переводять в найближчу за очками групу.

Для наступного туру пари гравців складають за тим же рейтинговим принципом, але з гравців однієї групи: найсильніший гравець з першої половини групи зустрічається (по можливості) з найсильнішим гравцем з другої половини тієї ж групи. Але при цьому одні і ті ж гравці не можуть провести в турнірі більше однієї партії. У шахових або шашкових турнірах, крім того, необхідно дотримуватися умова чергування кольору. Щоб умови були рівними, у кожного гравця від туру до туру повинен чергуватися колір фігур (має бути однакова кількість ігор чорними і білими). У шахах не допускається поспіль три (в шашках - чотири) гри фігурами одного кольору, за винятком останнього туру.

Розраховують кількість турів, виходячи з формули: [log2 N] + [log2 (k-1)], де N - кількість гравців, а k - кількість призових місць (при k = 1, [log2 (k-1)] = 0). Наприклад, для визначення переможця серед восьми гравців достатньо трьох турів, з шістнадцяти - чотирьох і т.д. Щоб визначити четвірку найсильніших,

досить чотирьох турів при восьми гравців, п'яти - при шістнадцяти і далі згідно з формулою. Місця розподіляють за кількістю набраних очок.

Якщо учасники набрали рівну кількість очок, то їх розподіляють за коефіцієнтом Бухгольца, який вираховується як сума очок, набрана в турнірі усіма суперниками даного учасника. Поряд з ним можливе застосування середнього рейтингу суперників (більш високе місце присудять учаснику, суперники якого мають середній рейтинг і вище) або «коефіцієнта прогресу» – перевага отримає гравець, довше перебував під час турніру на більш високому місці [1].

Об’єкт дослідження виступає програма, а саме інтерфейс користувача та програмна забезпечення для проведення турнірів за Швейцарською системою.

Задачі досліджень. Відповідно до теми в роботі поставлені такі задачі:

- зручний інтерфейс;
- максимальна швидкість обробки даних;
- коректна робота і видавати чітких релевантних результатів;
- чесна та чітка робота з жеребкуванням та розподілом команд.

Методи досліджень. При вирішенні поставленого завдання використовувалися наукові досягнення в областях розробки інформаційних систем і програмного забезпечення. В роботі використані технології для розробки інтерфейсу користувача та програмного забезпечення такі як, мова програмування C#, та мультизадачна проектна програма Visual Studio.

Результати досліджень. Результатом дипломної роботи є програмне забезпечення десктопної програми для реалізації Швейцарської системи проведення турнірів, для якої розроблений інтерфейс користувача.

1 ОПИС НАЯВНИХ АНАЛОГІВ, АЛГОРИТМИ РОБОТИ ТА ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

1.1 Огляд задач та методів які забезпечують турнір за Швейцарською системою

На теперішній час багато людей якщо хочуть провести якийсь спортивний захід зіштовхуються з проблемою автоматизації турніру і більшість речей приходиться робити в ручну. Керувати груповою стадією та єтапом на виліт в ручну є дуже складно. Програма дозволить автоматично створювати групи та сітки плей-офф що з економить організаторам безліч часу. Тепер не потрібно буде збирати людей для розподілення груп і більше не прийдеться використовувати велику кількість паперу що б описати групи та хто з ким іграє.

Раніше в жеребкуванні (рис. 1.1), та створення сітки турніру тратили дуже багато часу також його витрачали на перевірку всіх матчі та підрахування додаткових показників. Програма дозволить заносити та створювати сітку в пару кліків. З економлений час допоможе оптимізувати зайнятість як і учасників так і журі. Також у спорті як і в житті є багато спірних епізодів які будуть вирішуватися за допомогою чітко написаної програми яка не буде звертати уваги на зовнішні фактори протидії спортивного принципу. Зазвичай посів йде по рівню команд в головній спортивній системі в якій перебуває команда.

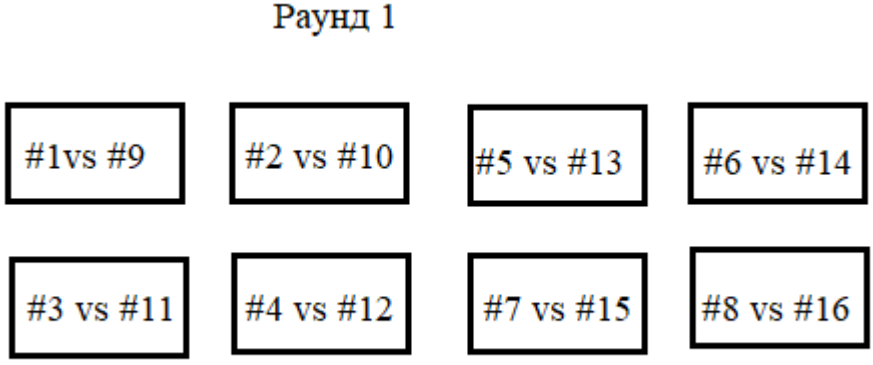

Рисунок 1.1 – Перший раунд в Швейцарській системі

В наступному раунді грають переможці з переможцями у яких відповідно статистика (1-0) та команди які отримали поразку відповідно (0-1) між собою (рис 1.2). В цьому етапі команди розподіляються згідно регламенту турніру якщо він передбачає жеребкування по посіву або завдяки випадковості.

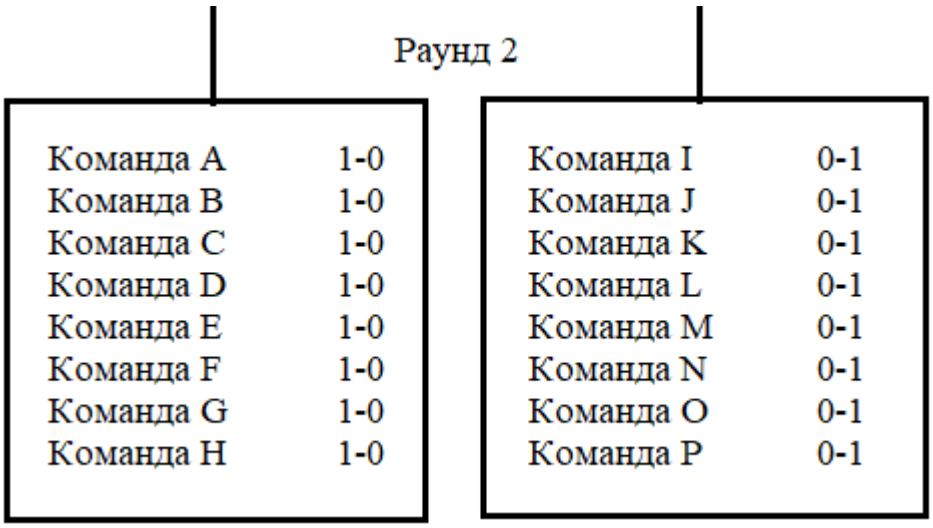

Рисунок 1.2 – Другий раунд в Швейцарській системі

Раунд номер три являє собою екватор турніру де вже визначилися фаворити з рахунком (2-0) і аутсайдери (0-2) (рис 1.3), також триває боротьба команд з статистикою (1-1). Також потрібно відміти що зазвичай в такому форматі відміняють повторні щоб зберегти інтригу і зменшити шанс того що команда яка сильніша за іншу попалася з нею ще раз зі статистикою (1-1). Це правило дотримують не всі організатори тай воно являється вибірковим.

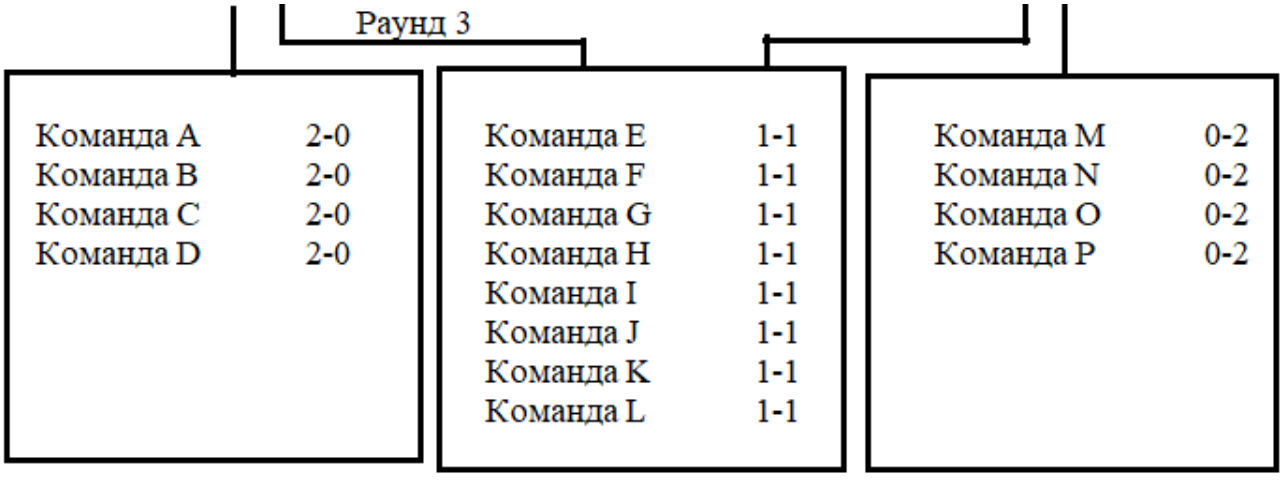

Рисунок 1.3 – Третій раунд в Швейцарській системі

Цей раунд (рис. 1.4), уже має перших чемпіонів та невдах (3-0) і (0-3) відповідно також загострюється протистояння адже команди за крок від вибули і проходу дальше (1-2) і (2-1) відповідно.

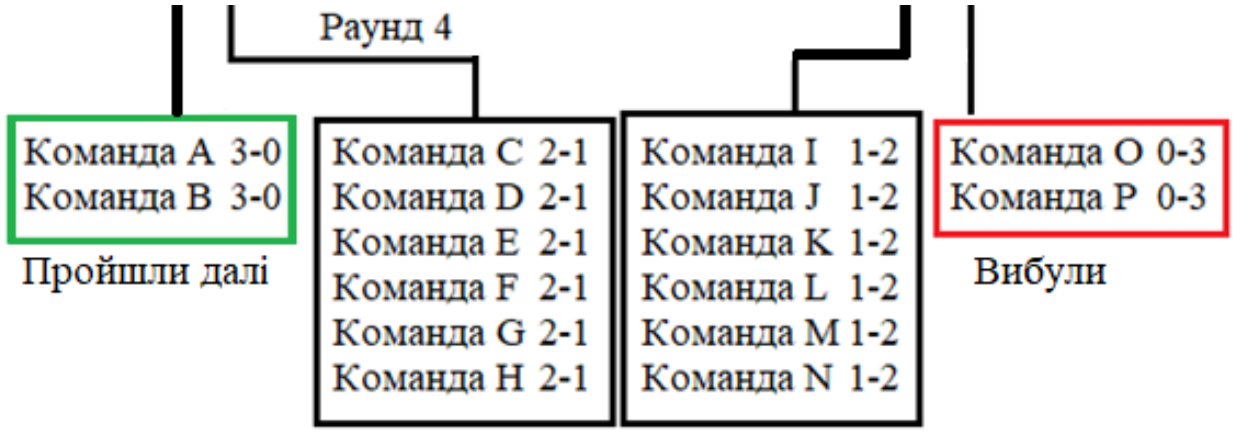

Рисунок 1.4 – Четвертий раунд в Швейцарській системі

Вирішальний раунд для команд зі статикою (2-2) (рис 1.5), які між собою будуть розігрувати путівки в основну частину.

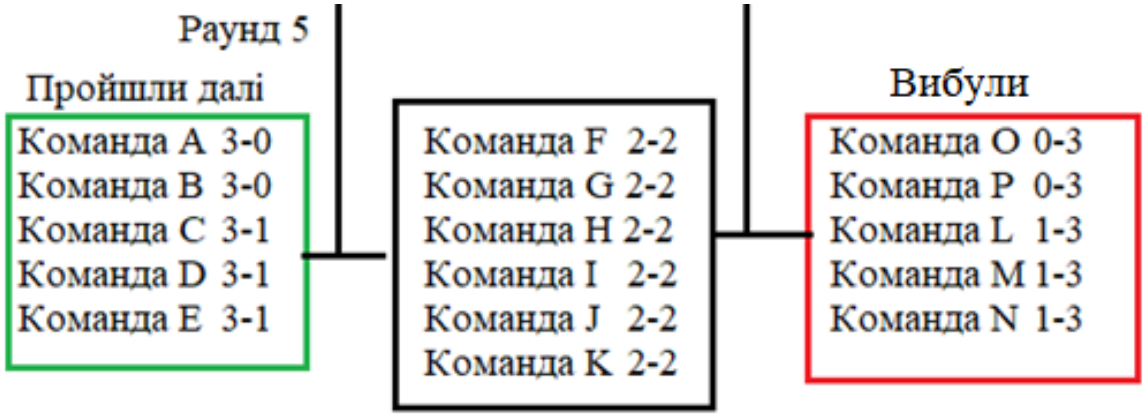

Рисунок 1.5 – П’ятий раунд в Швейцарській системі

Підсумкова таблиця (рис. 1.6), команди з трьома перемогами продовжують боротьбу в турнірі, команди з трьома поразками вибувають. Статистика команд Які пройшли далі (3-0), (3-1), (3-2) і відповідно які вибули (2-3), (1-3), (0-3).

Рисунок 1.6 – Підсумкова таблиця

Турнір буде продовжуватися по регламенту змагань чи це буде плей-офф або ще груповий етап будуть вирішувати організатори.

1.2 Огляд програмного забезпечення, що реалізує інформаційну підтримку за Швейцарською системою

Зараз в світі є багато сайтів які дозволяють користувачеві продивлятися матчі та їхню статистику, також турніри з їх сітками та груповими етапами. Загалом більшість інформації для організаторів спортивних подій вони получають з офіційних ресурсів таких як федерація футболу, баскетболу, хокею, шахмат і т.д.

Найчастіше можна зіткнутися з Швейцарською системою в таких дисциплінах як кібер-спорт, шахи, шашки та настільний теніс. Переваги Швейцарської системи це конкурентро здатні матчі, які роблять обстановку набагато напруженішею в самому турнірі де кожний раунд вирішує саме команда, а не якісь не відомі чинники.

Наступний великий турнір де буде задіяна Швейцарська система буде проходити в Берліні. Це турнір в кібер-спорті, а саме в диспліні Counter-Strike Global Offensive (рис. 1.7). Учасники цього турніру будуть кращі команди своїх регіонів по цілому світі.

Рисунок 1.7 – Банер турніру

Також на цьому турнірі буде присутня і Українська організація в яку входять не тільки гравці з українським паспортом, а й представляють інші країни.

Слідкувати за перепитіями на турнірі можна за допомогою інтернет ресурсів, а саме: liquipedia.net/counterstrike, csgomajor.starladder.com та інших сайтів які ведуть інтерактивну статистику.

На даний момент для організації спортивної події за допомогою Швейцарської системи є не великий перелік ресурсів. Беремо один з найпопулярніших онлайн сайтів по шахматним турнірам chessresults.ru. (рис. 1.8) Він являє собою хороший ресурс для проведення жеребкування турнірної сітки, та продовження в вигляді реалізації Швейцарської системи. Плюс цього ресурса це надійність визначена часом, та інформаційна підтримка оскільки це не просто програма для жеребкування, а й сайт для користувачів та фанатів шахмат. Перейдем до недоліків, головним являється відсутність відкритого коду для провірки на чесність, орієнтиру по сайту, та підтримка тільки одного виду дисципліни.

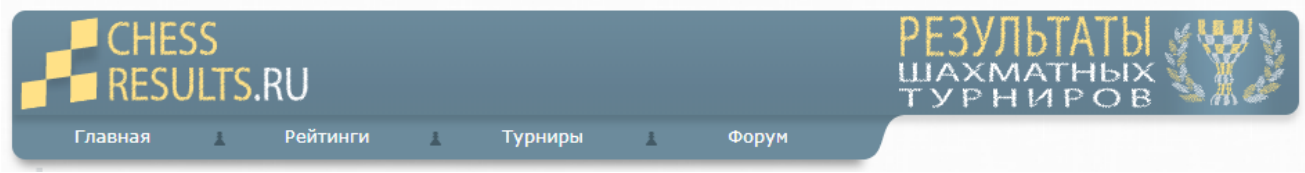

Рисунок 1.8 – Титульна сторінка сайту

Наступним конкурентом є закордонний аналог десктопна програма яку можна скачати з сайту swiss-chess.de. (рис. 1.9). Задачі програми типові сітка та жеребкування для турнірів з шахмат. Спокійний нічим не примітний інтерфейс та солідний з надилшнім набором фукцій які просто не потрібні. Плюси це мультизадачність та велика підтримка команд. Мінуси це платна програма і не інтересний інтерфейс.

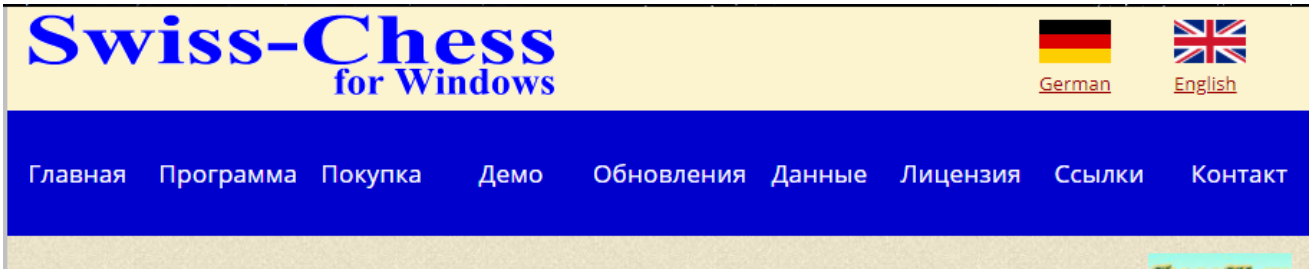

Рисунок 1.9 – Титульна сторінка сайту

Прирівнявши два об’єкта, виплили певні переваги моєї програми над іншими представленішими вище, а саме доступність, чіткість та безкоштовність.

1.3 Вимоги, обгрунтування середовища та постанова задачі

Вибір стеку технологій є одним із найважливіших етапів розробки будь-якого сайту чи додатку. Тому варто приділити цьому достатню кількість часу, зробити дослідження, проаналізувати доступні варіанти і вибрати найкращі технології, які чудово справлятимуться із поставленими задачами.

Важливі критерії при виборі стеку технологій:

- розмір і тип проекту, тут варто враховувати призначення проекту, потенційну аудиторію і т.д.;
- складність проекту.

Прості проекти це зазвичай сайти візитки, лендінги, прості інтернет-магазини, прості програми. Для таких типів проектів зазвичай вже є готові рішення - CMS або шаблони. Середні проекти - складні інтернет-магазини і маркетплейси, портали національного масштабу, різноманітні сервіси. Зазвичай використовуються фреймворки. Складні - величезні портали, соціальні мережі, інноваційні та нетипові рішення. Ядро таких проектів зазвичай розробляється на чистій (нативній) мові програмування:

- швидкість розробки, тобто терміни, які відведені на розробку проекту;
- вартість, слід аналізувати бюджет відведений під даний проект;
- доступні інструменти розробки, вибираючи технології перш за все.

Слід звертати увагу на те наскільки добре документований фреймворк, інструмент чи бібліотека. Адже обгрунтована, повна документація дозволяє легко знаходити методи розв’язування необхідних задачі чи виправляти помилки, з якими доводиться зустрічатися в процесі розробки;

- наявність готових рішень, якщо для поставленої задачі вже існує готове рішення, то постає питання чи варто взагалі створювати все з нуля, якщо можна використати вже розроблений алгоритм;
- вартість підтримки, якщо для розробки проекту використовуються платні сервіси, варто обрахувати вартість підтримки такого ресурсу в довготривалій перспективі;
- вимоги до навантажень, приблизна оцінка кількості відвідувань за день.

В залежності від навантаження необхідно вибрати відповідний сервер, щоб забезпечити безперервну роботу і запобігти перенавантаженню сервера:

- вимоги до безпеки, слід оцінити рівень важливості і конфіденційності особистих даних і вжити необхідних заходів для їхнього захисту;
- кросплатформеність, потреба у використанні розробленого продукту різними типами операційних систем і т.д.;

- можливості інтеграції з іншими рішеннями, забезпечує можливість розвитку і вдосконалення розробленого проекту в майбутньому.

Вибираючи технологію за такими критеріями, можливо домогтися об'єктивного вибору і тим самим заощадити час і гроші.

Для розробки даного програмного забезпечення було обрано наступний стек технологій:

Основні технології:

- Microsoft Visual Studio 2017 (версія 15.1), це інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Що дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms [2];
- Windows Forms – інтерфейс програмування додатків (API), відповідальний за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за допомогою створення обгортки для Win32 API в керованому коді [3];
- Microsoft .NET (версія 4.7), це програмна технологія, запропонована фірмою Microsoft як платформа для створення як звичайних програм, так і веб-застосунків [4];
- NET Core це безкоштовний крос-платформний фреймворк з керованим кодом підтримуваний на Windows [5];

Бібліотеки:

- MSDN Library – бібліотека офіційної технічної документації для розробників під ОС Microsoft Windows. MSDN розшифровується як Microsoft Developer Network. Бібліотека MSDN містить документацію на API продуктів Microsoft, а також код прикладів, технічні статті та іншу корисну для розробників інформацію [6].

C# – є одним з найбільш затребуваних (рис. 1.10), багатофункціональних і активно розвивающихся мов програмування на даний момент. З його допомогою

можна розробляти практично будь-яке програмне забезпечення, починаючи від простих windows forms додатків, до великих клієнт-серверних веб-додатків або навіть мобільні додатки і комп'ютерних ігор.

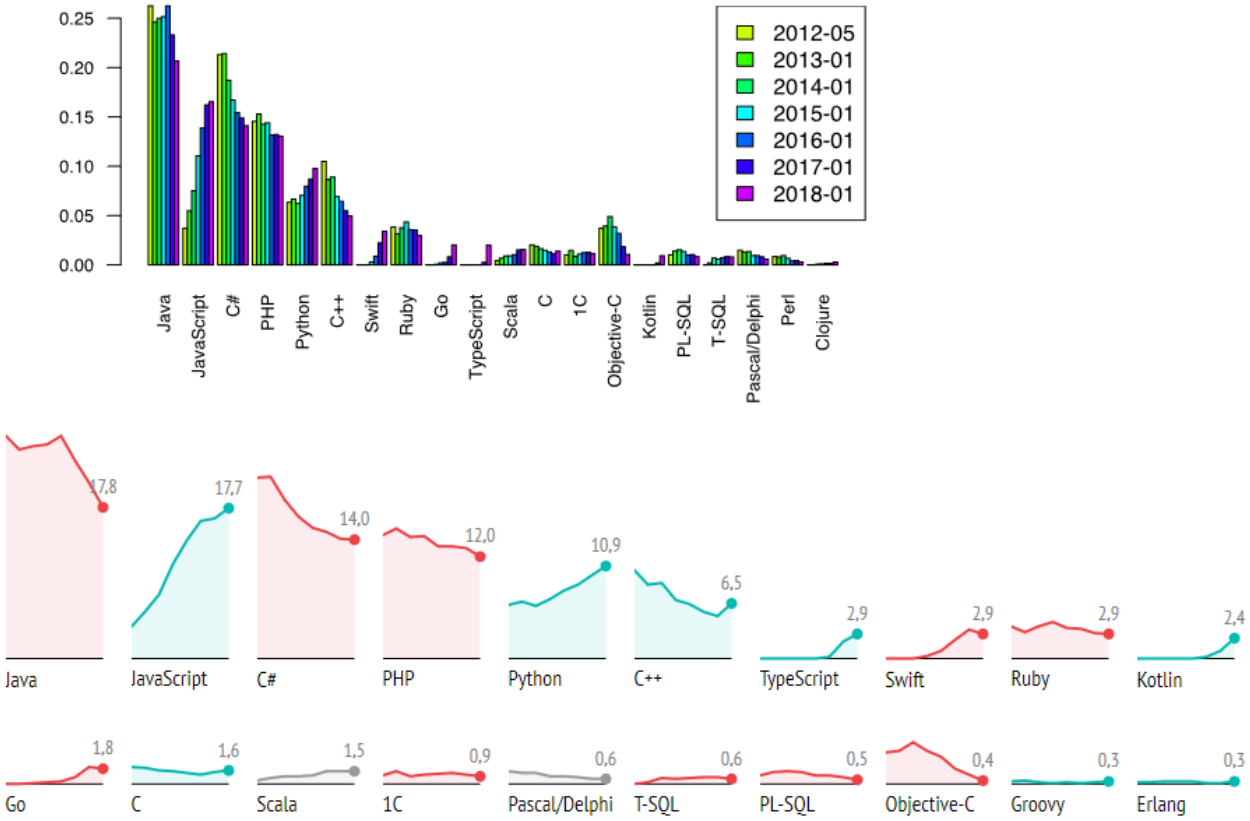

Рисунок 1.10 – Шкала затребувних мов програмування

Об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність буде порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких

позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо. Так як C # є об'єктно-орієнтованою мовою, то він підтримує спадкування, поліморфізм, інкапсуляцію, строгу типізацію змінних, перевантаження операторів і інше. Всі ці поняття будуть нами детально розглянуті в наступних статтях. Завдяки використанню парадигми об'єктно-орієнтованого проектування за допомогою мови програмування можна досить легко розробляти масштабні і при цьому гнучкі проекти. При цьому регулярно виходять нові версії мови C #, додаючи нову функціональність для спрощення життя розробника, збільшення швидкості розробки, підвищення продуктивності і надійності програми.

.NET Framework – це програмна платформа від компанії Microsoft дозволяє виконувати розробку на різних мовах програмування, так як для всіх мов використовується єдина загальномовне середовище виконання Common Language Runtime (CLR). Таким чином, основними можливостями платформи .NET є:

- кросплатформеність – .NET Framework підтримується більшістю сучасних операційних систем Windows, а також є можливість створювати додатки підтримувані Linux системами, і навіть мобільні додатки;
- багатомовність - так як вихідний код, написаний на використовувану мову програмування, транслюється в загальномовна Common Intermediate Language (CIL) код, з'являється можливість вести розробку на будь-якому підтримуваному мовою програмування, і навіть використовувати різні мови програмування в одному рішенні. Найбільш популярними підтримуваними мовами є C #, VB.NET, C ++, F #;
- велика бібліотека класів і технологій - існує величезна кількість готових до використання бібліотек для вирішення необхідних завдань. Найчастіше зовсім не доводиться реалізовувати низкоуровневую логіку роботи програми, досить скористатися готовим рішенням, зручно поставляється через менеджер пакетів nuget. Крім того, платформа .NET

передбачає легку розробку практично будь-якого типу необхідного програмного забезпечення [7].

Приклад оголошення і використання класу в C#:

```
class Child
{
    private int age;
    private string name;

    // Default constructor:
    public Child()
    {
        name = "N/A";
    }

    // Constructor:
    public Child(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Printing method:
    public void PrintChild()
    {
        Console.WriteLine("{0}, {1} years old.", name, age);
    }
     }
```

Клас – це абстрактний тип даних. Іншими словами, клас– це деякий шаблон, на основі якого будуть створюватися його екземпляри - об'єкти. У C# класи оголошуються за допомогою ключового слова class. При оголошенні класу модифікатор доступу можна не вказувати, при цьому застосовуватиметься режим за замовчуванням internal. Класи в C# можуть містити наступні члени – поля, константи, властивості, конструктори, методи, події, оператори, індексатори, вкладені типи. Всі члени класу, як і сам клас, мають свій рівень доступу. Тільки у членів їх може бути вже п'ять:

- public – доступ до члена можливий з будь-якого місця однієї збірки, або з іншої збірки, на яку є посилання;
- protected – доступ до члена можливий тільки усередині класу, або в класі-спадкоємця (при спадкуванні);

- internal – доступ до члена можливий тільки з збірки, в якої він проголошений;
- private – доступ до члена можливий тільки всередині класу;
- protected internal – доступ до члена можливий з однієї збірки, або з класу-спадкоємця інший збірки [8].

Нові класи створюються за допомогою оголошень класів. Оголошення класу починається з заголовка, в якому вказані атрибути і модифікатори класу, ім'я класу, базовий клас (якщо є) і інтерфейси, реалізовані цим класом. За заголовком між роздільниками { і } слід тіло класу, в якому послідовно оголошуються всі члени класу. Примірники класів створюються за допомогою оператора new, який виділяє пам'ять для нового екземпляра, викликає конструктор для ініціалізації цього примірника і повертає посилання на екземпляр. Займана об'єктом пам'ять автоматично звільняється, коли об'єкт стає недоступний. У C# немає ні необхідності, ні можливості звільняти пам'ять об'єктів явно. Визначення класу може задати набір параметрів. Список імен параметрів типу вказується в кутових дужках після імені класу. Параметри типу можна використовувати в тілі класу в визначеннях, що описують члени класу. Тип класу, для якого оголошені параметри типу, називається універсальним типом класу. Типи структури, інтерфейсу і делегата також можуть бути універсальними. Якщо ви використовуєте універсальний клас, необхідно вказати аргумент типу для кожного параметра типу. Універсальний тип, для якого вказані аргументи типу, як Pair <int, string> називається сконструйованим типом. В оголошенні класу можна вказати базовий клас, включивши ім'я базового класу після імені класу і параметрів типу, і відокремивши його двокрапкою. Якщо специфікація базового класу не вказана, клас успадковується від типу object. Клас успадковує члени базового класу. Спадкування означає, що клас неявно містить всі члени свого базового класу, за винятком конструкторів примірника, статичних конструкторів і методів завершення базового класу. Похідний клас може доповнити успадковані елементи новими елементами, але він не може видалити визначення для успадкованого члена. Використовується неявне перетворення з типу класу до

будь-якого з типів відповідного базового класу. Це означає, що змінна типу класу може посилатися як на екземпляр цього класу, так і на екземпляри будь-якого похідного класу [9].

Наступним етапом буде підключення бази даних. Для розробки додатків, підключених до даних, в Visual Studio, зазвичай установка системи бази даних на локальному комп'ютері розробки і потім розгорнути додаток і база даних в робочому середовищі, коли вони будуть готові. Visual Studio встановлює SQL Server Express LocalDB на локальному комп'ютері як частину зберігання і обробка даних робочого навантаження. Цей екземпляр LocalDB можна використовувати для розробки додатків, підключених до даних, швидко і легко. Visual Studio і .NET Framework разом надають великий API і підтримка засобів для підключення до баз даних, моделювання даних в пам'яті і відображення даних в інтерфейсі. Класи .NET Framework, які надають функціональні можливості доступу до даних, називаються ADO.NET. ADO.NET, разом з даними, інструменти Visual Studio була розроблена в основному для підтримки реляційних баз даних і XML. Наші дні багато постачальників бази даних NoSQL або сторонніх розробників, пропонують постачальників ADO.NET. Об'єкт DataSet об'єкт - це об'єкт в пам'яті, який по суті міні-база даних. Він містить DataTable, DataColumn, і DataRow об'єктів, в яких можна зберігати і змінювати дані з одного або декількох баз даних без необхідності відкритих з'єднань. Набір даних зберігає відомості про зміни в даних, тому поновлення може відстежити і відправляється в базу даних, коли програма стає повторно підключений [10].

Набори даних і родинні класи, визначені в System.Data простору імен в бібліотеці класів .NET Framework. Ви можете створювати і змінювати набори даних динамічно в коді за допомогою ADO.NET. Документація в цьому розділі показано, як працювати з наборами даних за допомогою конструкторів Visual Studio. Набори даних, які створюються за допомогою конструкторів TableAdapter об'єктів для взаємодії з базою даних. Набори даних, які створюються програмно використовувати DataAdapter об'єктів. Відомості про створення наборів даних програмними засобами, див. Розділ об'єкти DataAdapter і DataReader. Якщо

додатки повинен тільки зчитувати дані з бази даних, а не виконання оновлень, додає або видаляє, зазвичай можна отримати більш високу продуктивність за допомогою DataReader об'єкта для отримання даних в універсальний List об'єкта або інший об'єкт в колекції. При відображенні даних, можна здійснити даних прив'язку інтерфейсу користувача в колекцію [11].

Приклад підключення бази даних :

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        LoadData();
    }

    private void LoadData()
    {
        string connectString = "Data Source=.\\SQLEXPRESS;Initial
Catalog=LSTU_Schedule_autumn20172018;" +
            "Integrated Security=true;";

        SqlConnection myConnection = new SqlConnection(connectString);

        myConnection.Open();

        string query = "SELECT * FROM Faculty ORDER BY fac_id";

        SqlCommand command = new SqlCommand(query, myConnection);

        SqlDataReader reader = command.ExecuteReader();
```

Також для створення програми використовувався графічний редактор. Для створення графічних інтерфейсів за допомогою платформи .NET застосовуються різні технології - Window Forms, WPF, додатки для магазину Windows Store (для ОС Windows 8 / 8.1 / 10). Однак найбільш простий і зручною платформою досі залишається Window Forms або форми [12].

Windows Forms дозволяє розробляти інтелектуальні клієнти. Інтелектуальний клієнт – це програма з повнофункціональним графічним інтерфейсом, просте в розгортанні і оновленні, здатне працювати при наявності або відсутності підключення до Інтернету і використовує більш безпечний доступ

до ресурсів на локальному комп'ютері в порівнянні з традиційними додатками Windows. Windows Forms - це технологія інтелектуальних клієнтів для платформи .NET Framework, набір керованих бібліотек, що спрощують виконання поширених завдань, таких як читання і запис в файлової системі. При використанні середовища розробки, як Visual Studio, можна створювати додатки інтелектуальних клієнтів Windows Forms, які відображають інформацію, запитують введення від користувачів і взаємодіяти з віддаленими комп'ютерами по мережі. У Windows Forms форма – це візуальна поверхню, на якій виводиться інформація для користувача. Зазвичай додаток Windows Forms будується шляхом приміщення елементів управління на форму і написання коду для реагування на дії користувача, такі як клацання миші або натискання клавіш. Елемент управління – це окремий елемент призначеного для користувача інтерфейсу, призначений для відображення або введення даних. При виконанні користувачем якої-небудь дії з формою або одним з її елементів управління створюється подія. Додаток реагує на ці події за допомогою коду і обробляє події при їх виникненні.

Windows Forms включає широкий набір елементів управління, які можна додавати на форми: текстові поля, кнопки, списки, що розкриваються, перемикачі та навіть веб-сторінки. Список всіх елементів управління, які можна використовувати в формі, представлені в розділі Елементи керування для використання в формах Windows Forms. Якщо існуючий елемент управління не задовольняє потребам, в Windows Forms можна створювати власні елементи управління за допомогою класу UserControl.

До складу Windows Forms входять багатофункціональні елементи призначеного для користувача інтерфейсу, що дозволяють відтворювати можливості таких складних додатків, як Microsoft Office. Використовуючи елементи управління ToolStrip і MenuStrip, можна створювати панелі інструментів і меню, що містять текст і малюнки, підміню і інші елементи управління, такі як текстові поля і поля зі списками.

За допомогою перетягування і вставки конструктор Windows Forms в Visual Studio можна легко створювати додатки Windows Forms. Досить виділити елемент

керування курсором і помістити його в потрібне місце на формі. Для подолання труднощів, пов'язаних з вирівнюванням елементів управління, конструктор надає такі кошти, як лінії сітки і лінії прив'язки. За допомогою Visual Studio або компіляції з командного рядка, можна використовувати FlowLayoutPanel, TableLayoutPanel і SplitContainer елементи управління для створення складних макетів форм за менший час [13].

Приклад підключення Windows Froms :

```
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

2 РОЗРОБКА АЛГОРИТМІЧНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Структура програмного забезпечення

Створення якісного продукту, сайту чи додатку, вимагає ретельного планування, чітких специфікацій і якісно виконаної роботи. Перш за все додаток має бути зручним у користуванні (простий, зрозумілий інтерфейс), швидким (оперативно обробляти вхідні і вихідні запити), і повинен працювати коректно (видавати чіткі правильні результи).

Десктоп програма включає в себе код якій її запускає та сприяє її підримці і функціоналу. Програма починається з задання шаблоблону віндовс форми та підключення кнопок для віндовс форми. Потім іде дизайн (розположення кнопок та боксів). І сам алгоритм який буде рахувати систему за мірою потрбіності також кінцевий обрахунок і збереження даних.

Програма вимагає для своєї реалізації чітко подане завдання яке можна реалізувати за допомогою конструктора Visual Studio. В цій програмі зібрані всі необхідні інструменти та плагіни. Програма задовільнить любе побажання користувача чи то підключити базу даних чи перетворити все в веб контент за допомогою пари маніпуляцій.

Для написання якісної програми необхідно мати великий набір навичок. Для першого рівня роботи це:

- навички роботи з wind. form щоб забезпечити естетичний вигляд для програми;
- основи мови програмування C# та повного конролю всіх додаткових функцій які є не відємною частиною програми;
- головні принципи та орієнтація в .NET;

На теперішній час є багато аналогів для Visual Studio наприклад IntelliJ IDEA в якій використовується мова програмування Java і вибір між двома

гігантами фреймворку для мене в очевидь є Visual Studio завдяки якій можна краще оптимізовувати свій час та програми. Останні часи C# являється одним з незамінних і необхідних інструментів для багатьох розробників. Зараз сфера його застосування розширюється на інші області, такі як сервери, мікроконтролери та багато іншого. Цю мова програмування вибрають престижні університети, щоб навчати студентів основам інформатики.

Головними вимогами до розробки даного продукту є:

- зручний інтерфейс, який надасть можливість почати використання програми вже після першого знайомства;
- максимальна швидкість обробки даних, для забезпечення оперативної обробки вхідних і вихідних запитів;
- коректна робота і видавати чітких релевантних результатів;
- стабільність роботи, забезпечення умов для безперебійного функціонування;
- безпека даних.

Основні функціональні можливості:

- основи crud, 4 базові функції управління даними «створення, зчитування, зміна і видалення;
- жеребкування, випадковий вибір суперників для ігри в груповому етапі;
- груповий етап в якому будуть відображені переможці які пройшли далі і проігравші які вибули;
- плей-офф, команди будуть продовжувати боротьбу за чемпіонство.

Для ствоерння інтерактивного інтерфейса знадобиться windows forms це буде лице програми як відбувається створення першого вікна за допомогою різних інтсрументів можна додавати функціонал цього вікна. Створену форму можна налаштувати на любий лад та зробити її естетично красивою та надати весь функціонал де заманеться а найкраще це його оптимізація та робота яку він виконує і зразу все можна вставити та підключити.

Також окрім управлінням самою формою можна використовувати автоматичні кнопки та текстові бокси а ще багато іншого що знаходиться в панелі елементів. Панель елементів полегшує роботу, а саме головне оптимізує час роботи для використання все що потрібно це правильно його підключити. Visual Studio надає повний спектр потрібних ресурсів як і контейнерів так і меню панелів та ввід данних. Всі інструменти являють собою надзвичайно важливі і необхідні помічники в роботі. Працює ця система доволі легко і зрозуміло потрібний елемент можна перетягнути на форму і задати потрібне управління завдяки системі управління формою і задати код, щоб цей інструмент щось виконував нажати на нього подвійним кліком відкриється приватна строчка де можна буде задати функціонал який буде необхідний для поставленої задачі.

Після вибору правильних елементів для виконання роботи, а саме текстбокси та кнопки дії для програмування проекту. Головний плюс панелі можна по експерементувати з різними типами кнопок та елементів. Вся структура програми складається з елементів та їх розміщення і реалізації в чому дуже сильного допомагає Visual Studio.

Як й інші традиційні інженерні дисципліни, розробка програмного забезпечення має справу з проблемами якості, вартості та надійності. Деякі програми містять мільйони рядків вихідного коду, які, як очікується, повинні правильно виконуватися в умовах, що змінюються. Складність ПЗ порівнянна зі складністю найбільш складних з сучасних машин, таких як літаки.

Розробка програмного забезпечення може бути розділена на кілька розділів. Це:

- Вимоги до програмного забезпечення: витяг, аналіз, специфікація та ратифікація вимог для програмного забезпечення;
- Проектування програмного забезпечення: проектування програмного забезпечення засобами Автоматизованої Розробки Програмного Забезпечення (CASE) і стандарти формату описів, такі як Уніфікований Мова Моделювання (UML), використовуючи різні підходи: проблемно-орієнтоване проектування і т. д..

- Інженерія програмного забезпечення: створення програмного забезпечення за допомогою мов програмування;
- Тестування програмного забезпечення: пошук та виправлення помилок у програмі;
- Обслуговування програмного забезпечення: програмні системи часто мають проблеми сумісності та переносимості, а також потребують подальших модифікацій протягом довгого часу після того, як створена їх перша версія;
- Керування конфігурацією програмного забезпечення: оскільки системи програмного забезпечення дуже складні та модифікуються в процесі експлуатації, їх конфігурації повинні управлятися стандартизованим та структурованим методом;
- Керування розробкою програмного забезпечення: керування системами програмного забезпечення має запозичення з керування проектами, але є нюанси, що не трапляються в інших дисциплінах керування;
- Процес розробки програмного забезпечення: процес побудови програмного забезпечення гаряче обговорюється серед практиків, основними парадигмами вважаються agile або waterfall;
- Інструменти розробки програмного забезпечення, див. CASE: методика оцінки складності системи, вибору засобів розробки та застосування програмної системи;
- Якість програмного забезпечення: методика оцінки критеріїв якості програмного продукту та вимог до надійності;
- Локалізація програмного забезпечення, гілка мовної промисловості.

Протягом кількох десятиліть стоїть завдання пошуку повторюваного, передбачуваного процесу або методології, яка б поліпшила продуктивність, якість і надійність розробки. Одні намагалися систематизувати та формалізувати цей, мабуть, малопередбачуваний процес. Інші застосовували до нього методи управління проектами та методи програмної інженерії. Треті вважали, що без постійного контролю з боку замовника розробка ПЗ виходить з-під контролю, з'їдаючи зайвий час і кошти. Досвід управління розробкою програм відбивається

у відповідних посібниках, звичаях і стандартах. Якщо при розробці використовується декілька стандартів і нормативних документів, то має сенс скласти профіль. Інформатика як наукова дисципліна пропонує і використовує на базі методів структурного програмування технологію надійної розробки програмного забезпечення, використовуючи тестування програм та їх верифікацію на основі методів доказового програмування для систематичного аналізу правильності алгоритмів і розробки програм без алгоритмічних помилок. Дана методологія спрямована на вирішення завдань на ЕОМ, аналогічної технології розробки алгоритмів і програм, використовуваної на олімпіадах з програмування вітчизняними студентами та програмістами з використанням

тестування і структурного псевдокоду для документування програм в корпорації IBM з 70-х років. Методологія структурного проектування програмного забезпечення може використовуватися з застосуванням самих різних мов і засобів програмування для розробки надійних програм самого різного призначення. Одним з таких проектів була розробка бортового програмного забезпечення для космічного корабля «Буран», в якому вперше використовувався бортовий комп'ютер для автоматичного управління апарату, яка виконала успішний старт і посадку космічного корабля. При виборі методології розробки програмного забезпечення слід керуватися тим, що складність методології порівнянна з складністю структури програмного продукту, і невиправдана для продукту даної складності складність методології тільки невиправдано збільшить вартість розробки. Прикладом сучасної методології проектування може бути проблемно-орієнтоване проектування.

Найбільш поширеними проблемами, що виникають в процесі розробки ПЗ, вважають:

- Недолік прозорості. У будь-який момент часу складно сказати, в якому стані знаходиться проект і який відсоток його завершення. Дана проблема виникає при недостатньому плануванні структури (чи архітектури) майбутнього програмного продукту, що найчастіше є наслідком відсутності достатнього фінансування проекту: програма потрібна, скільки часу займе розробка, якими є

етапи, чи можна якісь етапи виключити або заощадити – наслідком цього процесу є те, що етап проектування скорочується;

- Недолік контролю. Без точної оцінки процесу розробки зриваються графіки виконання робіт і перевищуються встановлені бюджети. Складно оцінити обсяг виконаної роботи і роботи, що залишилася;
- Дана проблема виникає на етапі, коли проект, завершений більш ніж наполовину, продовжує розроблятися після додаткового фінансування без оцінки ступеня завершеності проекту;
- Недолік трасування;
- Недолік моніторингу. Неможливість спостерігати хід розвитку проекту не дозволяє контролювати хід розробки в реальному часі. За допомогою інструментальних засобів менеджери проектів приймають рішення на основі даних, що надходять в реальному часі;
- Дана проблема виникає в умовах, коли вартість навчання менеджменту володінню інструментальними засобами порівнянна з вартістю розробки самої програми;
- Неконтрольовані зміни. У споживачів постійно виникають нові ідеї щодо розроблюваного програмного забезпечення. Вплив змін може бути суттєвим для успіху проекту, тому важливо оцінювати пропоновані зміни та реалізовувати тільки схвалені, контролюючи цей процес за допомогою програмних засобів;
- Дана проблема виникає внаслідок небажання кінцевого споживача використовувати ті чи інші програмні середовища. Наприклад, коли при створенні клієнт-серверної системи споживач висуває вимоги не тільки до операційної системи на комп'ютерах-клієнтах, а й на комп'ютері-сервері;
- Недостатня надійність. Найскладніший процес – пошук і виправлення помилок у програмах на ЕОМ. Оскільки число помилок у програмах заздалегідь невідомо, то заздалегідь невідома і тривалість налагодження програм і відсутність гарантій відсутності помилок в програмах. Слід зазначити, що залучення доказового підходу до проектування ПЗ дозволяє виявити помилки в програмі до її виконання. У цьому напрямку багато працювали Кнут, Дейкстра і Вірт.

Професор Вірт при розробці Паскаля і Оберона за рахунок строгості їх синтаксису домігся математичної доказовості виконання і правильності програм, написаної на цих мовах;

- Дана проблема виникає при неправильному виборі засобів розробки. Наприклад, при спробі створити програму, що вимагає коштів високого рівня, за допомогою засобів низького рівня;
- Наприклад, при спробі створити засоби автоматизації з СУБД на асемблері. У результаті вихідний код програми виходить занадто складним і погано піддається структуруванню;
- Неправильний вибір методології розробки програмного забезпечення. Процес вибору необхідної методології може проблемно відбитися на всіх показниках програмного забезпечення — це його гнучкість, вартість і функціональність. Так звані гнучкі методології розробки допомагають вирішити основні проблеми, однак, варто відзначити, що і каскадна модель (waterfall) так само має свої переваги. У деяких випадках найбільш доцільним буде застосування гібридних методологій у зв'язці Agile + каскадна модель + MSF + RUP і т. д.;
- Відсутність гарантій якості і надійності програм через відсутність гарантій відсутності помилок в програмах аж до формальної здачі програм замовникам.

Дана проблема не є проблемою, що відноситься виключно до розробки ПЗ.

Учасники процесу розробки ПЗ:

- користувач;
- замовник;
- дизайнер;
- керівник проекту;
- аналітик;
- тестувальник;
- постачальник [14].

Також потрібно зрозуміти основні позиції створення інтерфейсу. Як використовувати інтерфейс в одній чи іншій ситуації його гнучкість та

стабільність, надійність і естетичність.

З цим допоможе розібратися елемент управління який називають віджетом.

Десктоп-віджет – це графічний модуль, який встановлюється на робочий стіл вашого комп'ютера. Віджети призначені як інструмент для швидкого доступу до певної інформації чи сервісів. Розробкою віджетів деякий час займалась компанія Konfabulator, що написала основний код з однойменною назвою, відома користувачам, які працюють на комп'ютерах Macintosh. Пізніше віджети стали з'являтися в операційній системі від Microsoft, яку тоді називали Longhorn. Через деякий час віджетоманія охопила світ Windows: віджети були створені для популярного застосунку для локального пошуку Google Desktop, а компанію Konfabulator купила відома компанія Yahoo, щоб створювати персональні віджети, що підтримують її сервіси.

Існує стандартний набір елементів інтерфейсу, що включає такі елементи управління:

- кнопка (button);
- split button (здвоєна кнопка) — кнопка, що викликає список із вторинною дією (кнопками);
- радіокнопка (radio button);
- прапорцева кнопка (check box);
- значок (іконка, icon);
- список (list box);
- дерево — ієрархічний список (tree view);
- Комбінований список (combo box, drop-down list);
- мітка (label);
- поле редагування (textbox, edit field);
- елемент для відображення табличних даних (grid view);
- меню (menu);
- головне меню вікна (main menu або menu bar);
- контекстне меню (popup menu);
- спадне меню (pull-down menu);

- вікно (window);
- діалогове вікно (dialog box);
- модальне вікно (modal window);
- панель (panel);
- вкладка (tab);
- панель інструментів (toolbar);
- смуга прокрутки (scrollbar);
- повзунок (slider);
- рядок стану (status bar);
- спливна підказка (tooltip, hint)

Зустрічаються й інші елементи управління, які можуть не входити в деякі набори :

- Радіальне меню (pie menu або radial menu) — кільцеве меню навколо курсора. Вибір пункту меню здійснюється рухом курсора в напрямку пункту меню;
- Кнопка послідовного вибору — елемент, значення в якому вибирається послідовним натисненням миші по ньому. На відміну від списку, що розкривається, така кнопка не дозволяє бачити інші значення, крім обраного;
- Лічильник — двонаправлений варіант для числових значень . Натискання на кнопку дозволяє змінити значення параметра на одиницю в більшу або меншу сторону;
- Heads — up display — відображення поверх всіх елементів значення якихось параметрів, або важливих повідомлень;
- Бульбашка — підказка, на зразок філактерів в коміксах, яка вказує на елемент — джерело повідомлення;
- Вал-кодер — обертовий елемент управління, на зразок ручки настройки в багатьох радіоприймачах. Може бути як одно- так і багатообертовим;

- Приховуваний віджет – елемент, що дозволяє приховати частину елементів управління, коли вони не використовуються;
- Індикатор рівня (Level Indicator) — елемент для індикації значення якої-небудь величини. Іноді замість нього використовується індикатор процесу, але деякі керівництва (наприклад, HIG від Apple) забороняють подібну практику [15].

2.2 Розробка структури бази даних

База даних находиться і зберігається в Visual Studio (рис. 2.1) і є легко доступною у використанні і швидко доступною у обзорі рішень програми її можна корегувати без зайвих маніпуляцій і швидко обновляти стартові дані і управляти нею.

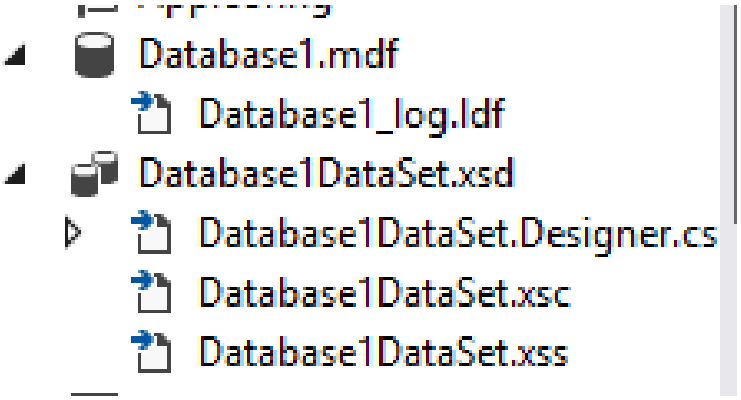

Рисунок 2.1 – Вигляд БД

Для програми не потрібна буде дуже велика база даних але вона повинна вміти чітко вносити та обновляти і видаляти команди з турніру. З точки зору заданої предметної області типовим користувачем розроблюваної бази даних буде адміністратор, тобто людина, яка буде заповнювати БД новими даними про команди або одиночні гравці все залежить від регламенту та типу турінру. Виділимо на основі аналізу предметної області головну колону це буде команди (Teams) і в ньому порядковий номер (ID) та назва (Name).

Таблиця складає основу бази даних - саме в них зберігається всі дані. Перед усім, повинна бути спланована структура таблиці (рис. 2.2). Структура таблиці обумовлюється вмістом тих вихідних форм, запитів та звітів, які повинні бути отримані при роботі з базою даних. При плануванні таблиці необхідно уникати повторення колонок в різних таблицях, тільки якщо вони не слугують для визначення зв'язків між ними.

Таблиця складається з записів (рядків), кожний з яких описує одну сутність. Кожна колонка таблиці - це поле. Поле містить однотипну інформацію, яка визначає тип даних. Тип даних визначає вид і межі допустимих значень, які можуть бути введені в поле, а також об'єм пам'яті, який виділяється для цього поля, що важливо при проектуванні БД.

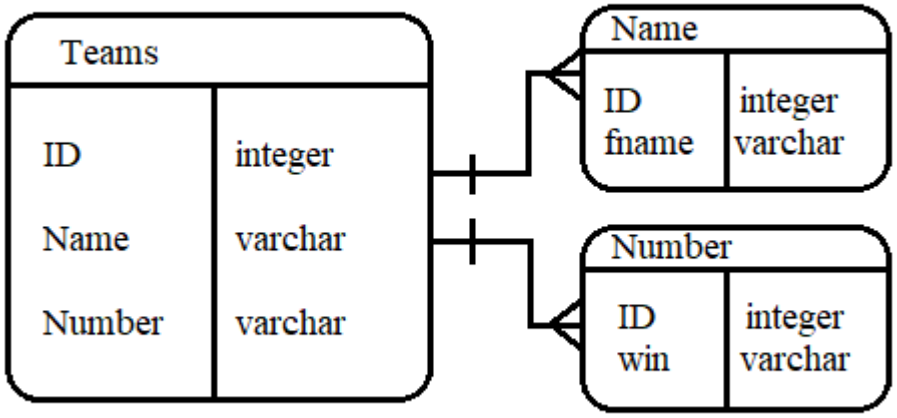

Рисунок 2.2 – Структура БД

Для моєї постанови задачі можна було і справитися і без бази даних але тоді інформація про проведений турнір пропадала при новій реєстрації учасників тому добавляння бази даних є цілком логічним шляхом для спрощення жеребкування.

2.3 Опис зв’язків бази даних

Розробка бази даних за методом сутність-зв'язок (рис. 2.3) складається з наступних етапів:

- визначення сутностей;
- визначення зв'язків;
- визначення атрибутів;
- визначення ключів сутностей;
- визначення ступеня зв'язку;
- визначення класу приналежності.

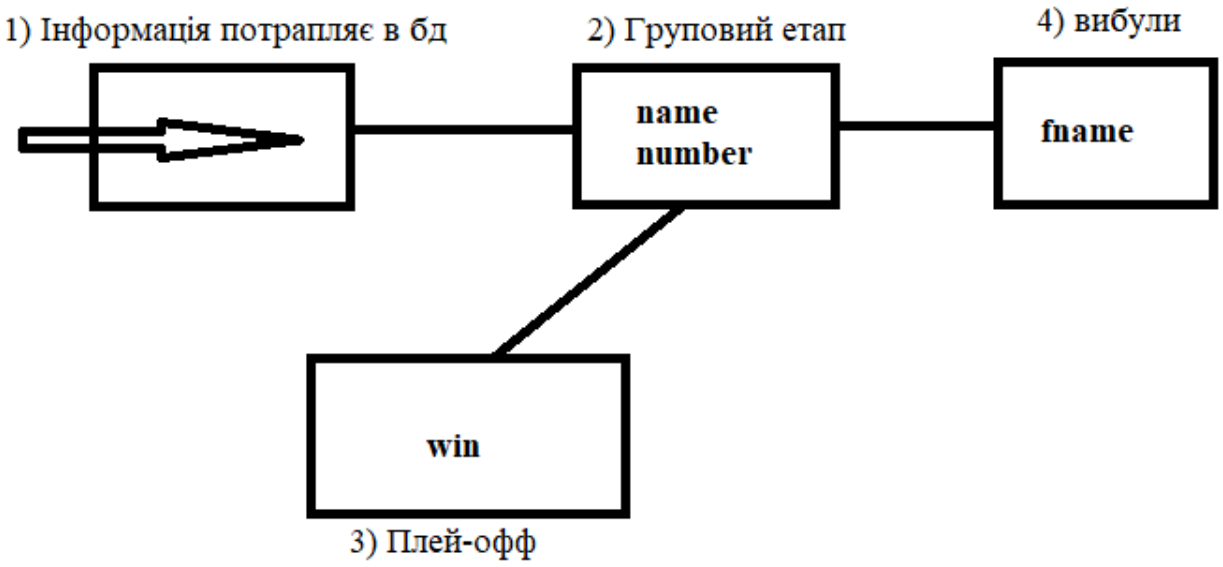

Рисунок 2.3 – Зв'язок між бд

На першому етапі проектування необхідно визначити сутності. Сутність - це якийсь об'єкт, що представляє інтерес для користувача.

Команда (Teams), в ній знаходиться порядковий номер (ID), назва (Name), номер (Number).

Перейдемо до другого етапу - визначення зв'язків між виділеними сутностями.

- Номер (Number) включає порядковий номер (ID);
- Номер (Number) включає переможця (Win);

- Назва (Name) включає проігравших (Fname).

Далі визначимо атрибути або властивості для кожної з сутностей, а так само ключі для кожної сутності. Ключем суті називається атрибут або набір атрибутів, що дозволяють однозначно ідентифікувати примірник сутності:

- Команди (Teams);
- Ідентифікаційний номер (id) – внутрішній ключ;
- Назву;
- Номер (для жеребкування).

Перший і четвертий етапи проектування бази даних за методом сутність-зв'язок вимагають визначити ступінь зв'язку і клас приналежності. Ступінь зв'язку показує скільки безпосередніх зв'язків може мати кожен екземпляр сутності з іншим екземпляром сутності, з яким він пов'язаний. Клас приналежності може бути обов'язковим і необов'язковим, якщо екземпляри даної суті мають брати участь у зв'язку, то участь називається обов'язковим, а якщо екземпляри можуть не брати участь у зв'язку, то необов'язковим. Всі представлені зв'язки є бінарними, так як вони пов'язують тільки дві сутності.

Для кожного бінарного зв’язку отримуються попередні відношення за такими правилами:

- Якщо ступінь бінарного зв’язку дорівнює 1:1 і клас належності обох сутностей є обов’язковим, то потрібно лише одне відношення, первинним ключем якого може бути ключ будь-якого з двох відношень;
- Якщо ступінь бінарного зв’язку дорівнює 1:1 і клас належності однієї сутності є обов’язковим («О»), а другої не обов’язковим («НО»), то необхідно побудувати два відношення. На кожну сутність потрібно виділити одне відношення, при цьому ключ сутності повинен слугжити первинним ключем для відповідного відношення. Крім того ключ сутності, для якої клас належності є «НО», додається в якості атрибуту у відношення, що виділене для сутності з обов’язковим класом належності;
- Якщо ступінь бінарного зв’язку дорівнює 1:1 і клас належності обох сутностей є «НО», то необхідно використовувати три відношення, по

одному для кожної сутності, ключі яких слугуватимуть в якості первинних у відповідних відношеннях та одного відношення для зв’язку серед своїх атрибутів. Відношення, що виділяється для зв’язку буде мати по одному ключу сутності від кожної сутності;

- Якщо ступінь бінарного зв’язку дорівнює 1:N та клас належності однозв`язної сутності є «О», то достатнім є використання двох відношень по одному на кожну сутність за умови, що ключ кожної сутності слугує в якості первинного ключа для відповідного відношення. Додатково ключ N-зв'язної сутності повинен бути добавлений, як атрибут у відношення, що відводиться для однозв`язної сутності;
- Якщо ступінь бінарного зв’язку дорівнює 1:N та клас належності однозв`язної сутності є «НО», то необхідно формувати три відношення: по одному для кожної сутності, при чому ключ кожної сутності служить первинним ключем відповідної сутності та одного відношення для зв’язку. Зв'язок повинен мати серед своїх атрибутів ключ сутності кожної з сутностей;
- Якщо ступінь бінарного зв’язку дорівнює N:N, то для зберігання даних необхідно три відношення: по одному для кожної сутності, використовуючи в якості первинного ключа ключ відповідного відношення та одного відношення для зв'язку. Останнє повинно серед числа своїх атрибутів мати ключ кожної сутності [16].

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНТЕРФЕЙСУ ПРОГРАМИ

3.1 Створення інтерфейсу кристувача

Інтерфейс користувача (рис. 3.1) – засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача; у графічних системах інтерфейс користувача, втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями як самих вікон, так і окремих їх елементів (файли, теки, ярлики, шрифти тощо), доступністю багатокористувацьких налаштувань [17].

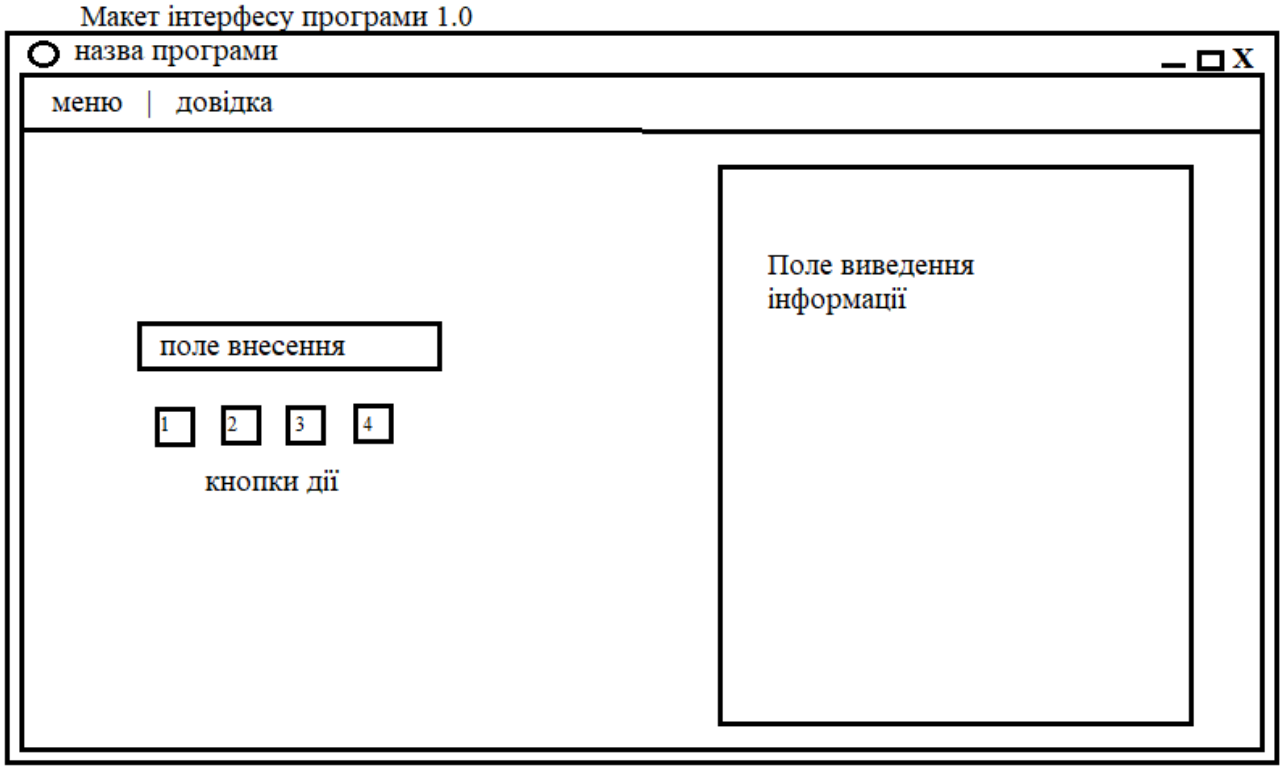

Рисунок 3.1 – Перший етап розробки інтерфейсу (макет)

За віянням моди на 2019 рік править мінімалізм оскільки інтерфейс має приваблювати користувача було використано мінімальні кнопочки іконки та

текстове поле. Для чітко поставленої задачі нам потрібно розставити пріорітети що потрібно на самперед добавляти в інтерфейс яка інформація буде лишня, а яка корисна також інтерфейс має бути дуже зручним і легким в використанні.

Інтерфейс користувача, в галузі промислового дизайну, взаємодії між людиною та комп'ютером – це простір, де відбувається співпраця між людьми та машинами. Мета цієї співдії, полягає у тому, щоби забезпечити досконалу роботу та керування машиною з боку людини, а машина одночасно, надає інформацію, яка допомагає прийняттю рішень операторами. Прикладами цієї широкої концепції інтерфейсів користувача, є: інтерактивні аспекти комп'ютерних операційних систем, ручні інструменти, операторські засоби керування важким обладнанням та елементи контролю над технологічними процесами. Конструктивні міркування, що застосовуються під час створення інтерфейсів користувача, пов'язано з такими галузями, як ергономіка та психологія [17].

Продовжемо констроювання інтерфейсу і другим кроком (рис. 3.2) буде приблизний макет другого вікна програми в якому будуть проходити зустрічі команд або гравців. На макеті зображене поле для матчів і таблиця.

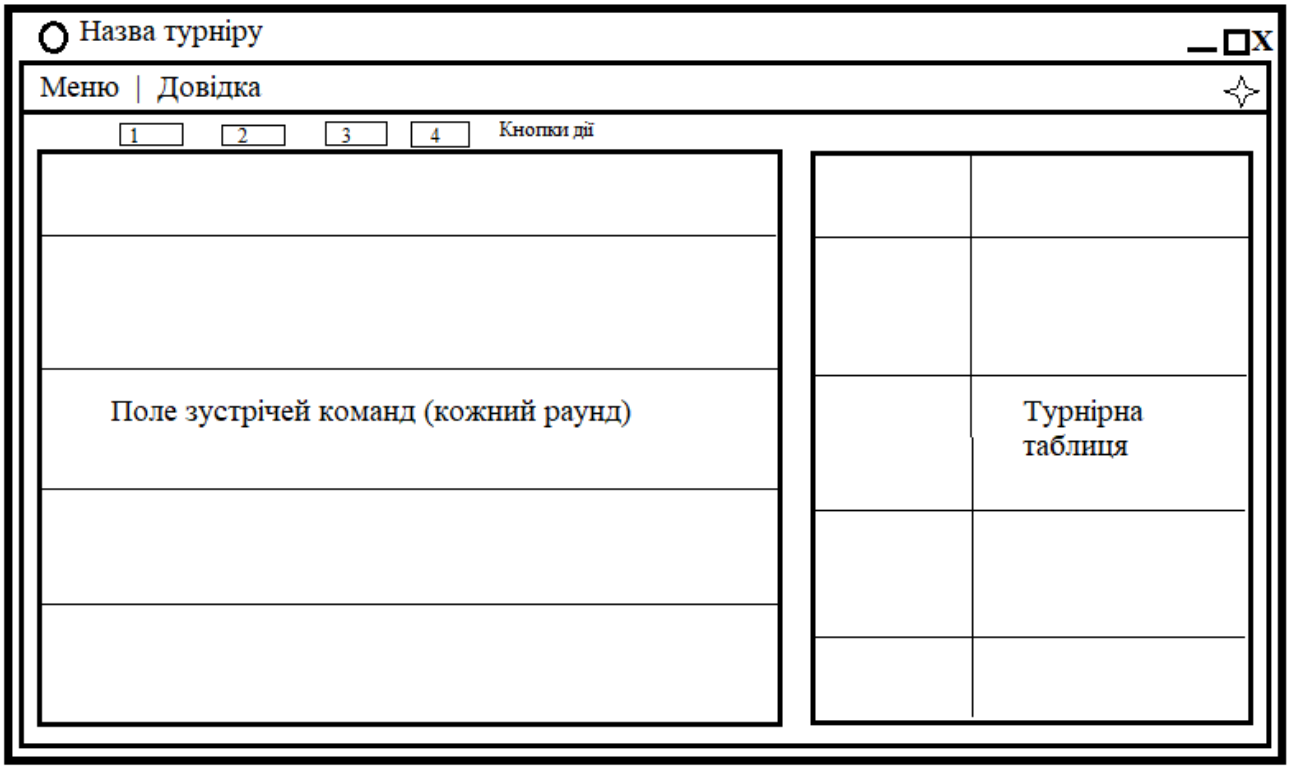

Рисунок 3.2 – Макет інтерфесу програми

Головні функції які повинен містити інтерфейс:

- меню це – елемент інтерфейсу користувача, що дозволяє вибрати одну з декількох перерахованих опцій програми. В сучасних операційних системах меню є найважливішим елементом графічного інтерфейсу користувача [18];
- довідка – допомагає користувачу в складних ситуаціях є в кожній програмі, служить так званим мануалом до програми;
- стрічка для вводу – допомагає задавати інформацію в програму;
- кнопки для дій – виконують ті дії за якими закріплені ;
- вікно занесеної інформації – інформаційний вивід інформації для користувача.

Закінчуємо проектування інтерфейсу програми третій етап (рис. 3.3) це послідне вікно яке буде показувати плей-офф.

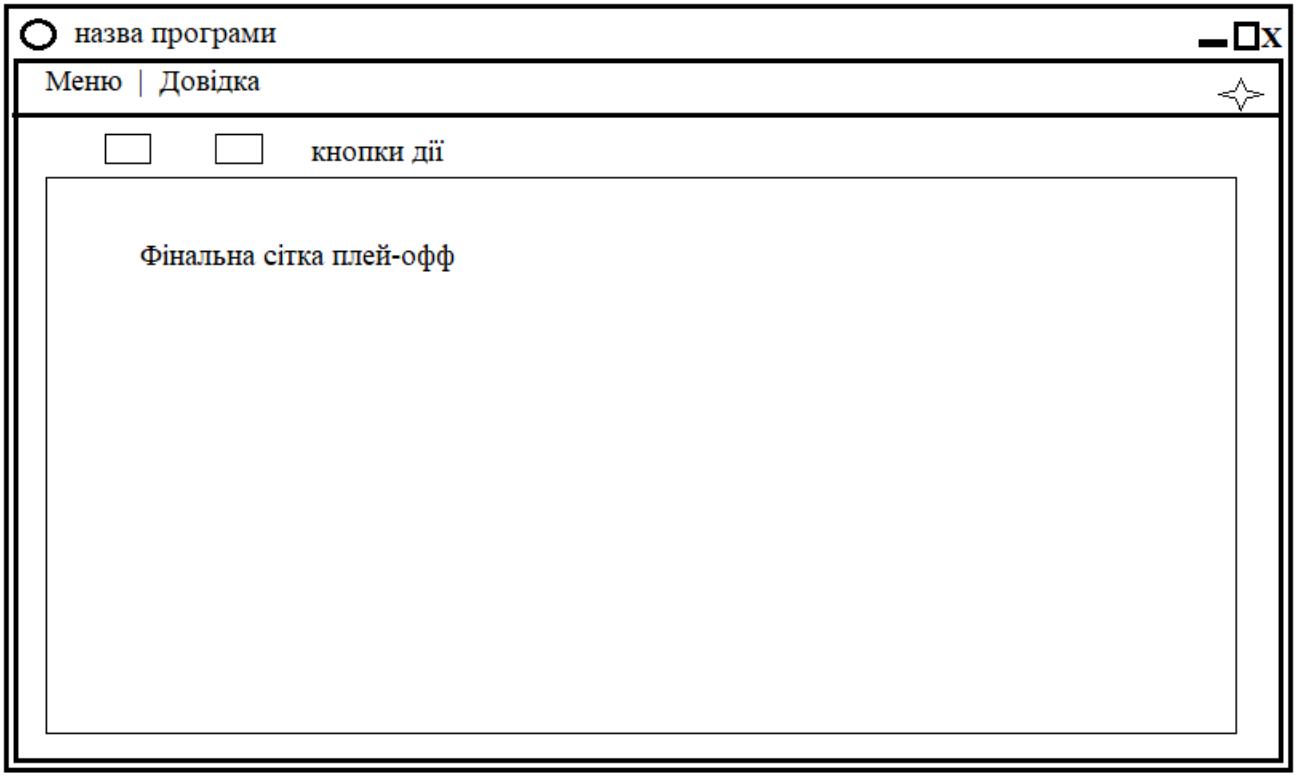

Рисунок 3.3 – Макет фінальної форми для програми

Приступаєм для виконання завданя в Visual Studio. Головне завдання створити доступний інтерфейс який буде допомгата користувачеві розвібратися в кнопках та подальших діях. Тому кожну кнопку прийдеться підписати випливаючим вікном (рис. 3.4) коли наводиш на кнопку чи ділянку програми буде

доступна підсказка в якій буде написано що означає ця кнопка чи поле для чого воно потрібне.

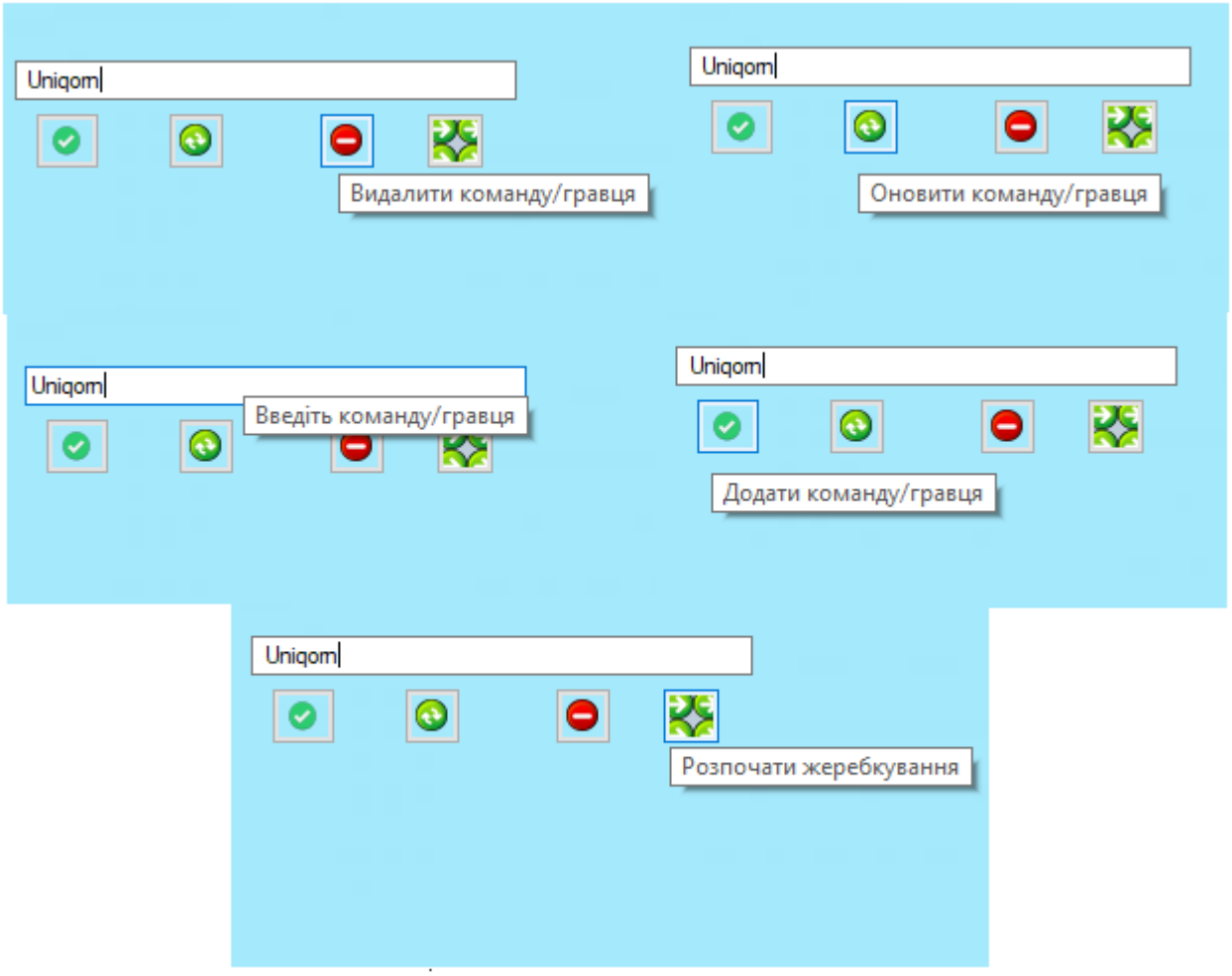

Рисунок 3.4 – Допоміжний інструмент

Нижче представлений код який виконує функцію підказки для певного інструмента якій допомагає орієнтуватися в програмі. Діє він дуже просто на що користувач наводить свій курсор ту інформацію він получає.

```
private void Form1_Load(object sender, EventArgs e)
        {
            ToolTip t = new ToolTip();
            t.SetToolTip(button1, "Додати команду/гравця");
            ToolTip d = new ToolTip();
            d.SetToolTip(button2, "Видалити команду/гравця");
            ToolTip c = new ToolTip();
            c.SetToolTip(button3, "Оновити команду/гравця");
            ToolTip j = new ToolTip();
            j.SetToolTip(button4, "Розпочати жеребкування");
            ToolTip w = new ToolTip();
            w.SetToolTip(textBox1, "Введіть команду/гравця");
```

```
            ToolTip h = new ToolTip();
            h.SetToolTip(listBox1, "Список занесених команд/гравців");
            ToolTip q = new ToolTip();
            q.SetToolTip(listBox2, "Груповий етап");
            ToolTip r = new ToolTip();
            r.SetToolTip(listBox3, "Таблиця");
            ToolTip k = new ToolTip();
            k.SetToolTip(listBox2, "Плей-офф");

             }
```

Як правило, мета створення інтерфейсу користувача, полягає у тому, щоби зробити інтерфейс користувача, який спрощує (самозрозумілий), ефективний і приємний (зручний для користувача) для керування машиною таким чином, щоби забезпечити бажаний результат. Це, зазвичай, означає, що оператор повинен застосовувати щонайменші зусилля для досягнення очікуваного підсумку, а також, щоби машина зменшувала небажані результати для людини.

За допомогою елемента керування TextBox (рис. 3.5) користувач може вводити текст у програмі. Цей елемент керування має додаткові функціональні можливості, які не знайдено в стандартному вікні управління Windows, включаючи багаторядкове редагування і маскування символів пароля. Як правило, елемент керування TextBox використовується для відображення або

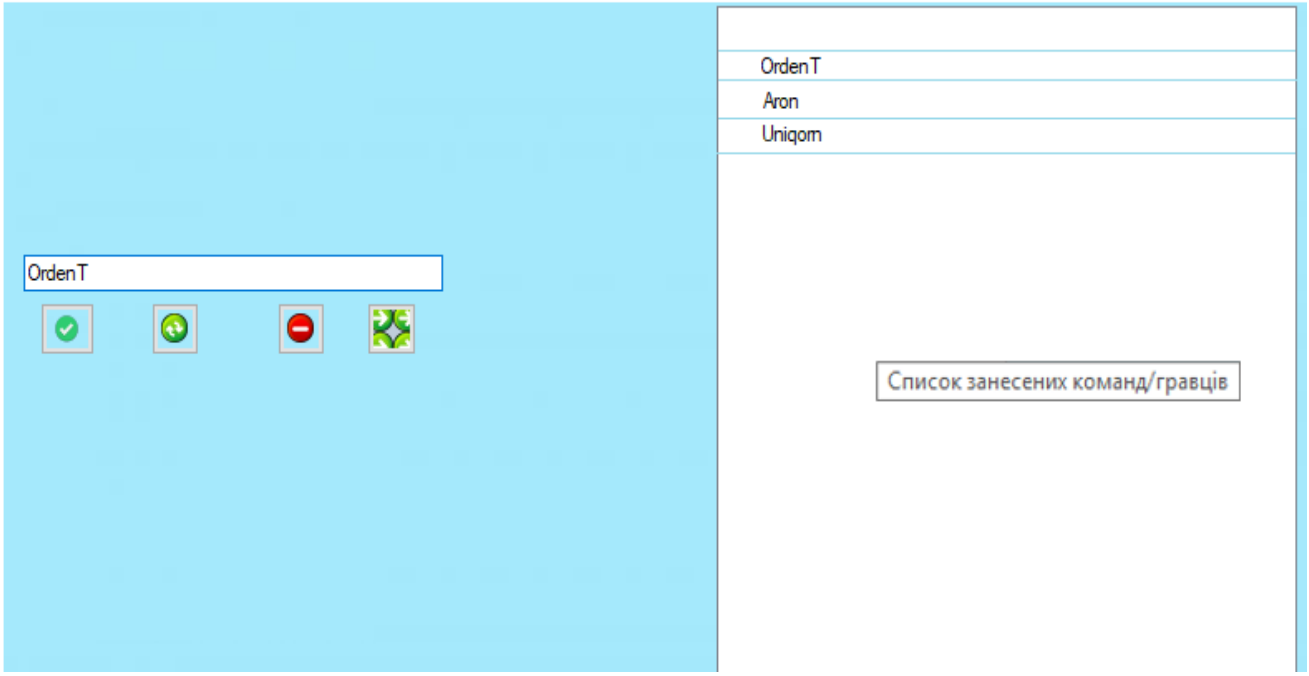

Рисунок 3.5 – Вікно занесеної інформації

прийому в якості одного рядка тексту. Ви можете використовувати властивості Multiline і ScrollBars, щоб дозволити відображення чи введення декількох рядків тексту. Встановіть властивості AcceptsTab і AcceptsReturn у true, щоб увімкнути більшу маніпуляцію текстом у багаторядковому контролі TextBox [19].

Ви можете обмежити кількість тексту, введеного в елемент керування TextBox, встановивши властивість MaxLength на певну кількість символів. Управління TextBox також може використовуватися для прийняття паролів та іншої конфіденційної інформації. Ви можете використовувати властивість PasswordChar для маскування символів, введених в однорядкову версію елемента керування. Використовуйте властивість CharacterCasing, щоб дозволити користувачеві вводити лише верхній регістр, лише малі літери або комбінацію символів верхнього та нижнього регістрів у елемент керування TextBox. Контроль ListBox дозволяє відображати список елементів користувачеві, який користувач може вибрати, натиснувши. Контроль ListBox може забезпечити одно- або кількох виборів за допомогою властивості SelectionMode. ListBox також надає властивість MultiColumn, щоб дозволити відображення елементів у стовпцях замість прямого вертикального списку елементів. При цьому елемент керування може відображати більше видимих елементів, і користувачеві більше не потрібно прокручувати елемент.

Як правило, Windows обробляє завдання малювання елементів для відображення у ListBox. Ви можете використовувати властивість DrawMode і обробляти події MeasureItem і DrawItem, так що ви можете замінити автоматичний малюнок, який надає Windows, і малювати елементи самостійно. Для відображення елементів, зображень або зображень з різними висотами, зображень або іншого кольору або шрифту для тексту кожного елемента списку можна скористатися елементами керування ListBox. Властивість HorizontalExtent, GetItemHeight і GetItemRectangle також допомагають малювати власні елементи.

Окрім функцій відображення та вибору, ListBox також надає функції, які дозволяють ефективно додавати елементи до списку ListBox та знаходити текст у межах списку. Методи BeginUpdate і EndUpdate дозволяють додавати велику

кількість елементів до списку ListBox без повторного копіювання елемента під час додавання елемента до списку. Методи FindString і FindStringExact дозволяють шукати елемент у списку, який містить певний рядок пошуку [20].

Щоб прокрутити вміст TextBox, поки курсор (каретка) не знаходиться у видимій області керування, можна скористатися методом ScrollToCaret. Щоб вибрати діапазон тексту в текстовому полі, можна скористатися методом Select.
Щоб обмежити введення тексту в елемент керування TextBox, можна створити обробник подій для події KeyDown, щоб перевірити, чи кожен символ введено в елемент керування. Ви також можете обмежити вхід даних в елемент керування TextBox, встановивши для властивості ReadOnly значення true.
З посиленням використання персональних комп'ютерів та відносним зниженням обізнаності (інтересу) суспільства про важкі машини, термін «інтерфейс користувача», як правило, передбачає графічний інтерфейс користувача, тоді як промислові панелі керування та механізми контролю за обладнанням, частіше стосуються (HMI) людино-машинної взаємодії. Графічний користувальницький інтерфейс є найбільш популярним UI. Він являє собою вікно, в якому містяться різні елементи управління. Взаємодія користувача з програмою за допомогою миші і за допомогою клавіатури.

Контроль TableLayoutPanel розміщує його вміст у сітці. Оскільки макет виконується як під час розробки, так і під час виконання, він може динамічно змінюватися при зміні середовища програми. Це надає елементам керування на панелі можливість пропорційно змінювати розміри, так що він може реагувати на зміни, такі як зміна розмірів батьківського контролю або зміна довжини тексту через локалізацію.

Будь-який елемент управління Windows Forms може бути дочірнім елемента керування TableLayoutPanel, включаючи інші екземпляри TableLayoutPanel. Це дозволяє створювати складні макети, які адаптуються до змін під час виконання.

Керування TableLayoutPanel може розширюватися, щоб розмістити нові елементи керування, коли вони додаються, залежно від значення властивостей RowCount, ColumnCount і GrowStyle. Встановлення значення RowCount або

ColumnCount значення 0 визначає, що TableLayoutPanel буде незв'язаним у відповідному напрямку.

Ви також можете керувати напрямком розгортання (горизонтальним або вертикальним) після того, як елемент керування TableLayoutPanel наповнений дочірніми елементами керування. За замовчуванням елемент керування TableLayoutPanel розширюється вниз шляхом додавання рядків [21].

Закінчує опису інтерфейсу і його додатків use case diagram (рис. 3.6) відображає повну інформацію про задачу програми.

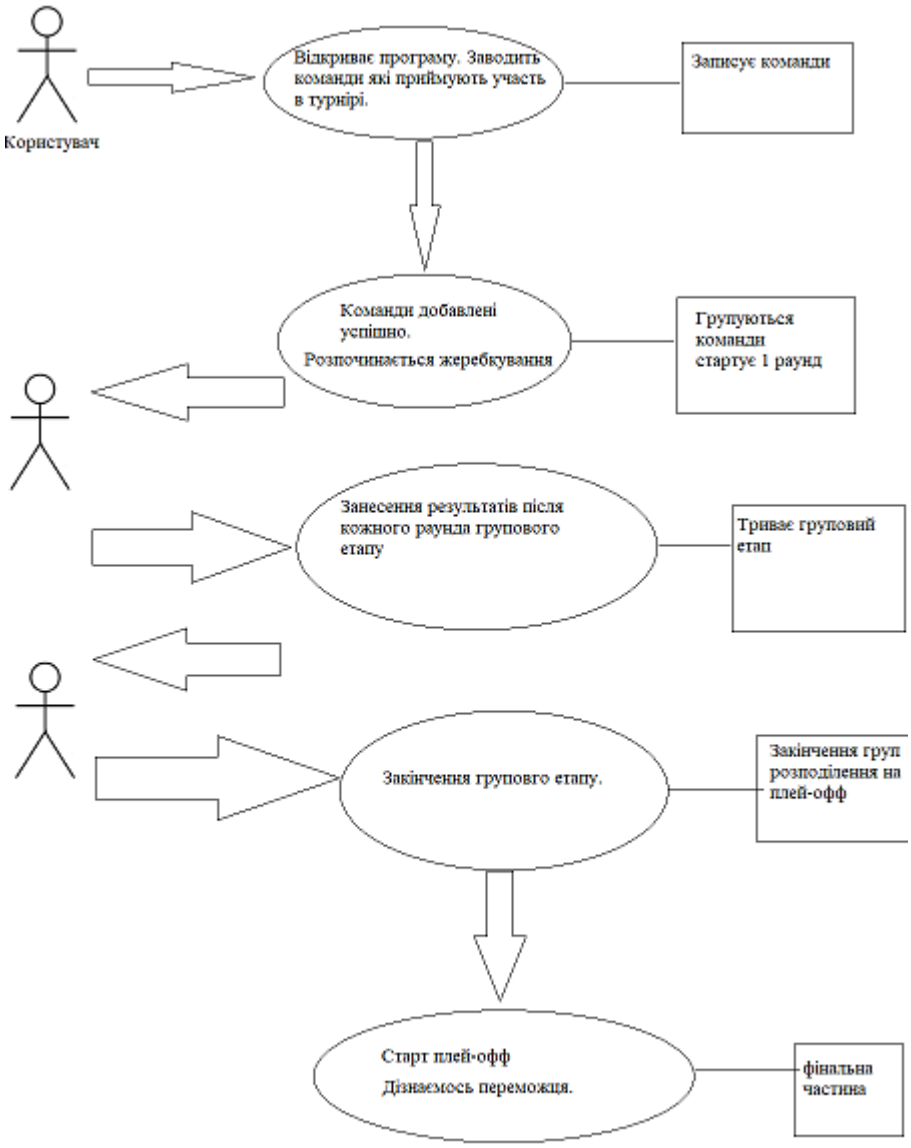

Рисунок 3.6 – Діаграма послідовних подій

Найпростіша діаграма випадків використання являє собою уявлення про взаємодію користувача з системою, яка показує взаємозв'язок між користувачем і різними випадками використання, в яких задіяний користувач. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, а також часто супроводжуватиметься іншими типами діаграм. Випадки використання представлені або колами, або еліпсами.

Незважаючи на те, що сам випадок використання може розробити багато деталей про кожну можливість, діаграма використання може допомогти забезпечити більш високий рівень системи. Раніше було сказано, що "діаграми випадків використання є кресленнями для вашої системи". Вони забезпечують спрощене та графічне представлення того, що система дійсно повинна робити.

Завдяки своїй спрощеній природі діаграми випадків використання можуть бути гарним засобом комунікації для зацікавлених осіб. Малюнки намагаються імітувати реальний світ і дають змогу зацікавленим особам зрозуміти, як буде розроблятися система. Сіау і Лі провели дослідження, щоб визначити, чи існувала дійсна ситуація для діаграм випадків використання або взагалі не потрібні. Було виявлено, що діаграми випадків використання передають цілі системи більш спрощеним способом для зацікавлених сторін і що вони "інтерпретуються більш повно, ніж діаграми класів".

Мета діаграм випадків використання полягає в тому, щоб просто забезпечити високий рівень системи і передати вимоги до умов мирян для зацікавлених сторін. Додаткові діаграми та документація можуть бути використані для забезпечення повного функціонального та технічного вигляду системи [22].

3.2 Розробка програмного забезпечення

Отже всі навички набуті для старта розробки програмного забезпечення перше нам потрібно реалізувати елементарні принципи роботи з даними. Перше у плані це додавання інформації (рис. 3.7).

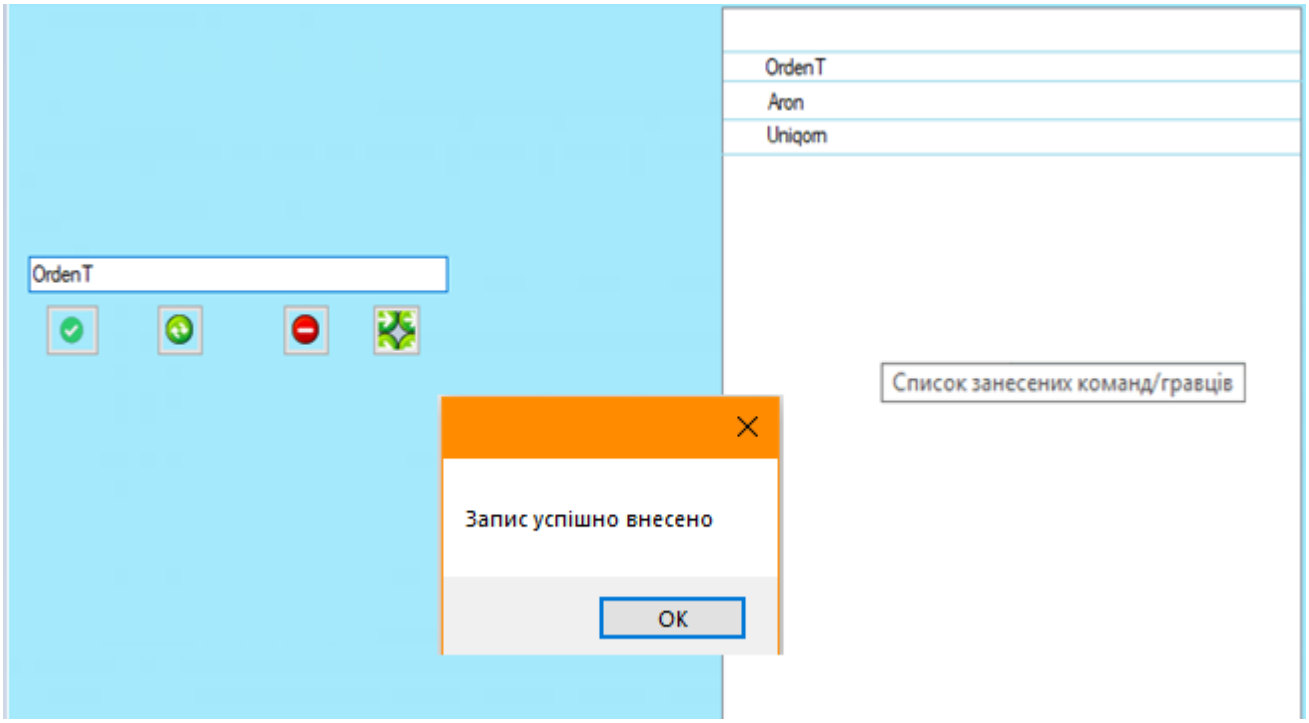

Рисунок 3.7 – Реалізація insert

```
private void button1_Click(object sender, EventArgs e)
            {
                if (textBox1.Text != "")
                {
                    cmd = new SqlCommand("insert into Teams(Name)", con);
                    con.Open();
                    cmd.Parameters.AddWithValue("@Name", textBox1.Text);
                    cmd.ExecuteNonQuery();
                    con.Close();
                    MessageBox.Show("Запис успішно внесено");
                    DisplayData();
                    ClearData();
                }
                else
                {
                    MessageBox.Show("Будь ласка, введіть детальніше
інформацію про команду/гравця!");
                }
            }
```

Insert - оператор мови SQL, який дозволяє додати рядки в таблицю, заповнюючи їх значеннями. Значення можна вставляти перерахуванням за допомогою слова values і перерахувавши їх в круглих дужках через кому або оператором select [23].

Під час виконання оператора можуть виникнути помилки:

1. Якщо при створенні таблиці для поля був вказаний параметр not null і не було визначено значення за замовчуванням (див. create), то при відсутності для нього вставляється значення виникне помилка. Рішення очевидно:

- або прибрати параметр not null;
- або вказати значення за замовчуванням;
- або вставити значення.

2. Якщо станеться спроба вставки в поле з типом identity (автоінкремент), то також відбудеться помилка. Вирішити проблему можна двома способами:

- не вставляти значення в це поле;
- вказати опцію identity_insert on після чого вставити унікальне значення для цього стовпця.

Наступним етапом йде добавляння функції оновлення (рис. 3.8) даних яке є дуже важлив, щоб уникнути казусів з неправильною назвою команди чи імені, нікнейма гравця.

```
private void button2_Click(object sender, EventArgs e)
            {
                if (textBox1.Text != "")
                {
                    cmd = new SqlCommand("update Teams from Name where
ID=@id", con);
                    con.Open();
                    cmd.Parameters.AddWithValue("@id", ID);
                    cmd.Parameters.AddWithValue("@Name", textBox1.Text);
                    cmd.ExecuteNonQuery();
                    MessageBox.Show("Запис успішно оновлено");
                    con.Close();
                    DisplayData();
                    ClearData();
                }
                else
                {
```

```
                MessageBox.Show("Будь-ласка, оберіть записати для
оновлення");
            }
        }
```

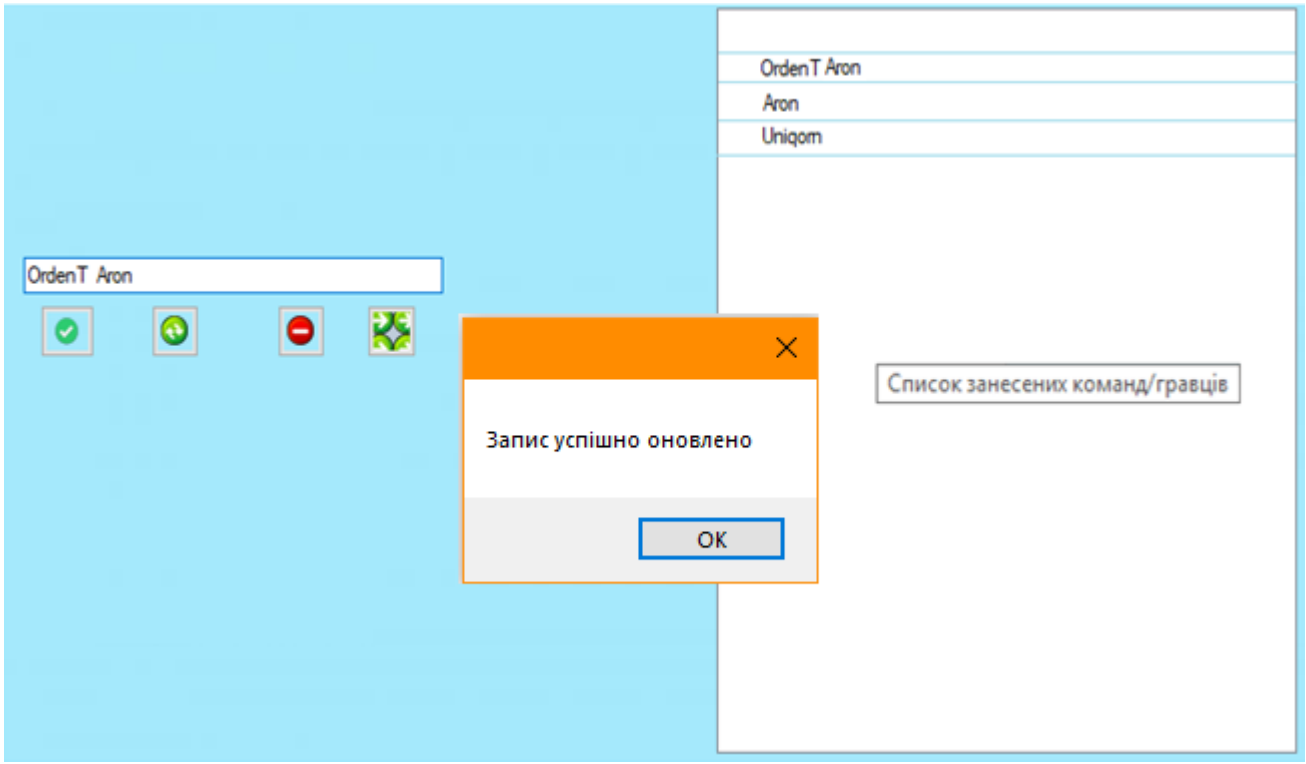

Рисунок 3.8 – Реалзіація update

Update - оператор мови SQL, що дозволяє оновити значення в заданих стовпцях таблиці. У деяких базах даних, таких як PostgreSQL, коли існує пропозиція FROM, по суті відбувається те, що цільова таблиця приєднана до таблиць, згаданих у списку, і кожна вихідна рядок об'єднання представляє операцію оновлення для цільової таблиці. При використанні FROM слід переконатися, що об'єднання створює не більше одного рядка виводу для

кожного рядка, який потрібно змінити. Іншими словами, цільова рядок не повинна приєднуватися до декількох рядків з іншої таблиці. Якщо це відбудеться, то для оновлення цільового рядка буде використано лише один з рядків приєднання, але який буде використовуватися, не можна легко передбачити.

Через цю невизначеність посилання на інші таблиці тільки в межах суб-селекцій є більш безпечним, хоча часто важче читати і повільніше, ніж при використанні об'єднання [24].

Наступним кроком буде розробка видалення даних (рис. 3.9) для корегування різних випадкових або навмисно не вдалих занесених даних які повністю не відповідають або не потрібні для бази адміністратора.

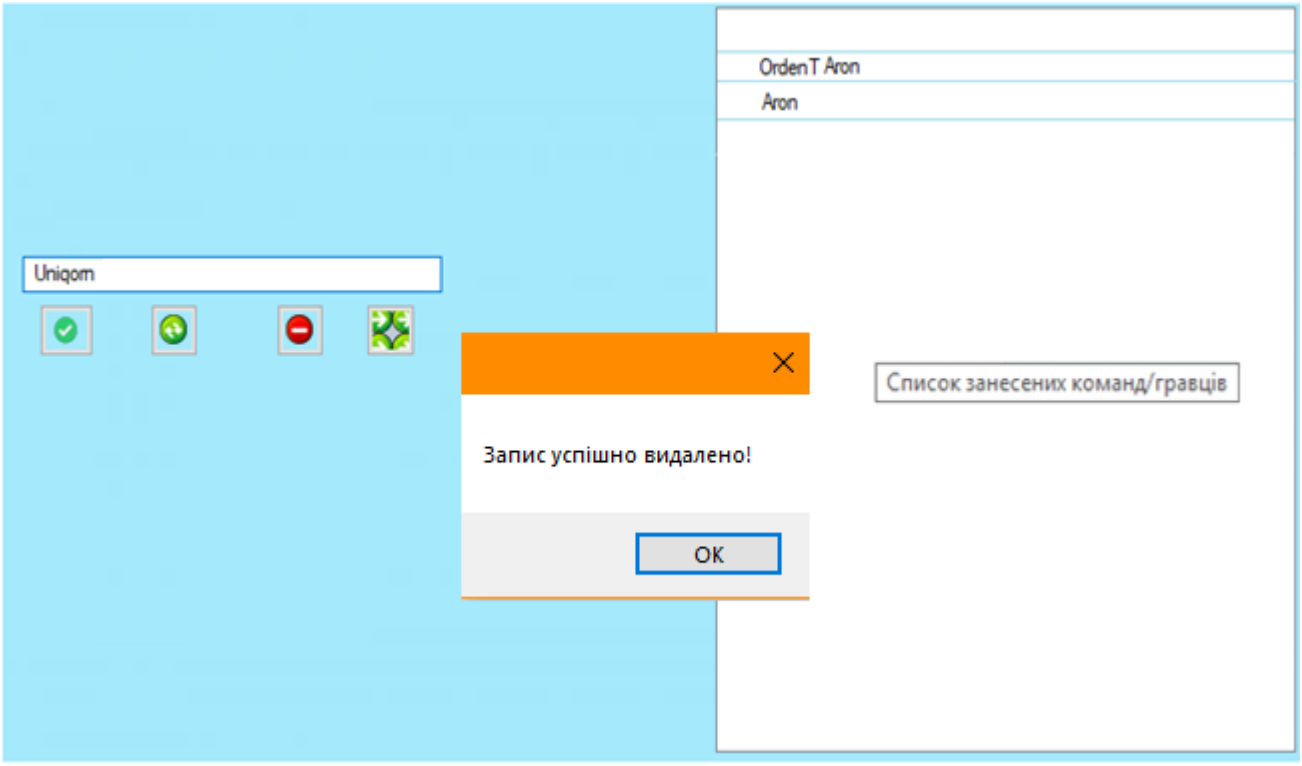

Рис 3.9 – Реалізація delete

```
private void button3_Click(object sender, EventArgs e)
            {

                if (ID != 0)
                {
                    cmd = new SqlCommand("delete Teams where ID=@id",
con);
                    con.Open();
                    cmd.Parameters.AddWithValue("@id", ID);
                    cmd.ExecuteNonQuery();
                    con.Close();
                    MessageBox.Show("Запис успішно видалено!");
                    DisplayData();
                    ClearData();
                }
                else
                {
                    MessageBox.Show("Будь-ласка, оберіть Записати для
видаленя");
                }
                  }
```

Delete – у мовах, подібних SQL, DML-операція видалення записів з таблиці. Критерій відбору записів для видалення визначається виразом Where. У разі, якщо критерій відбору не визначений, виконується видалення всіх записів [25].

Етап підключення головних функцій бази даних, а саме – insert, update, delete відбувся успішно. Тепер можна спокійно заповнювати базу командами або гравцями (рис. 3.10) які будуть приймати учать в турнірі. Всі належні функції для цього присутні вони забезпечують економію часу та оптимізацію турніру адже записувати все в ручну було б довго і допуститися помилки набагато легше і виправляти її було б набагато важче.

На ступним кроком для розробки програмного забезпечення буде підключення наступної сторінки інтерфейсу, в якій буде виконуватися жеребкування та самий груповий етап. Це самий головний момент в програмі груповий етап буде відбуватися через формат Швейцарська система про який уже було сказано достатньо. В першу чергу треба створити групу команд які будуть приймати участь. Перш ніж почати жеребкування заповнимо нашу таблицю учасниками. Заповнення буде відбуватися в ручну через інтерфейс програми буде задіювати команду insert якщо буде допущена помилка тоді апдейт, а може й делейт.

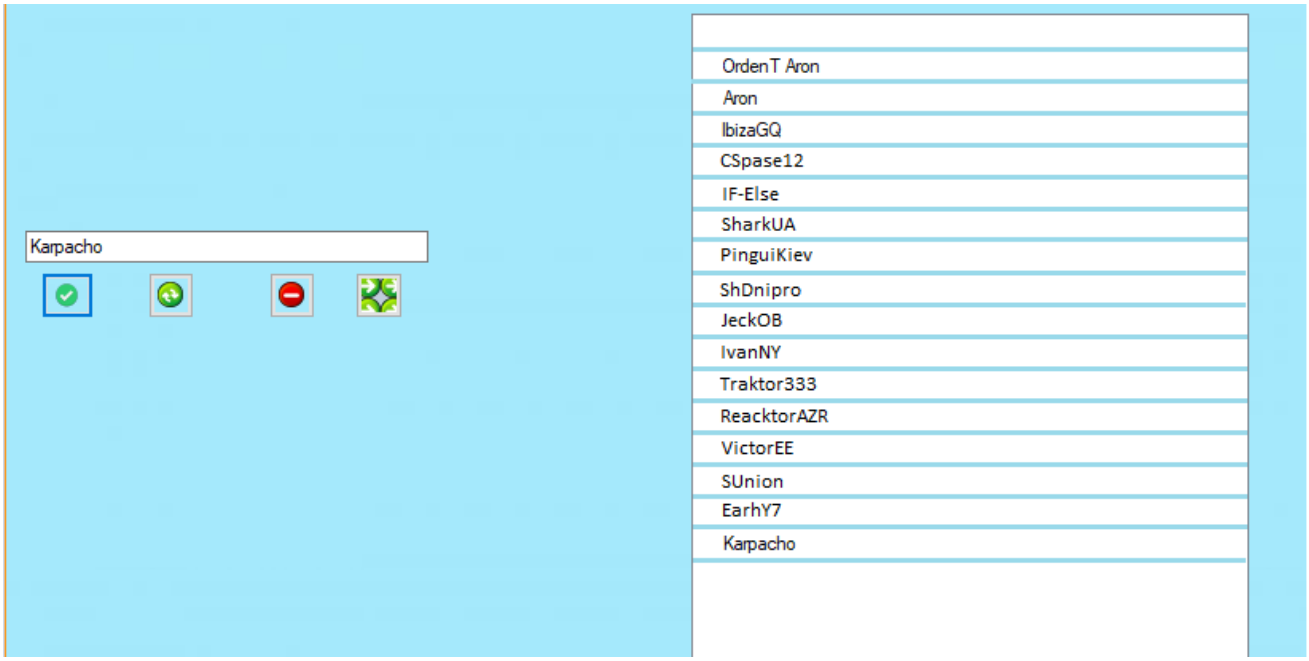

Рисунок 3.10 – Заповнена таблиця

Жеребкування перших пар відбувається навмання що б зробити інтригу для турніру і глядачів. Раніше потрібно було засипати в мішочок листочки з назвами команд, а тепер просто потрібно нажати одну кнопку (рис. 3.11).

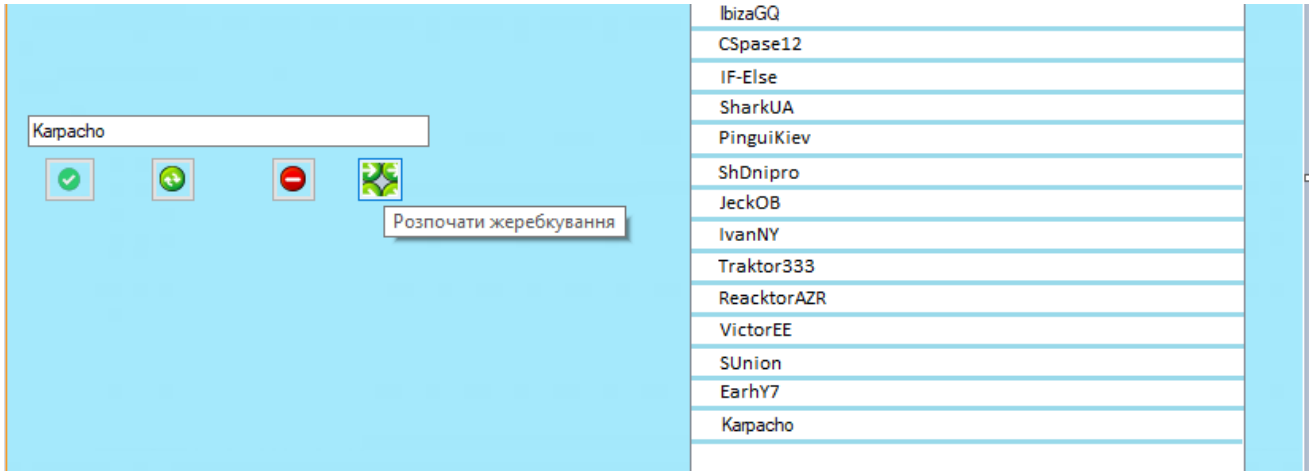

Рисунок 3.11 – Кнопка для жереба

```
{
       static Random _random = new Random();

         public static string[] RandomizeStrings(string[] arr)
         {
             List<KeyValuePair<int, string>> list = new
List<KeyValuePair<int, string>>();
             foreach (string s in arr)
             {
                 list.Add(new KeyValuePair<int, string>(_random.Next(),
s));
             }

             var sorted = from item in list
                          orderby item.Key
                          select item;
             string[] result = new string[arr.Length];
                        int index = 0;
             foreach (KeyValuePair<int, string> pair in sorted)
             {
                 result[index] = pair.Value;
                 index++;
             }
             return result;
         }
          }
```

Псевдовипадкові числа вибираються з однаковою ймовірністю з кінцевого набору чисел. Вибрані числа не є абсолютно випадковими, оскільки для їх вибору

використовується математичний алгоритм, але вони є досить випадковими для практичних цілей. Поточна реалізація класу Random ґрунтується на модифікованій версії алгоритму генератора випадкових чисел Дональда Е. Кнута [26].

Роль випадкавого жеребу дуже велика адже є команди або гравці які знають один одного дуже добре і якщо їх зведе жереб то глядачі подачать шалений матч. Або навпаки команда одна сильніша на рівень за іншу і буде великий погром але можей й статися сенсація. Головний плюс сліпого жеребу це те що команди отримують суперника без всякої задньої думки.

Жеребкування допомагає командам дізнатися свого суперника за допомогою одного натиснення кнопки що зекономить велику кількість часу учасників.

Після жеребкування починається перший рануд (рис. 3.12) де команди іграють матчі між собою і в цьому етапі переможці будуть набирати статистику перемог для проходу в плей-офф з 16 командами це займе п’ять раундів. Команди розпределені між собою. І наступна дія це старт нового раунду всі результати матчі потрібно вносити в ручну, таблиця генерується автоматично.

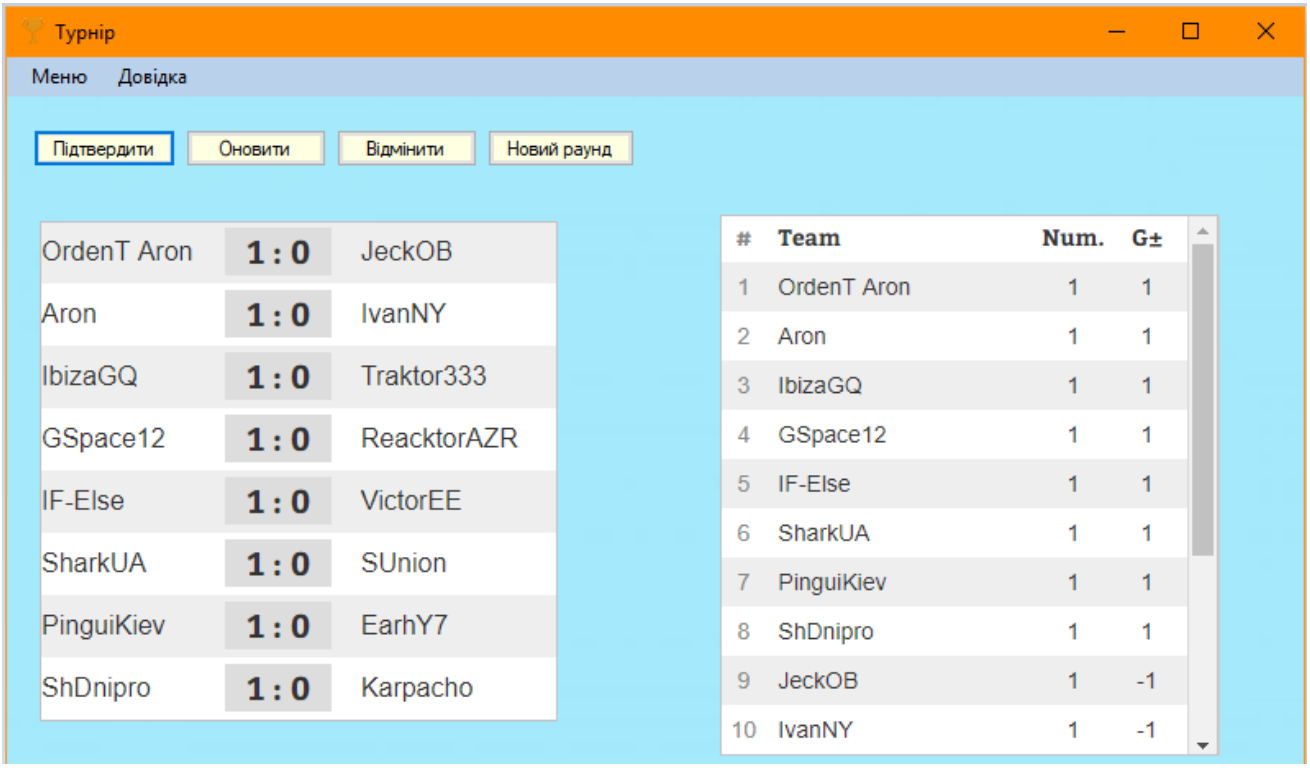

Рисунок 3.12 – Перший раунд

Можна розпочинати другий раунд (рис 3.13). Цей раунд у якому уже визначаються певні фаворити та невдахи. Команди які маю статистику (2-0) уже за крок від виходу до плей-офф, але насправді це самий складний крок. Через те що вони будуть іграти проти такоїж команди у якої статистика (2-0). Глядачі побачать чудовий матч рівних опонентів. Аутсайдери отримують шанс на поверненя в турнір. Через те, що вони також іграють протів таких аутсейдерів з рахунком (0-2) раунд під номер три буде дуже не передбачуваним та інтресним.

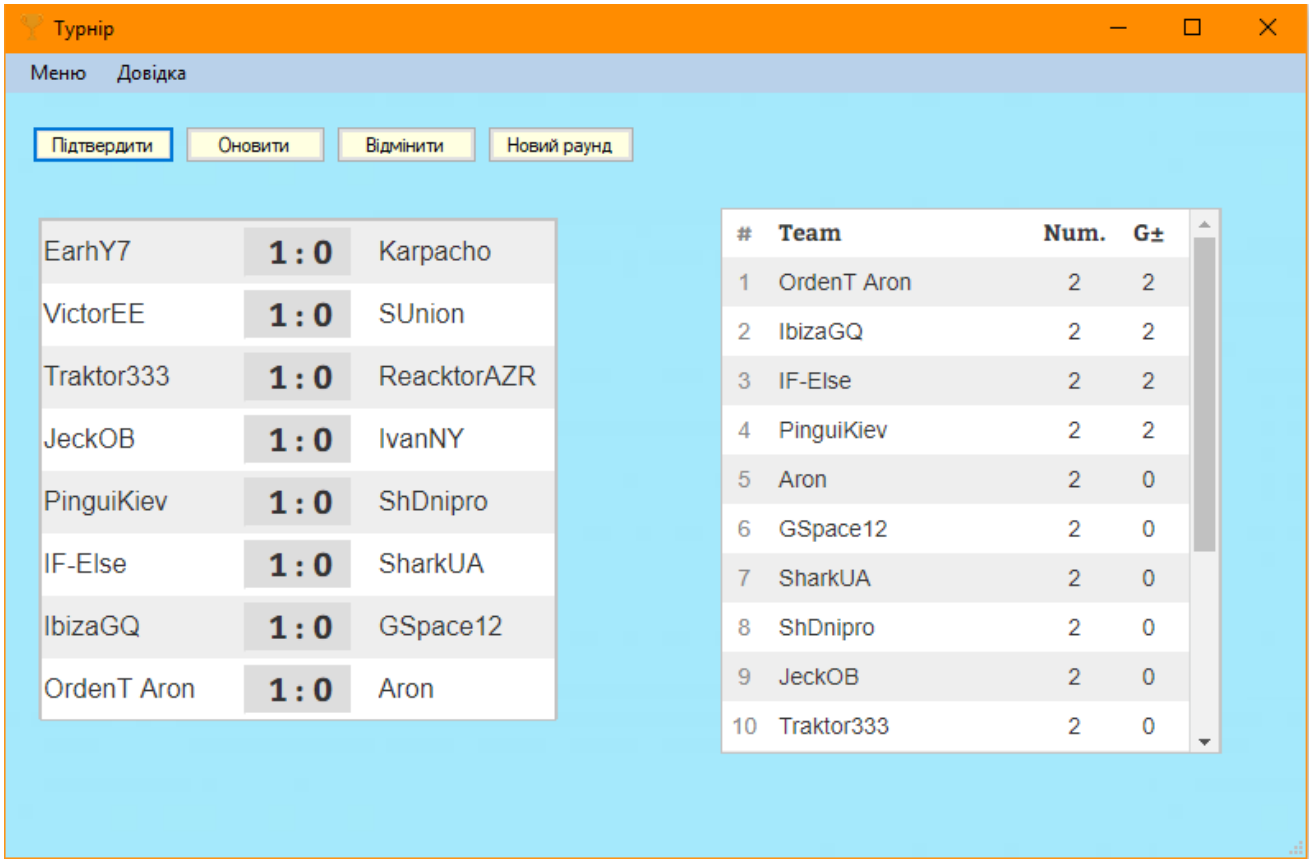

Рисунок 3.13 – Другий раунд

Третій раунд показує хто вибув з турніру хто продовжує шлях, а хто уже готується до плей-офф (рис. 3.14). Головний шарм Швейцарської системи це те, що починаючи з друго раунду команди іграють протів команд рівних собі по силі. Звісно цей формат більше підходить для турінірів з не високую фізичною нагрузкою. Якщо застосувати цей формат для футболу то прийдеться витрачати більше часу на груповий етап і команди які вийдуть з статистикою (3-0) вони будуть мати більшу перевагу в плані фізики на наступні поєдинки плей-офф.

Також не меньш важливим фактором для такого виду спорту як футбол це є складова грошей. Нажаль Швейцарський формат чуть затягнутий і це не дає футбольним асоціаціям можливість його використовувати.

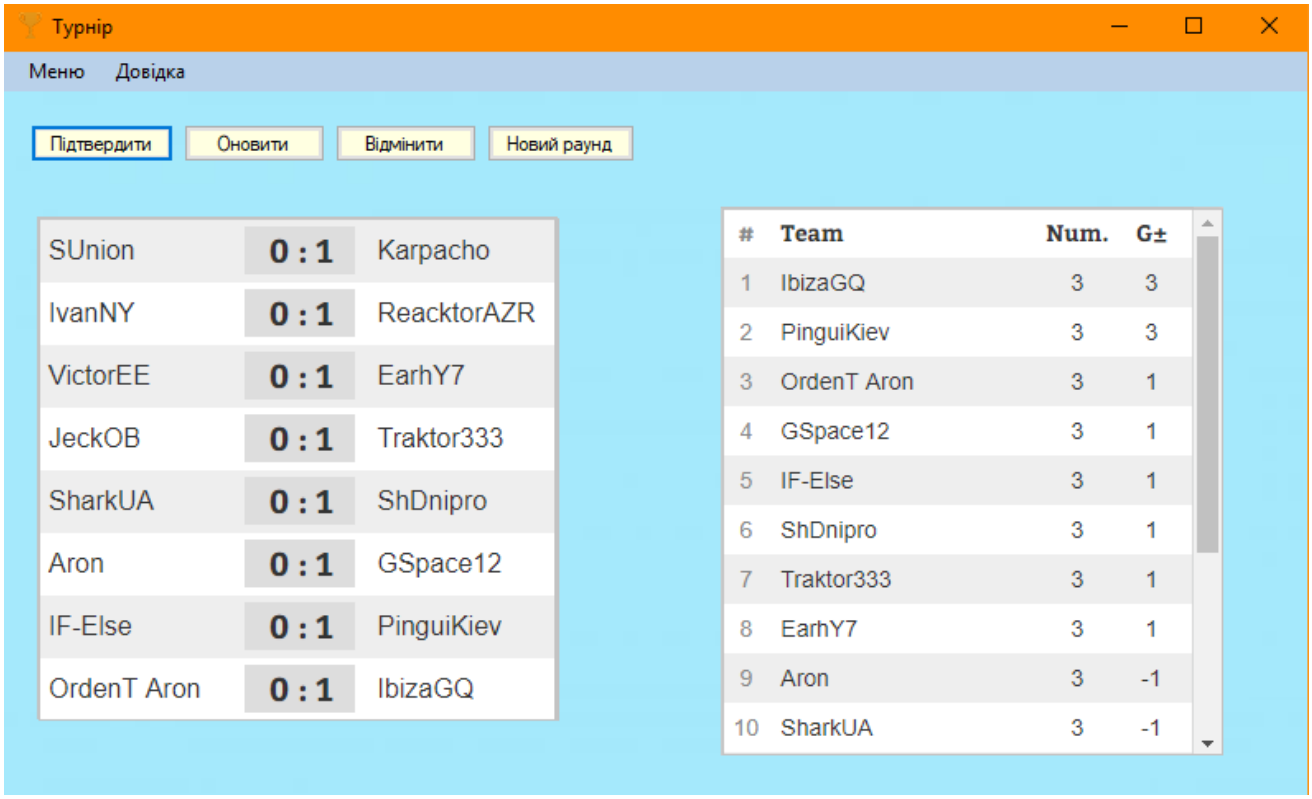

Рисунок 3.14 – Третій раунд

Команда IbizaGQ та PinguinKiev закінчують груповий етап з статистикою (3-0) і прямують прямо в плей-офф. Також залишають турнір команди SUnion та IvanNY статистика (0-3) (рис. 3.15). Після того як команда або команди покідають турнір організатор найчастіше використовують практику їхньої заміни на наступний сезон або турнір. Це назвивають пониження в класі застосовують цю методику для росту інших колетктивів та розвитку іміджу своєї організації турінру. Є практика огранізаторів спортивних подій приглашати країну господаря прямо на турнір, це робиться для популізації виду спорту в країнах де він мало розвинутий. Щоб люди які проживають в тій країні могли підтримати свою збірну.

Також під час самого матчу не меньш важлива є підтримка вболівальників які підтримують свою команду. Це надає велику моральньну підтримуку для спортсменів і вони демонстують більшу самовіддачу яка впливає на результат.

#	Team	Num.	G±
7	Traktor333	3	1
8	EarhY7	3	1
9	Aron	3	-1
10	SharkUA	3	-1
11	JeckOB	3	-1
12	ReacktorAZR	3	-1
13	VictorEE	3	-1
14	Karpacho	3	-1
15	IvanNY	3	-3
16	SUnion	3	-3

Рисунок 3.15 – Команди які вибули

Передостанній раунд (рис. 3.16) також покаже які команди проходять дальше в плей-офф, а які вибувають.

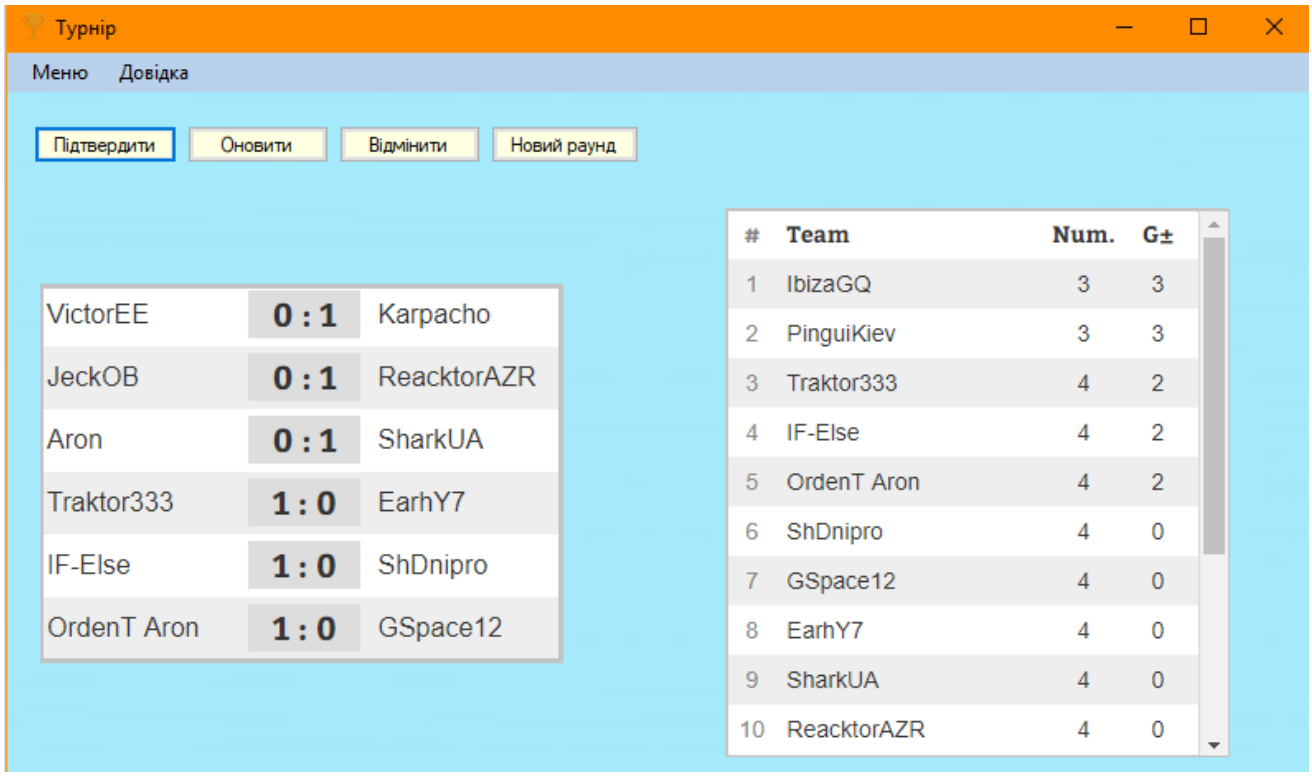

Рисунок 3.16 – Четвертий раунд

Зазвичай такі матчі показують хто уже виправив свої помилки, ахто почав

допускати їх ще більше. Адже статистика (2-0) навіть не гарантує команді учать в плей-офф. Команди Traktor333, IF-Else, OrdenT Aron закінчують груповий етап зі статикою (3-1), та приєднються до команд які уже закінчили груповий етап і продовжать свій шлях в турнірі уже в плей-офф. Також турнір покидають три команди Aron, VictorEE, JeckOB (рис. 3.17), зі статискиою (1-3). Зазвичай в різних спортивних лігах команди які вибувають другими, вони не зразу понижаються в класі або взагалі не понижаються. Для них часто використовують матчі на виліт з претендентами на один клас нижчної ліги, які заняли місця за переможцями але попали в верхню частину.

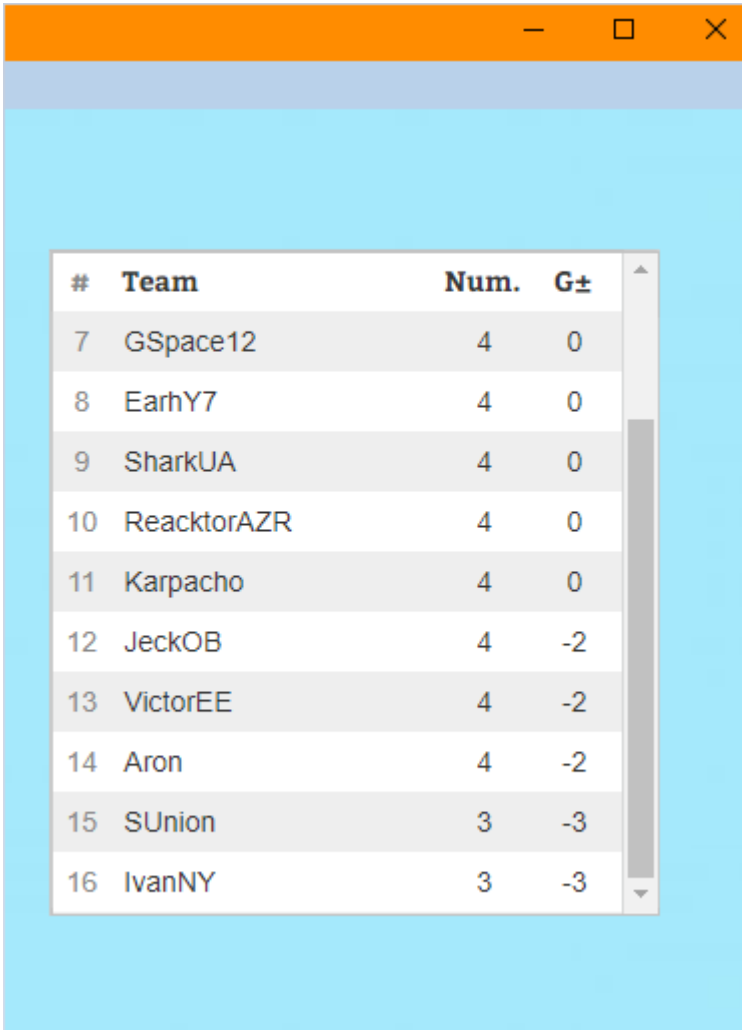

Рисунок 3.17 – Команди які вибули

Послідній раунд який визначить яка команда залишається в боротьбі за чемпіонство і потрапляє в наступний етап змагань, а яка команда покине турнір за крок від плей-офф. Зазвичай такі матчі збирають найбільний ажіотаж в груповоме

етапі. Команди мають статистику (2-2) і ніодна не хоче покидати турнір матч буде проходити під великою напругою команди або гравці будуть боротися за кожний вдалий момент (рис. 3.18). Що б перемогти і задовільнити свої амбіції і порадувати вболівальників які їх підтримують.

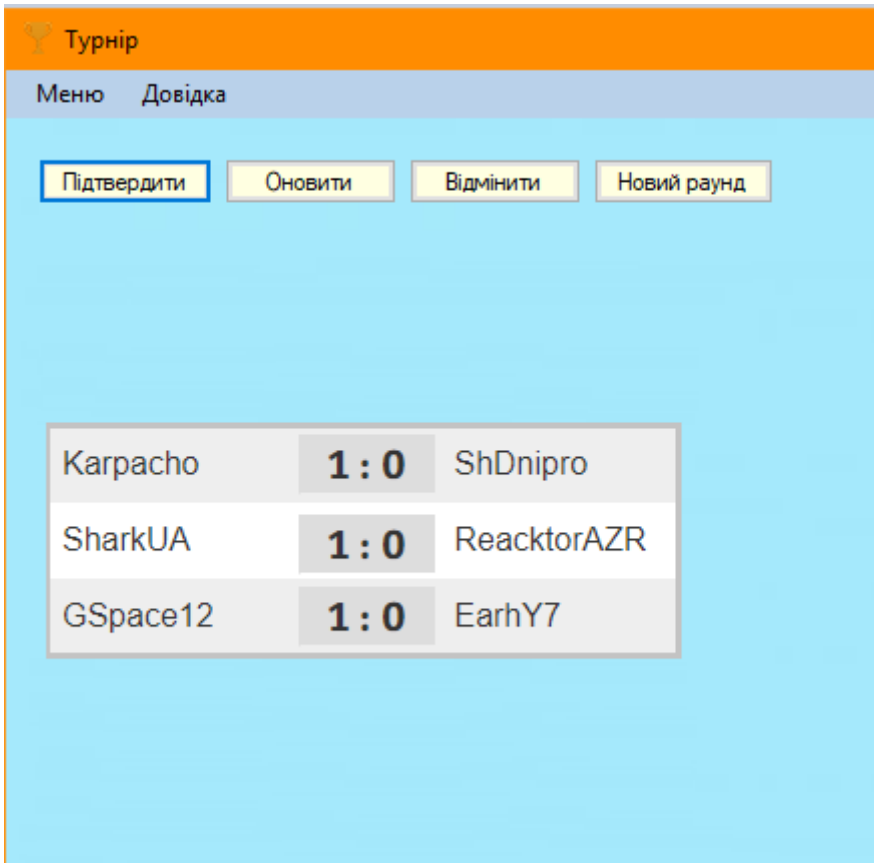

Рисунок 3.18 – Послідній раунд

Команда Karpacho, SharkUA, GSpace12 проходять в плей-офф зі статистикою (3-2), а їні опоменти заилшають турнір. Команди ShDnipro. ReacktorAZR, EarhY7 покинули турнір з статистикою (2-3) (рис 3.19).

6	GSpace12	5	1
7	SharkUA	5	1
8	Karpacho	5	1
9	ShDnipro	5	-1
10	EarhY7	5	-1
11	ReacktorAZR	5	-1

Рисунок 3.19 – Фінальна таблиця

```
private static List<Team> CreateTeams(int totalTeams)
    {
        return Enumerable.Range(1, totalTeams).Select(x => new Team("Team
" + x, x)).ToList();
    }

    private static List<Matchup> CreateMatchups(List<Team> teams)
    {
        var matchups = new List<Matchup>();

        for (int i = 0; i < teams.Count / 5; i++)
        {
            matchups.Add(new Matchup(teams[i], teams[teams.Count - (i +
1)]));
        }

        return matchups;
    }

    private static void PrintBracket(List<Matchup> Matchups, int round)
    {


        Console.WriteLine("Round " + round + ":\n");
        foreach (var matchup in Matchups.OrderBy(x =>
x.GetFavored().Seed))
        {
            Console.WriteLine(matchup);
        }
    }

    private static List<Team> PlayRound(List<Matchup> matchups)
    {
        return matchups.Select(x => x.GetFavored()).ToList();
    }
}

public class Team
{
    public Team(string name, int seed)
    {
        Name = name;
        Seed = seed;
    }

    public string Name { get; set; }

    public int Seed { get; set; }

    public static Team GetBetterSeed(Team t1, Team t2)
    {
```

```
            return t1.Seed < t2.Seed ? t1 : t2;
        }
    }

    public class Matchup
    {
        public Matchup(Team t1, Team t2)
        {
            Team1 = t1;
            Team2 = t2;
        }

        public Team Team1 { get; set; }
        public Team Team2 { get; set; }

        public override string ToString()
        {
            return Team1.Name + " vs. " + Team2.Name;
        }

        public Team GetFavored()
        {
            return Team.GetBetterSeed(Team1, Team2);
        }
    }
```

Наступним етапом будуть ігри плей-офф. Вісім команд будуть боротися за чемпіонство. Плей-офф реалізується типовую системою програв-вибув. Цю систему використовують на олімпійських іграх та інших широко маштабних турнірах. Звісно організатори можуть добавляти різні схеми для ігор на виліт, але найчастіше використовують просту одинарну сітку. Команди жеребуються між собою на розсуд орагіназторів. Це може бути як і сильніший протів слабшого (3-0) протів (3-2) і т.д. або випадково. Два варіанта є цілком вірними, оскільки перший він є цілком логічним для команд які витратили більше сил або поставили завдання виграти групувий етап (3-0) і вони розраховують на те, що в іграх на вибування їм це допоможе зіграти з слабшим суперником. Другий варіант він більш азартний і добавляє бльшої інтриги і напргуи для глядачів і фанів. Єдений мінус що в другому варіанті команди з статистикою (3-0) можуть зустрітися між собою, і коли один фаворит вибває другого на першому етапі плей-офф це не додає іміджу організаторам. Тому найкращий варінт розставити команди з (3-0) по різним куткам сітки і дальше провести випадкове жеребкування.

Отже вісім команд утворюють ¼ фіналу вже з зрозумілим жеребкування (рис. 3.20). ¼ фіналу команди за 3 кроки від фіналу. Нерв таких ігор просто зашкалює глядачів та вболівальників стає ще більше.

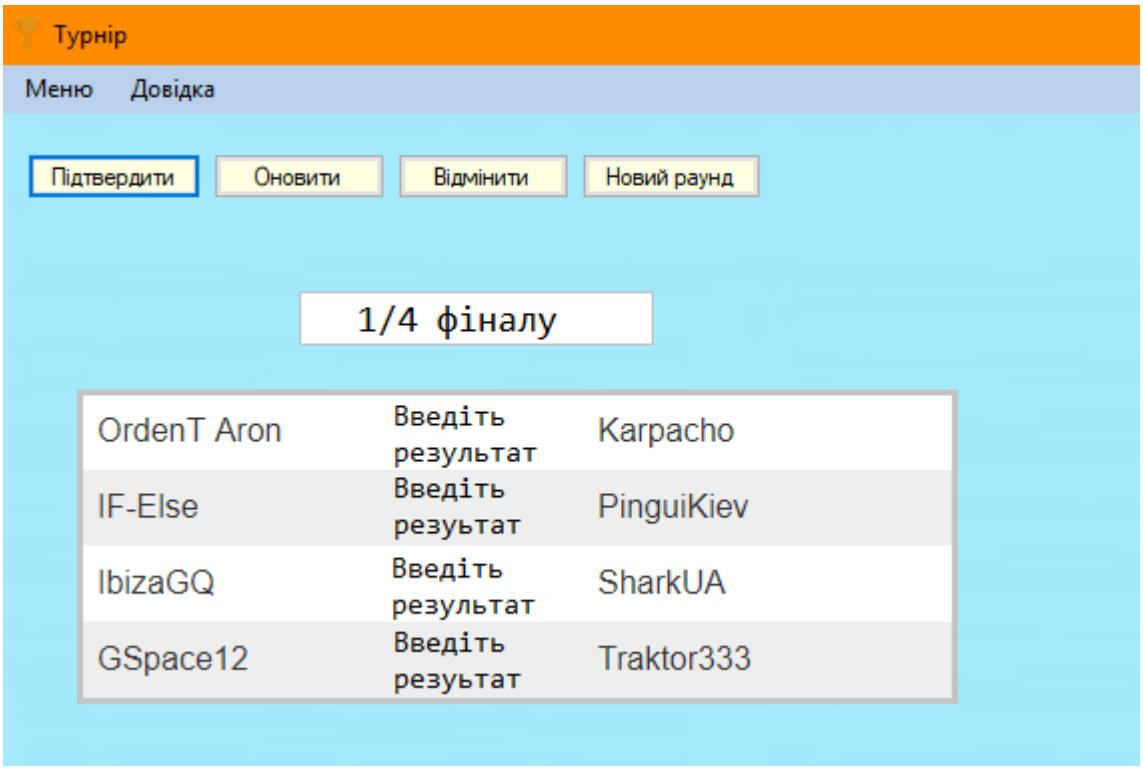

Рисунок 3.20 – ¼ фіналу

Матчі проведені, внесений результат (рис. 3.21) команди починають готуватися до нового раунду плей-офф.

Рисунок 3.21 – Результати ¼

Наступний раунд плей-офф (рис. 3.22) це півфінал де залишилося чотирикоманди. Karpacho які здоали OrdenT Aron, IF-Else і IbizaGQ відповідно PinguiKiev, SharkUA та GSpace12 які перемогли Traktor333. Півфінал великої спортивної події викликає велику хвилю ажіотажу навіть у тих людей які рідко дивлються спорт. Наступний раунд плей-офф (рис 3.20) це півфінал де залишилося чотирикоманди. Karpacho які здоали OrdenT Aron, IF-Else і IbizaGQ відповідно PinguiKiev, SharkUA та GSpace12 які перемогли Traktor333. Півфінал великої спортивної події викликає велику хвилю ажіотажу навіть у тих людей які рідко дивлються спорт. Команди жеребкуються уже або випадково або по заданій сітці все вирішують організатори.

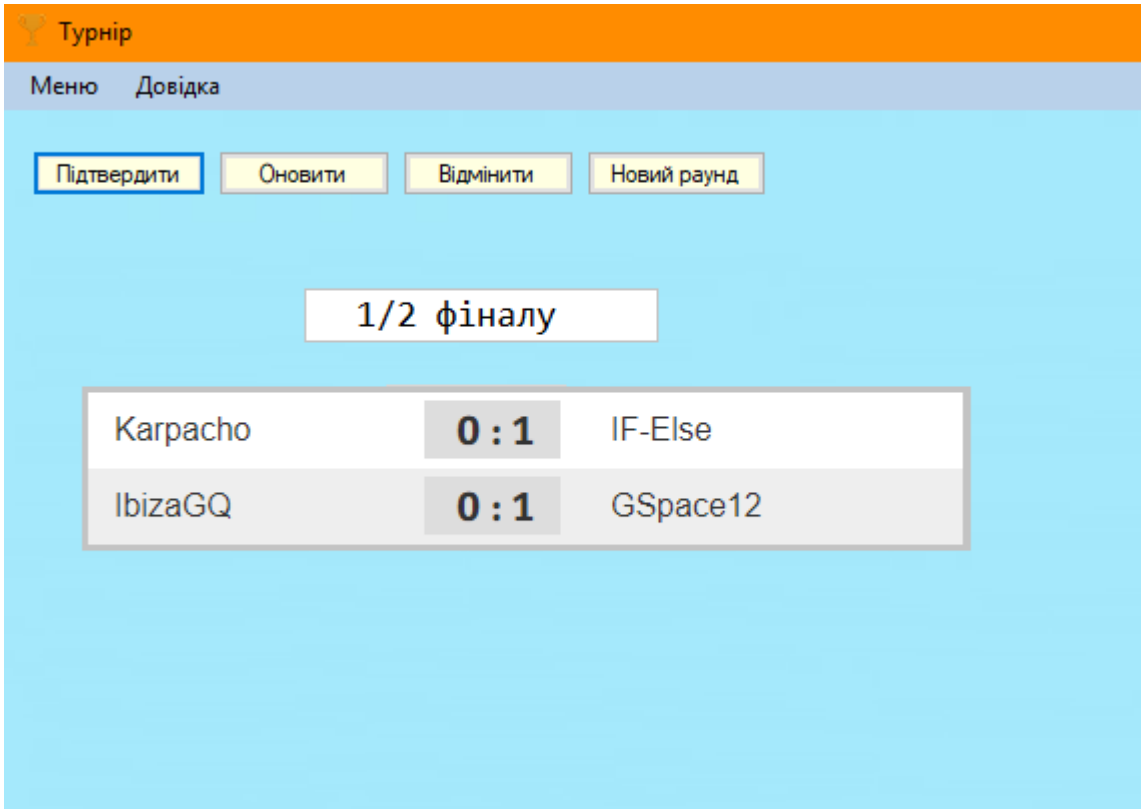

Рисунок 3.22 – Результати ½

Отже фіналісти відомі це команди IF-Else та GSpace12 які переграли своїх опонентів Karpacho та IbizaGQ відповідно. Фінал любої спортивної події викликає небувалий інтерес в спільноти. Деякі засоби масової інформації давали статистику що фінал чемпіоната світу по футболу 2018 року по девилося рекордна кількість осіб приблизно кожна 6 людина яка живе на планеті. У фіналі зійдуться команди

IF-Else та GSpace12 та виявлять хто сильніший (рис. 3.23). У справжньому спорті амбіції гравців часто зашкалюють і вони готові робити різні речі часто не коректні або взагалі не доречні і не красиві щоб добитися успіху. Але щоб стати чемпіоном не можна просто надіятися на свою силу потрібна удача і всі методи які зможуть наблизити до перемоги є дієві.

Рисунок 3.23 – Результат фіналу.

Переможцем стає команда IF-Else яка закінчила груповий етап з рахунком (3-1).

```
public void ListMatches(List<string> ListTeam)
            {
                if (ListTeam.Count % 2 != 0)
                {
                    ListTeam.Add("Names");
                }

                int numNames = (numTeams - 1);
                int halfSize = numTeams / 2;

                List<string> teams = new List<string>();

                teams.AddRange(ListTeam.Skip(halfSize).Take(halfSize));
                teams.AddRange(ListTeam.Skip(1).Take(halfSize -
1).ToArray().Reverse());

                int teamsSize = teams.Count;

                for (int day = 0; day < numNames; day++)
```

```
                {
                    Console.WriteLine("Names {0}", (day + 1));

                    int teamIdx = day % teamsSize;

                    Console.WriteLine("{0} vs {1}", teams[teamIdx],
ListTeam[0]);

                    for (int idx = 1; idx < halfSize; idx++)
                    {
                        int firstTeam = (day + idx) % teamsSize;
                        int secondTeam = (day + teamsSize - idx) %
teamsSize;
                        Console.WriteLine("{0} vs {1}", teams[firstTeam],
teams[secondTeam]);
                    }
                     }
```

Турнір завершився. Організатори роздають індивідуальні нагороди учасникам турніра за різні досягнення. Турнір зберігається в текстовому форматі для створення нового турінру потрібно в меню нажати новий турнір, або завершити старий і автоматично данні очистяться.

ВИСНОВКИ

Під час виконання даного дипломного завдання було розробити програмне забезпечення для проведення турнірів за Швейцарською системою.

Програма розроблена засобами C# в конструкторі Visual Studio. Проект розроблений з розрахунком на подальше вдосконалення і додавання нових функціональних можливостей. Тому інтеграція нових функцій займатиме мінімум часу і зусиль.

Результатом роботи є програма яка генерує Швейцарську систему проведення турнірів для різних видів спорту.

Програма була протестована на кількість місць, не вірне жеребкування та інші можливі баги. Програма справилися зі всіма поставленими на неї задачами також показала свою швидкість та оптимізованість. Робота тестування була завершена з позитивними характеристиками.

Дипломна робота включає в себе вичерпний теоретичний і практичний опис проекту та кожної його частини.

Дипломну роботу виконано у повній відповідності до завдання і всіма нормативними вимогами. Додаток дозволяє адміністратору керувати (додавати, редагувати, видаляти) всіма даними які є доступні в програмі.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Швейцарська система турнірів. Матеріал з cbgames. URL: http://www.cbgames.kiev.ua/stati/12147.html (дата звернення: 15.12.2018).
2. Visual Studio. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio (дата звернення: 15.12.2018).
3. Windows Froms. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Windows_Forms (дата звернення: 15.12.2018).
4. .NET Framework. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/.NET_Framework (дата звернення: 25.12.2018).
5. .NET Core. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://en.wikipedia.org/wiki/.NET_Core (дата звернення: 25.12.2018).
6. MSDN Library. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://en.wikipedia.org/wiki/Microsoft_Developer_Network#Library (дата звернення: 25.12.2018).
7. Платформа .NET Framework і мова програмування C#. Матеріал з shwanoff. URL: https://shwanoff.ru/dotnet-framework (дата звернення: 25.12.2018).
8. Class. Матеріал з programer.in.ua. URL: http://programer.in.ua (дата звернення: 15.01.2019).
9. Class. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/class (дата звернення: 15.01.2019).
10. База даних. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/ru-ru/visualstudio/data-tools/installing-database-systems-tools-and-samples (дата звернення: 15.01.2019).
11. База даних. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/ru-ru/visualstudio/data-tools/dataset-tools-in-visual-studio (дата звернення: 15.01.2019).

12. Windows Forms. Матеріал з mentanit. URL: https://metanit.com/sharp/windowsforms (дата звернення: 15.02.2019).

13. Windows Forms. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/windows-forms-overwiew (дата звернення: 15.02.2019).

14. Розробка програмного забезпечення. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Розробка_програмного_забезпечення (дата звернення: 15.02.2019).

15. Віджет. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Віджет (дата звернення: 15.02.2019).

16. Звязки бази даних. Матеріал з support.office.com. URL: https://support.office.com/uk-ua/article/Посібник-зі-зв-язків-між-таблицями (дата звернення: 15.02.2019).

17. Інтерфейс користувача. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Інтерфейс_користувача (дата звернення: 15.02.2019).

18. Меню. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Меню_(інтерфейс) (дата звернення: 15.02.2019).

19. Text box. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.textbox (дата звернення: 25.03.2019).

20. List box. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.listbox (дата звернення: 25.03.2019).

21. Table layout panel. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.tablelayoutpanel (дата звернення: 15.11.2019).

22. Use case diagram. Матеріал з Вікіпедії – вільної енциклопедії. . URL: https://en.wikipedia.org/wiki/Use_case_diagram (дата звернення: 25.03.2019).

23. Insert. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Insert_(SQL) (дата звернення: 25.03.2019).

24. Update. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Update_(SQL). (дата звернення: 25.03.2019).

25. Delete. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Delete_(SQL) (дата звернення: 05.04.2019).

26. Random. Матеріал з docs.microsoft – сховище документації Майкрософт. URL: https://docs.microsoft.com/en-us/dotnet/api/system.random (дата звернення: 05.04.2019).

27. Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#: ИД Питер, 2011. 784 с.

28. Ликнесс Д. Приложения для Windows 8 на C# и XAML Петербург: ИД Питер, 2013. 368 с.

29. Петцольд Ч. Программирование для Microsoft Windows 8: ИД Питер, 2014. 1008 с.

30. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#: ИД Питер, 2018. 896 с.

31. Кляйн К. Е. SQL. Справочник. Москва: Символ-Плюс, 2010. 310 с.

Додаток А

Підключення бд.

```
public partial class Form1 : Form
    {
        SqlConnection con = new SqlConnection(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\123\superdo\SuperDO\
SuperDO\Database1.mdf;Integrated Security=True;");
        SqlCommand cmd;
        SqlDataAdapter adapt;
        ToolTip toolTip;

        int ID = 0;

        public Form1()
        {
            InitializeComponent();
            DisplayData();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
ToolTip t = new ToolTip();
            t.SetToolTip(button1, "Додати команду/гравця");
            ToolTip d = new ToolTip();
            d.SetToolTip(button2, "Видалити команду/гравця");
            ToolTip c = new ToolTip();
            c.SetToolTip(button3, "Оновити команду/гравця");
            ToolTip j = new ToolTip();
            j.SetToolTip(button4, "Розпочати жеребкування");
            ToolTip w = new ToolTip();
            w.SetToolTip(textBox1, "Введіть команду/гравця");
            ToolTip h = new ToolTip();
            h.SetToolTip(listBox1, "Список занесених команд/гравців");
            ToolTip q = new ToolTip();
            q.SetToolTip(listBox2, "Груповий етап");
            ToolTip r = new ToolTip();
            r.SetToolTip(listBox3, "Таблиця");
            ToolTip k = new ToolTip();
            k.SetToolTip(listBox2, "Плей-офф");

        }

        private void button1_Click(object sender, EventArgs e)
            {
                if (textBox1.Text != "")
                {
```

```
                    cmd = new SqlCommand("insert into Teams(Name)",
con);
                    con.Open();
                    cmd.Parameters.AddWithValue("@Name",
textBox1.Text);
                    cmd.ExecuteNonQuery();
                    con.Close();
                    MessageBox.Show("Запис успішно внесено");
                    DisplayData();
                    ClearData();
                }
                else
                {
                    MessageBox.Show("Будь ласка, введіть детальніше
інформацію про команду/гравця!");
                }
        }

        private void button2_Click(object sender, EventArgs e)
            {
                if (textBox1.Text != "")
                {
                    cmd = new SqlCommand("update Teams from Name where
ID=@id", con);
                    con.Open();
                    cmd.Parameters.AddWithValue("@id", ID);
                    cmd.Parameters.AddWithValue("@Name",
textBox1.Text);
                    cmd.ExecuteNonQuery();
                    MessageBox.Show("Запис успішно оновлено");
                    con.Close();
                    DisplayData();
                    ClearData();
                }
                else
                {
                    MessageBox.Show("Будь-ласка, оберіть записати для
оновлення");
            }
     }
        }

private void button3_Click(object sender, EventArgs e)
            {

                if (ID != 0)
                {
                    cmd = new SqlCommand("delete Teams where ID=@id",
con);
                    con.Open();
                    cmd.Parameters.AddWithValue("@id", ID);
```

```
                    cmd.ExecuteNonQuery();
                    con.Close();
                    MessageBox.Show("Запис успішно видалено!");
                    DisplayData();
                    ClearData();
                }
                else
                {
                    MessageBox.Show("Будь-ласка, оберіть Записати для
видаленя");
                }
                 }

                private void DisplayData()
        {
            con.Open();
            DataTable dt = new DataTable();
            adapt = new SqlDataAdapter("select * from Teams", con);
            adapt.Fill(dt);
            dataGridView1.DataSource = dt;
            con.Close();
        }

        private void ClearData()
        {
            textBox1.Text = "";
            textBox2.Text = "";
            textBox3.Text = "";
            ID = 0;
        }


        private void dataGridView1_RowHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
        {

                ID =
Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString
());
                textBox1.Text =
dataGridView1.Rows[e.RowIndex].Cells[1].Value.ToString();
                textBox2.Text =
dataGridView1.Rows[e.RowIndex].Cells[2].Value.ToString();
                textBox3.Text =
dataGridView1.Rows[e.RowIndex].Cells[3].Value.ToString();

        }
    }
}
```

Груповий етап.

```
public static void Main()
{
    var teamCount = 16;
    var round = 5;

    var teams = CreateTeams(teamCount);

    while (teams.Count > 1)
    {
        var matchups = CreateMatchups(teams);
        PrintBracket(matchups, round++);
        teams = PlayRound(matchups);
    }

    Console.WriteLine("\n" + teams.Single().Name + " is the winner!");
}

//creates dummy teams: "Team1" Seed 1, "Team2" Seed 2, etc...
private static List<Team> CreateTeams(int totalTeams)
    {
        return Enumerable.Range(1, totalTeams).Select(x => new
Team("Team " + x, x)).ToList();
    }

    private static List<Matchup> CreateMatchups(List<Team> teams)
    {
        var matchups = new List<Matchup>();

        for (int i = 0; i < teams.Count / 5; i++)
        {
            matchups.Add(new Matchup(teams[i], teams[teams.Count - (i
+ 1)]));
        }

        return matchups;
    }

    private static void PrintBracket(List<Matchup> Matchups, int
round)
    {


        Console.WriteLine("Round " + round + ":\n");
        foreach (var matchup in Matchups.OrderBy(x =>
x.GetFavored().Seed))
        {
            Console.WriteLine(matchup);
        }
    }
```

```
    private static List<Team> PlayRound(List<Matchup> matchups)
    {
        return matchups.Select(x => x.GetFavored()).ToList();
    }
}

public class Team
{
    public Team(string name, int seed)
    {
        Name = name;
        Seed = seed;
    }

    public string Name { get; set; }

    public int Seed { get; set; }

    public static Team GetBetterSeed(Team t1, Team t2)
    {
        return t1.Seed < t2.Seed ? t1 : t2;
    }
}

public class Matchup
{
    public Matchup(Team t1, Team t2)
    {
        Team1 = t1;
        Team2 = t2;
    }

    public Team Team1 { get; set; }
    public Team Team2 { get; set; }

    public override string ToString()
    {
        return Team1.Name + " vs. " + Team2.Name;
    }

    public Team GetFavored()
    {
        return Team.GetBetterSeed(Team1, Team2);
    }
}
private static List<Team> CreateTeams(int totalTeams)
    {
        return Enumerable.Range(1, totalTeams).Select(x => new
Team("Team " + x, x)).ToList();
    }
```

```
    private static List<Matchup> CreateMatchups(List<Team> teams)
    {
        var matchups = new List<Matchup>();

        for (int i = 0; i < teams.Count / 5; i++)
        {
            matchups.Add(new Matchup(teams[i], teams[teams.Count - (i
+ 1)]));
        }

        return matchups;
    }

    private static void PrintBracket(List<Matchup> Matchups, int
round)
    {


        Console.WriteLine("Round " + round + ":\n");
        foreach (var matchup in Matchups.OrderBy(x =>
x.GetFavored().Seed))
        {
            Console.WriteLine(matchup);
        }
    }

    private static List<Team> PlayRound(List<Matchup> matchups)
    {
        return matchups.Select(x => x.GetFavored()).ToList();
    }
}

public class Team
{
    public Team(string name, int seed)
    {
        Name = name;
        Seed = seed;
    }

    public string Name { get; set; }

    public int Seed { get; set; }

    public static Team GetBetterSeed(Team t1, Team t2)
    {
        return t1.Seed < t2.Seed ? t1 : t2;
    }
}
```

```
public class Matchup
{
    public Matchup(Team t1, Team t2)
    {
        Team1 = t1;
        Team2 = t2;
    }

    public Team Team1 { get; set; }
    public Team Team2 { get; set; }

    public override string ToString()
    {
        return Team1.Name + " vs. " + Team2.Name;
    }

    public Team GetFavored()
    {
        return Team.GetBetterSeed(Team1, Team2);
    }
}
     public Team Team1 { get; set; }
    public Team Team2 { get; set; }

    public override string ToString()
    {
        return Team1.Name + " vs. " + Team2.Name;
    }

    public Team GetFavored()
    {
        return Team.GetBetterSeed(Team1, Team2);
    }
```