Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до курсової роботи

з дисципліни

Об’єктно-орієнтоване програмування

на тему:

**«Аналіз та порівняння гравців Dota 2»**

Виконав: студент ОІ-11

**Онищенко І.Р.**

Прийняв: кандидат технічних наук

Антонів В.Я.

Львів-2023

**Зміст**

Вступ………………………….……………………………………………2

1. Постановка задачі………………………………………………………3

2. Огляд літератури…………………………………………..……………4

3.Опис етапу проектування………………………...…………………….7

4. Програмне рішення.……………………………………..……………10

5. Опис проведених експериментів…………………………………….13

Список використаної літератури……………………...………………..17

Висновок……………………………...………………………………….18

Додатки…………………………………………………………………...20

**Вступ**

У сучасному світі комп'ютерних ігор широко розповсюджені онлайн-ігри, які залучають мільйони гравців з усього світу. Однією з найпопулярніших є Dota 2, яка відома своїм складним геймплеєм та конкурентним середовищем. Велика кількість гравців, що беруть участь у грі, створює потребу в аналізі та порівнянні їх навичок, стратегій та результатів.

Метою цієї курсової роботи є розробка програмного рішення, яке дозволить аналізувати та порівнювати гравців Dota 2. Для цього використовується API, що надає доступ до статистики гравців, результатів ігор та інших даних. Завдяки цьому програмному рішенню можна отримати уявлення про навички та досягнення гравців, визначити їх сильні та слабкі сторони, а також здійснити порівняння між різними гравцями.

У подальшому розвитку проекту можна розглянути додавання нових функцій, таких як прогнозування результатів майбутніх ігор, візуалізація даних або додатковий аналіз впливу різних факторів на результати гри. Це дозволить глибше дослідити гру Dota 2 і надати корисні інструменти для аналізу та вдосконалення геймплею гравців.

**Постановка задачі**

1. **Функціональні можливості програмної розробки:**

* Отримання статистичних даних про гравців, таких як рейтинг, ранг, досягнення тощо.
* Аналіз та порівняння результатів гравців, включаючи кількість перемог та поразок, тривалість ігор, вибір героїв тощо.
* Відображення інформації про гравців у зручному табличному форматі.
* Можливість сортування та фільтрації даних для отримання певних висновків і порівнянь.

1. **Вимоги до програмної розробки:**

* Підтримка доступу до API, яке надає статистичні дані про гравців та їх ігри у Dota 2.
* Інтерактивний та зручний інтерфейс користувача для введення ідентифікаторів гравців та відображення результатів аналізу.
* Сумісність з різними апаратними та програмними платформами, що дозволяє користувачам використовувати програму на різних пристроях.

1. **Вимоги до користувачів програми:**

* Можливість введення ідентифікаторів гравців для отримання їх статистики та результатів ігор.
* Зручний та інтуїтивно зрозумілий інтерфейс для перегляду, сортування та фільтрації даних.
* Повідомлення про будь-які помилки або недійсні ідентифікатори гравців.

**Огляд літератури**

**Предметна область для коду:** ця програма може бути використана для усіх гравців Dota 2, та допоможе краще проаналізувати свою статистику. Навідміну від своїх аналогів програма поки безкоштовна.

**Місце проекту серед схожих проектів:**

* **Dotabuff:** Dotabuff є одним з найпопулярніших веб-сервісів для статистики та аналізу гравців у грі Dota 2. В порівнянні з Dotabuff, мій проект має певну перевагу в графічному інтерфейсі, що дозволяє зручніше взаємодіяти зі статистикою.
* **OpenDota:** OpenDota є веб-сервісом, який надає публічний API для отримання статистики гравців та матчів у грі Dota 2. Мій проект використовує OpenDota API для отримання даних, але може мати перевагу в тому, що він пропонує графічний інтерфейс, що полегшує користування статистикою.
* **Stratz:** Stratz є ще одним веб-сервісом, який надає статистику гравців та матчів у грі Dota 2. Він має своє власне API та графічний інтерфейс. У порівнянні з моїм проектом, Stratz може мати перевагу в розширеному наборі функцій та можливостях для аналізу даних.

**Прототипи та підходи до розв'язання аналогічних задач:**

* **Використання публічних API:** Для отримання даних про гравців, матчі і героїв використовуються публічні API, надані розробниками гри.
* **Аналітика та візуалізація даних:** Є багато методів аналізу та візуалізації даних, які можуть бути застосовані для представлення статистики гравців у зручний спосіб. Це можуть бути графіки, діаграми, кластерний аналіз та інші методи візуалізації даних.
* **Машинне навчання та штучний інтелект:** Можна застосувати методи машинного навчання і штучного інтелекту для аналізу статистики гравців Dota 2. Наприклад, можна побудувати моделі прогнозування результатів матчів, рекомендаційні системи для вибору героїв або класифікаційні моделі для визначення стилю гри гравця так як це представлено у платній підписці Dota PLUS.
* **Обробка текстових даних:** Дані про матчі та гравців часто мають текстовий формат, такі як описи героїв, коментарі до матчів або профілі гравців. Застосування методів обробки текстових даних, таких як аналіз тональності, екстракція ключових слів або кластеризація тексту, може допомогти у виявленні цікавих залежностей та трендів.
* **Веб-скрапінг:** Якщо не вдається отримати потрібні дані за допомогою публічних API, можна використовувати техніку веб-скрапінгу для отримання інформації з веб-сайтів, таких як форуми, блоги або соціальні медіа. Це може бути корисно для отримання додаткової статистики або оцінок гравців.

**Опис етапу проектування**

1. **Загальна архітектура:**

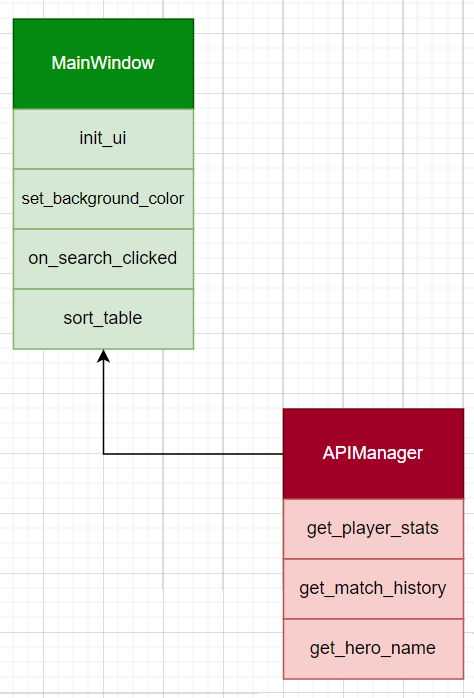
* Код написаний мовою Python з використанням бібліотеки PyQt5 для розробки графічного інтерфейсу користувача.
* Головний клас *MainWindow* успадковує клас *QWidget* і представляє вікно програми.
* Клас *APIManager* містить статичні методи для взаємодії з API Dota 2 для отримання статистики гравця та історії його матчів.

1. **Розбиття задачі на підзадачі:**

* **Розробка графічного інтерфейсу:**
* Ініціалізація головного вікна та його віджетів, налаштування стилів та кольорів.
* Розміщення віджетів у вікні з використанням різних контейнерів для організації розташування елементів.
* Додавання прокручування до вікна для підтримки великої кількості даних.
* **Взаємодія з API Dota 2:**
* Реалізація статичних методів в класі *APIManager* для отримання статистики гравця, історії його матчів та назв героїв.
* Використання бібліотеки requests для виконання HTTP-запитів до API.
* **Обробка подій:**
* При натисканні кнопки ***"Search Player Stats"*** відбувається обробка події, де отримується введений Steam ID гравця та запит до API для отримання статистики та історії матчів.
* Відображення отриманих даних у відповідних віджетах, таких як етикетки, таблиця тощо.
* Сортування таблиці за різними стовпцями при кліку на заголовок стовпця.

1. **Опис окремих частин програмної системи:**

* **Головне вікно (*MainWindow*):**
* Містить різні віджети, такі як етикетки, текстові поля, кнопки та таблицю.
* Налаштовує стилі та кольори вікна.
* Відповідає за обробку подій користувача та взаємодію з іншими частинами системи.
* **Клас *APIManager*:**
* Надає методи для взаємодії з API Dota 2 та отримання різних видів даних, таких як статистика гравця, історія матчів та назви героїв.
* **Таблиця матчів (*match\_history\_table*):**
* Відображає інформацію про останні 20 матчів гравця.
* Містить стовпці зі значеннями, такими як ID матчу, герой, тривалість та результат.
* Надає можливість сортування за різними стовпцями.

**рис.** Діаграма класів

**Програмне рішення**

Детальний опис роботи над програмною реалізацією включає наступні етапи:

1. **Імпорт бібліотек:** В коді використовуються наступні бібліотеки:

* ***sys*:** модуль системних функцій Python.
* ***PyQt5.QtWidgets*:** модуль для роботи з графічним інтерфейсом користувача (GUI) на основі Qt.
* ***PyQt5.QtGui*:** модуль з функціями для роботи з графічним інтерфейсом користувача (GUI) на основі Qt.
* ***PyQt5.QtCore*:** модуль з основними класами та функціями Qt.
* ***requests*:** бібліотека для виконання HTTP-запитів до API сервісу OpenDota.

1. **Клас *APIManager*:** Цей клас містить статичні методи для взаємодії з API сервісу OpenDota. Методи *get\_player\_stats, get\_match\_history та get\_hero\_name* використовуються для отримання даних про гравця, його матчів та назви героїв в грі Dota 2 відповідно. Вони використовують модуль requests для здійснення HTTP-запитів до API сервісу OpenDota і отримання відповідей у форматі JSON.
2. **Клас MainWindow:** Цей клас успадковує клас *QWidget* з модуля *PyQt5.QtWidgets* і є головним вікном програми. Він містить методи для налаштування та керування графічним інтерфейсом користувача (GUI).
3. **Графічний інтерфейс користувача (GUI):** Код містить налаштування різних елементів GUI, таких як мітки, кнопки, таблиця тощо. Він також включає обробники подій, які виконуються при взаємодії користувача з елементами GUI, наприклад, при натисканні кнопки ***"Search Player Stats"***.
4. **Взаємодія з API сервісу OpenDota:** При натисканні кнопки ***"Search Player Stats"*** викликається метод *on\_search\_clicked*, який отримує введений користувачем Steam ID гравця та використовує клас *APIManager* для отримання статистики гравця та його останніх матчів. Отримані дані відображаються у відповідних елементах GUI, таких як мітки та таблиця.
5. **Сортування таблиці:** В коді реалізовано сортування таблиці матчів за різними стовпцями. При натисканні на заголовок стовпця в таблиці викликається метод *sort\_table*, який сортує дані в таблиці за вибраним стовпцем і змінює порядок сортування.
6. **Обробка отриманих даних та відображення в графічному інтерфейсі:**

* Після отримання статистики гравця та його останніх матчів від API сервісу OpenDota, дані обробляються для відображення у вікні програми.
* Інформація про гравця, така як аватар, нікнейм, рейтинг тощо, відображається у відповідних елементах GUI, таких як мітки.
* Останні матчі гравця представляються у таблиці з відповідними стовпцями, такими як "Match ID", "Hero", "Duration" та "Result".
* Для кожного матчу, ID героя конвертується у його назву за допомогою методу get\_hero\_name класу APIManager, і ця назва відображається в таблиці замість ID.
* Додаткова обробка даних може включати форматування тривалості матчу у зручний формат (наприклад, години та хвилини), а також позначення результату матчу (перемога, поразка, нічия) за допомогою відповідних символів або кольорів.

1. **Обробка помилок та повідомлення користувача:**

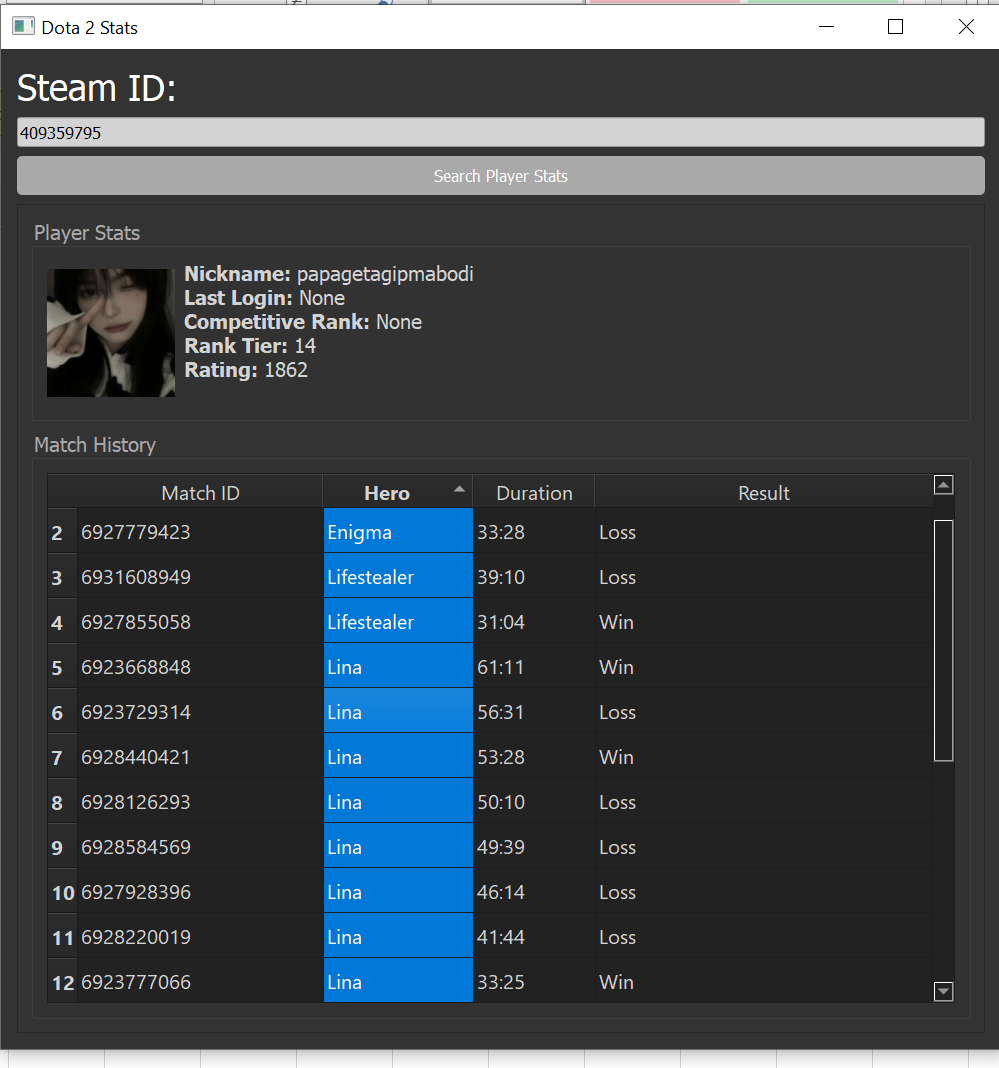
* Під час взаємодії з API сервісу OpenDota можуть виникати помилки, такі як невірний Steam ID, відсутність з'єднання з мережею або невдалий запит.
* Код містить обробку таких помилок та повідомлення користувача про них.
* Наприклад, якщо статистика гравця або матчі не можуть бути отримані, відображається відповідне повідомлення у вікні програми, щоб повідомити користувача про помилку та просити перевірити введений Steam ID або з'єднання з Інтернетом.

**Опис проведених експериментів**

Експериментальна частина коду представлена в класі MainWindow. У цьому розділі описуються дії, необхідні для використання програмної розробки з різних перспектив: користувача, адміністратора та програміста.

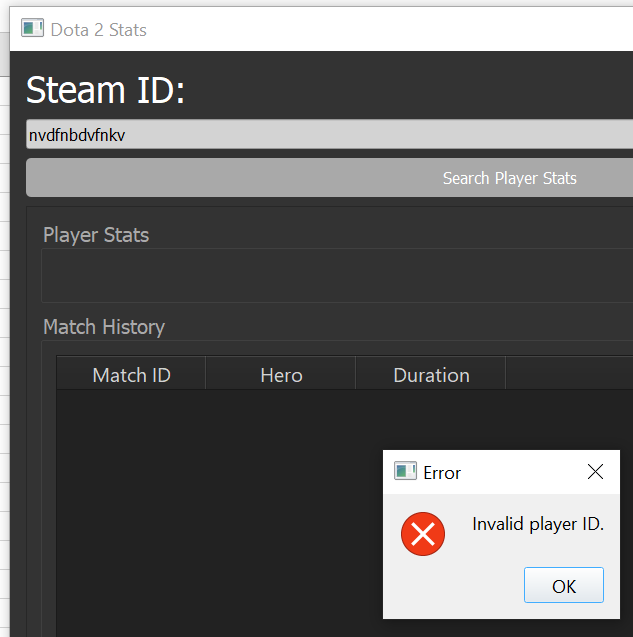
1. **Користувач:** Користувачу потрібно ввести Steam ID у відповідне поле введення (*player\_id\_input*) і натиснути кнопку ***"Search Player Stats"***. Після цього відбувається запит до API, отримані дані про гравця відображаються на екрані.
2. **Адміністратор:** Адміністратор повинен мати доступ до програмного коду і встановити необхідні залежності, зокрема *PyQt5* і бібліотеку requests. Після цього можна запустити програму, виконавши файл коду.
3. **Програміст:** Програміст мати доступ до коду. Після цього можна продовжити роботу над проектом, зробити необхідні зміни і запустити програму.

**Програмна реалізація:**

****

При натисканні на певний головний ствопчик бачимо, що дані сортуюються при другому натискані дані сортуюються за протилежним порядком.

**Контроль помилкових введень:**

****

Як бачимо програма працює, якщо ввести правильно дані у іншому випадку, нам висвітлюється повідомлення з помилкою. Потрібно зачекати певний час(10 с.), щоб програма відобразила статистику, так як витрачається час на запит API.

**Список використаних джерел**

Документація PyQt5: <https://doc.qt.io/qtforpython-6/>

Документація OpenDota API: <https://docs.opendota.com/>

Документація Python: <https://docs.python.org/>

Stack Overflow: [https://stackoverflow.com/](%20https:/stackoverflow.com/)

Python.org: <https://www.python.org/>

Qt Documentation: [https://doc.qt.io/](%20https:/doc.qt.io/)

PyQt5: <https://github.com/baoboa/pyqt5>

Документація бібліотеки requests: <https://docs.python-requests.org/>

Документація бібліотек Python: <https://docs.python.org/3/library/>

**Висновок**

1. **Огляд задачі:**

* У рамках курсової роботи було розроблено програмну систему, яка використовує API для отримання статистики гравця та його історії матчів у грі Dota 2.
* Система дозволяє користувачам шукати та переглядати статистику гравця, таку як нікнейм, останній вхід, рейтинг, ранг тощо, а також переглядати деталі останніх 20 матчів гравця, включаючи ID матчу, назву героя, тривалість та результат.
* Задача полягала у створенні зручного та інтуїтивно зрозумілого інтерфейсу користувача, який дозволяє швидко та ефективно отримувати статистику гравця.

1. **Обмеження та перспективи розвитку:**

* Наразі програмна система обмежується використанням API для отримання даних з гри Dota 2. Це означає, що можливості системи обмежені функціями, які надає API. Якщо API оновлюється або додаються нові функції, програмна система може бути розширена для використання цих можливостей.
* Одним з можливих напрямків розвитку є розширення функціональності програмної системи, додавання нових функцій, таких як отримання статистики команд, аналіз даних та прогнозування результатів матчів. Такі розширення додадуть більше цінності та корисності для користувачів системи.

1. **Набуті знання та вміння:**

* Розробка цієї програмної системи дозволила набути досвіду роботи з *PyQt5*, бібліотекою для розробки графічного інтерфейсу користувача на мові Python. Це включало створення вікон, кнопок, таблиць та інших елементів інтерфейсу, а також обробку подій.
* Було освоєно використання API, зокрема роботу з запитами HTTP та обробку відповідей у форматі JSON. Це відкрило можливість взаємодії з зовнішніми сервісами та отримання актуальних даних для відображення користувачу.
* Під час розробки програмної системи були вдосконалені навички програмування на мові Python, включаючи роботу з функціями, класами, рядками та списками. Було отримано досвід управління даними та їх обробкою.

У результаті розробки програмної системи для отримання статистики гравця та його історії матчів в грі Dota 2, було створено функціональну систему, яка забезпечує користувачам зручний спосіб отримання актуальних даних про гравця та його гри. Додатково, розробка системи дозволила поглибити знання та вміння в галузі розробки графічних інтерфейсів, використання API та роботи з даними в Python.

**Додатки**

**Програма:**

**import sys**

**from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton, QGroupBox, QScrollArea, QHBoxLayout, QMessageBox, QTableWidget, QTableWidgetItem**

**from PyQt5.QtGui import QFont, QPixmap, QColor, QPalette**

**from PyQt5.QtCore import Qt**

**import requests**

**class APIManager:**

**@staticmethod**

**def get\_player\_stats(player\_id):**

**url = f"https://api.opendota.com/api/players/{player\_id}"**

**response = requests.get(url)**

**data = response.json()**

**return data**

**@staticmethod**

**def get\_match\_history(player\_id):**

**url = f"https://api.opendota.com/api/players/{player\_id}/matches"**

**response = requests.get(url)**

**data = response.json()**

**return data**

**@staticmethod**

**def get\_hero\_name(hero\_id):**

**url = f"https://api.opendota.com/api/heroes"**

**response = requests.get(url)**

**data = response.json()**

**for hero in data:**

**if hero["id"] == hero\_id:**

**return hero["localized\_name"]**

**return None**

**class MainWindow(QWidget):**

**def \_\_init\_\_(self):**

**super().\_\_init\_\_()**

**self.init\_ui()**

**self.sort\_order = Qt.DescendingOrder**

**self.last\_sorted\_column = None**

**def init\_ui(self):**

**self.setWindowTitle("Dota 2 Stats")**

**layout = QVBoxLayout()**

**self.set\_background\_color("#333333")**

**font = QFont()**

**font.setPointSize(18)**

**player\_id\_label = QLabel("Steam ID:", self)**

**player\_id\_label.setStyleSheet("color: white;")**

**player\_id\_label.setFont(font)**

**layout.addWidget(player\_id\_label)**

**self.player\_id\_input = QLineEdit(self)**

**self.player\_id\_input.setStyleSheet("background-color: lightgrey;")**

**self.player\_id\_input.setPlaceholderText("Enter Steam ID")**

**layout.addWidget(self.player\_id\_input)**

**search\_button = QPushButton("Search Player Stats", self)**

**search\_button.setStyleSheet("background-color: darkgray; color: white; padding: 10px; border-radius: 5px; font-size: 16px;")**

**search\_button.clicked.connect(self.on\_search\_clicked)**

**layout.addWidget(search\_button)**

**player\_stats\_group = QGroupBox("Player Stats")**

**player\_stats\_group.setStyleSheet("color: darkgray; font-size: 20px;")**

**player\_stats\_layout = QVBoxLayout()**

**player\_info\_layout = QHBoxLayout()**

**self.avatar\_label = QLabel(self)**

**self.avatar\_label.setMaximumSize(128, 128)**

**player\_info\_layout.setAlignment(Qt.AlignTop)**

**player\_info\_layout.addWidget(self.avatar\_label)**

**self.player\_stats\_label = QLabel(self)**

**self.player\_stats\_label.setStyleSheet("color: darkgray; font-weight: bold;")**

**player\_info\_layout.addWidget(self.player\_stats\_label)**

**player\_stats\_layout.addLayout(player\_info\_layout)**

**player\_stats\_group.setLayout(player\_stats\_layout)**

**match\_history\_group = QGroupBox("Match History")**

**match\_history\_group.setStyleSheet("color: darkgray; font-size: 20px;")**

**match\_history\_layout = QVBoxLayout()**

**self.match\_history\_table = QTableWidget()**

**self.match\_history\_table.setColumnCount(4)**

**self.match\_history\_table.setHorizontalHeaderLabels(["Match ID", "Hero", "Duration", "Result"])**

**self.match\_history\_table.horizontalHeader().setStretchLastSection(True)**

**self.match\_history\_table.setStyleSheet("color: lightgrey; background-color: #222222;")**

**palette = self.match\_history\_table.palette()**

**palette.setColor(QPalette.Text, Qt.white)**

**self.match\_history\_table.setPalette(palette)**

**self.match\_history\_table.horizontalHeader().sectionClicked.connect(**

**self.sort\_table)**

**match\_history\_layout.addWidget(self.match\_history\_table)**

**match\_history\_group.setLayout(match\_history\_layout)**

**scroll\_area = QScrollArea(self)**

**scroll\_area.setWidgetResizable(True)**

**scroll\_content = QWidget(scroll\_area)**

**scroll\_layout = QVBoxLayout(scroll\_content)**

**scroll\_layout.addWidget(player\_stats\_group)**

**scroll\_layout.addWidget(match\_history\_group)**

**scroll\_content.setLayout(scroll\_layout)**

**scroll\_area.setWidget(scroll\_content)**

**layout.addWidget(scroll\_area)**

**self.setLayout(layout)**

**self.setGeometry(1000, 1000, 1000, 1000)**

**self.show()**

**def set\_background\_color(self, color):**

**palette = QPalette()**

**palette.setColor(QPalette.Window, QColor(color))**

**self.setPalette(palette)**

**def on\_search\_clicked(self):**

**player\_id = self.player\_id\_input.text()**

**player\_stats = APIManager.get\_player\_stats(player\_id)**

**if "profile" in player\_stats:**

**player\_stats\_text = ""**

**if "profile" in player\_stats and "avatarfull" in player\_stats["profile"]:**

**avatar\_url = player\_stats["profile"]["avatarfull"]**

**pixmap = QPixmap()**

**pixmap.loadFromData(requests.get(avatar\_url).content)**

**self.avatar\_label.setPixmap(pixmap.scaledToWidth(128, Qt.TransformationMode.SmoothTransformation))**

**if "profile" in player\_stats and "personaname" in player\_stats["profile"]:**

**player\_stats\_text += f"<strong>Nickname:</strong> {player\_stats['profile']['personaname']}<br>"**

**if "profile" in player\_stats and "last\_login" in player\_stats["profile"]:**

**last\_login = player\_stats["profile"]["last\_login"]**

**player\_stats\_text += f"<strong>Last Login:</strong> {last\_login}<br>"**

**if "competitive\_rank" in player\_stats:**

**competitive\_rank = player\_stats["competitive\_rank"]**

**player\_stats\_text += f"<strong>Competitive Rank:</strong> {competitive\_rank}<br>"**

**if "rank\_tier" in player\_stats:**

**rank\_tier = player\_stats["rank\_tier"]**

**player\_stats\_text += f"<strong>Rank Tier:</strong> {rank\_tier}<br>"**

**if "mmr\_estimate" in player\_stats and "estimate" in player\_stats["mmr\_estimate"]:**

**mmr\_estimate = player\_stats["mmr\_estimate"]["estimate"]**

**player\_stats\_text += f"<strong>Rating:</strong> {mmr\_estimate}<br>"**

**self.player\_stats\_label.setText(player\_stats\_text)**

**self.player\_stats\_label.setStyleSheet("color: lightgrey;")**

**match\_history = APIManager.get\_match\_history(player\_id)**

**match\_history\_text = "<strong></strong><br>"**

**if isinstance(match\_history, list):**

**last\_20\_matches = match\_history[:20]**

**self.match\_history\_table.setRowCount(len(last\_20\_matches))**

**for i, match in enumerate(last\_20\_matches):**

**match\_id = match["match\_id"]**

**hero\_id = match["hero\_id"]**

**hero\_name = APIManager.get\_hero\_name(hero\_id)**

**duration = match["duration"]**

**radiant\_win = match["radiant\_win"]**

**result = "Win" if radiant\_win else "Loss"**

**match\_id\_item = QTableWidgetItem(str(match\_id))**

**hero\_name\_item = QTableWidgetItem(hero\_name)**

**duration\_item = QTableWidgetItem(f"{duration // 60}:{duration % 60:02d}")**

**result\_item = QTableWidgetItem(result)**

**match\_id\_item.setFlags(match\_id\_item.flags() & ~Qt.ItemIsEditable)**

**hero\_name\_item.setFlags(hero\_name\_item.flags() & ~Qt.ItemIsEditable)**

**duration\_item.setFlags(duration\_item.flags() & ~Qt.ItemIsEditable)**

**result\_item.setFlags(result\_item.flags() & ~Qt.ItemIsEditable)**

**self.match\_history\_table.setItem(i, 0, match\_id\_item)**

**self.match\_history\_table.setItem(i, 1, hero\_name\_item)**

**self.match\_history\_table.setItem(i, 2, duration\_item)**

**self.match\_history\_table.setItem(i, 3, result\_item)**

**else:**

**QMessageBox.critical(self, "Error", "Invalid player ID.")**

**def sort\_table(self, logical\_index):**

**if logical\_index == 1:**

**if self.last\_sorted\_column == logical\_index:**

**self.sort\_order = Qt.DescendingOrder if self.sort\_order == Qt.AscendingOrder else Qt.AscendingOrder**

**else:**

**self.last\_sorted\_column = logical\_index**

**self.sort\_order = Qt.AscendingOrder**

**self.match\_history\_table.sortItems(logical\_index, self.sort\_order)**

**else:**

**if self.last\_sorted\_column == logical\_index:**

**self.sort\_order = Qt.AscendingOrder if self.sort\_order == Qt.DescendingOrder else Qt.DescendingOrder**

**else:**

**self.last\_sorted\_column = logical\_index**

**self.sort\_order = Qt.DescendingOrder**

**self.match\_history\_table.sortItems(logical\_index, self.sort\_order)**

**header = self.match\_history\_table.horizontalHeader()**

**header.setSortIndicator(logical\_index, self.sort\_order)**

**header.setSortIndicatorShown(True)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app = QApplication(sys.argv)**

**app.setStyle('Fusion')**

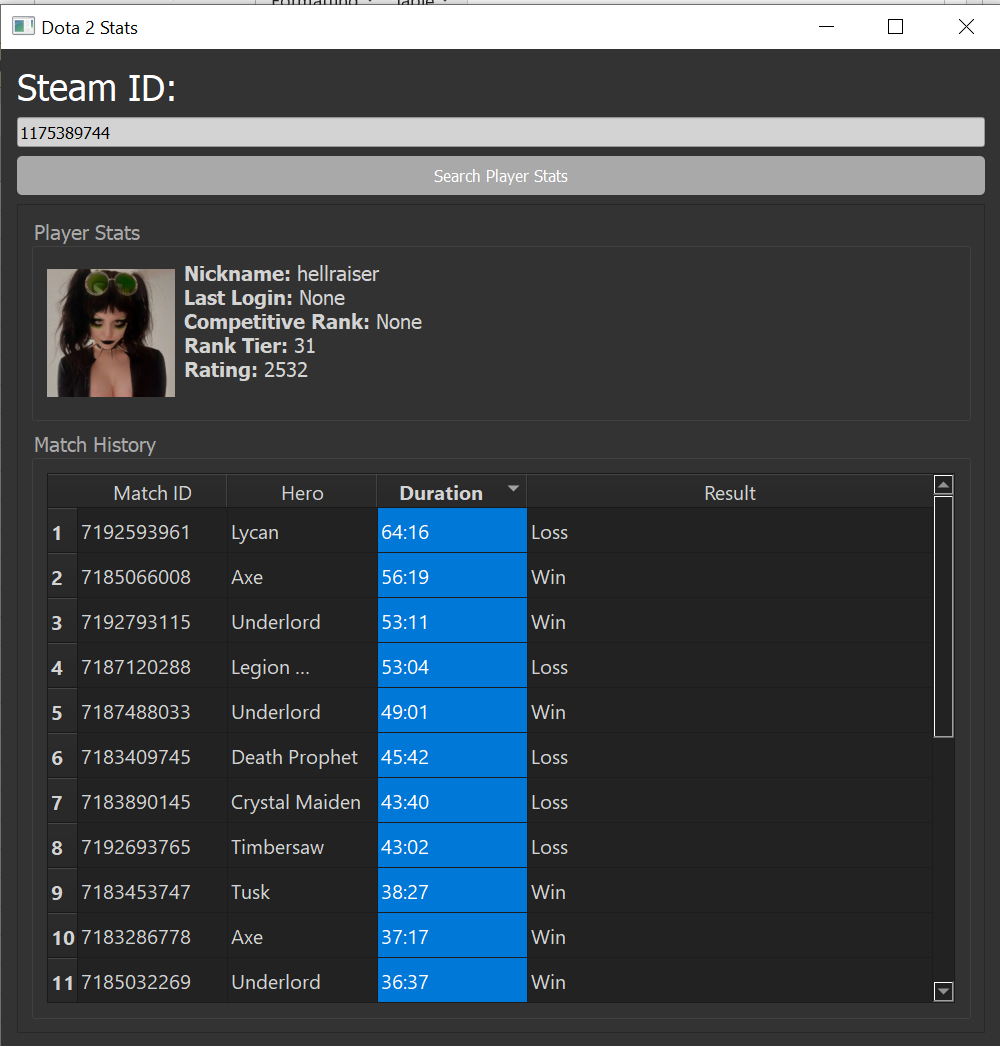
**window = MainWindow()**

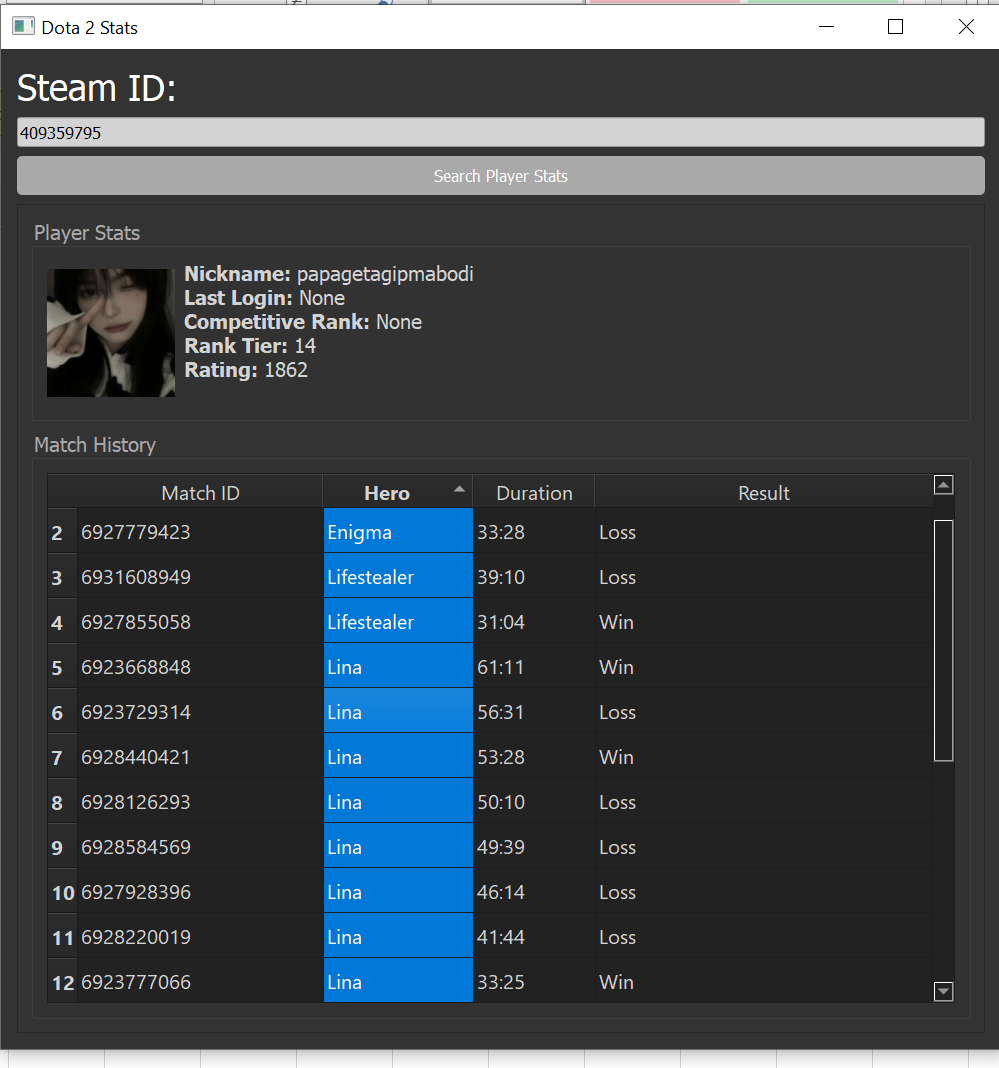
**sys.exit(app.exec\_())**

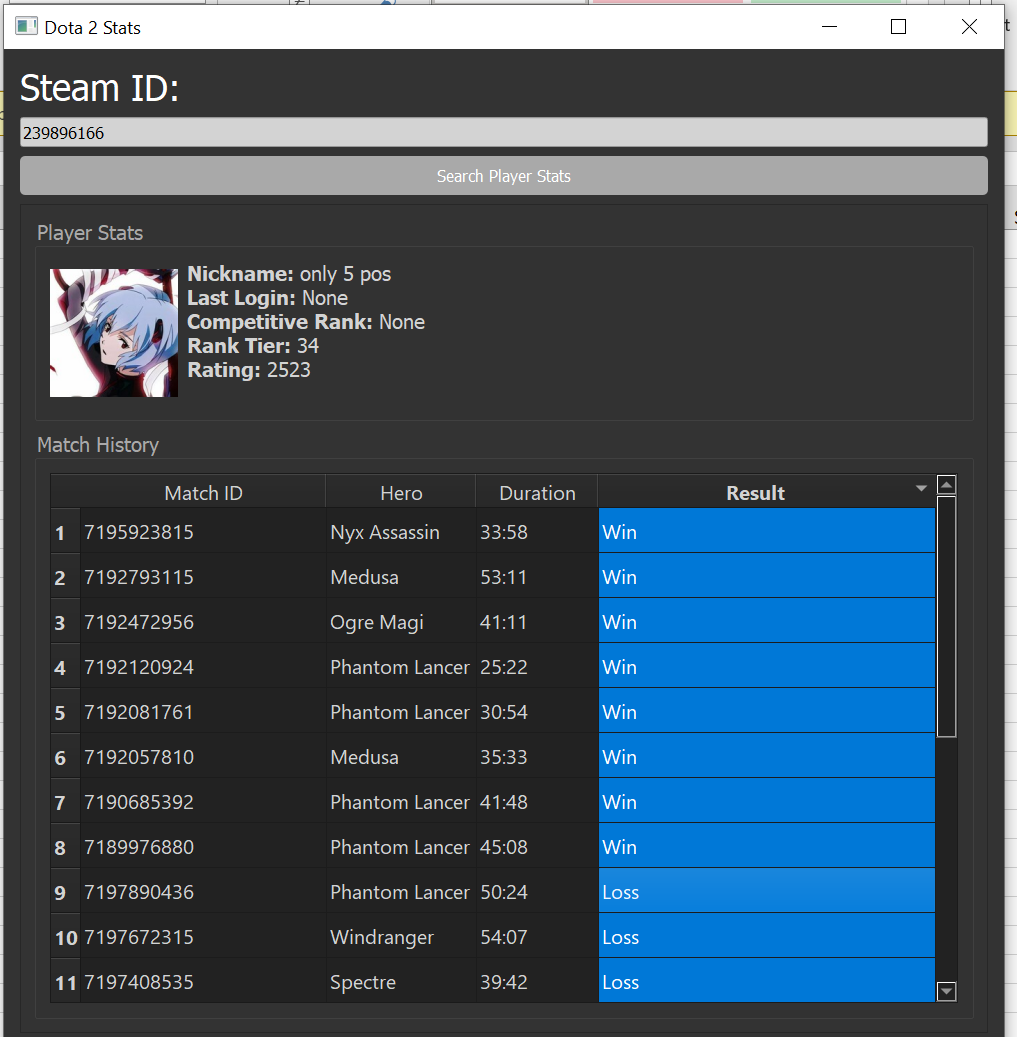
**Вхідні дані:**

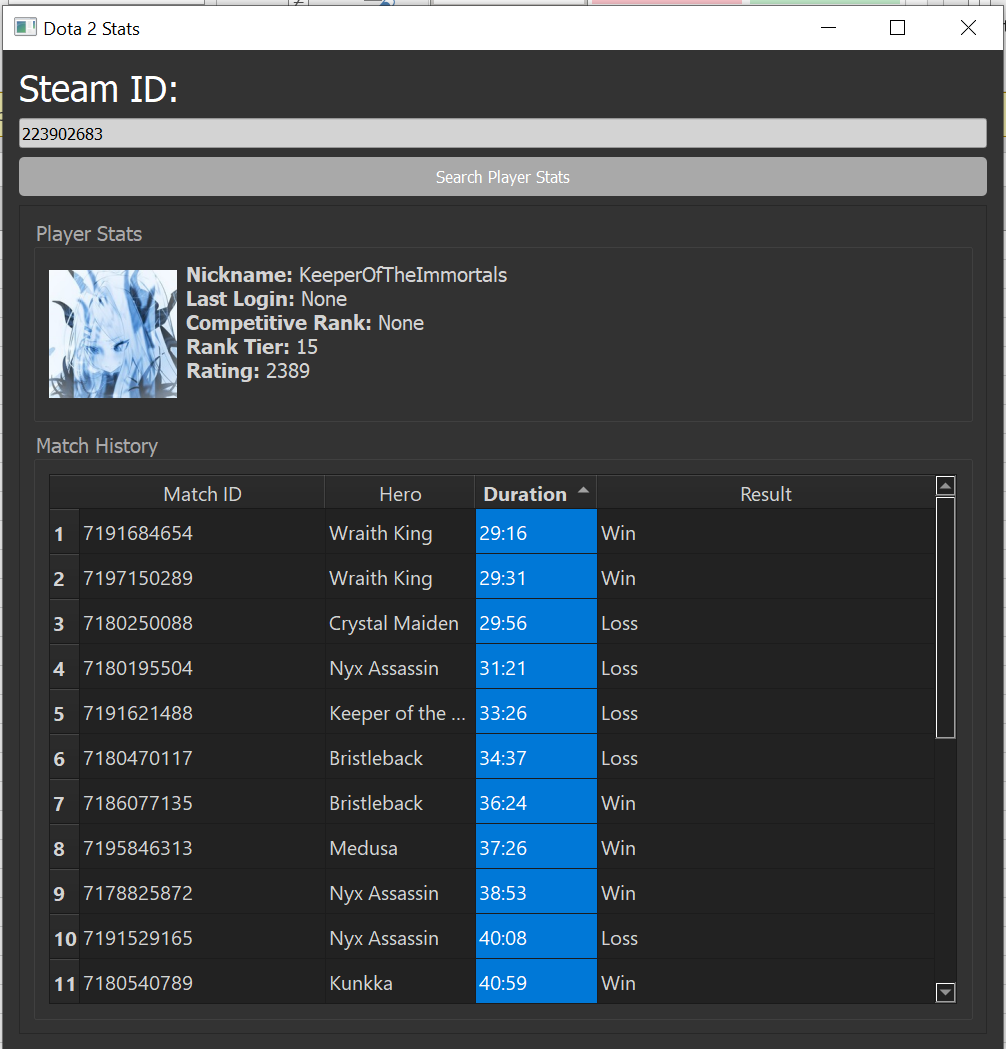
|  |
| --- |
| 1175389744 |
| 409359795 |
| 239896166 |
| 223902683 |

**Програмні реалізації:**

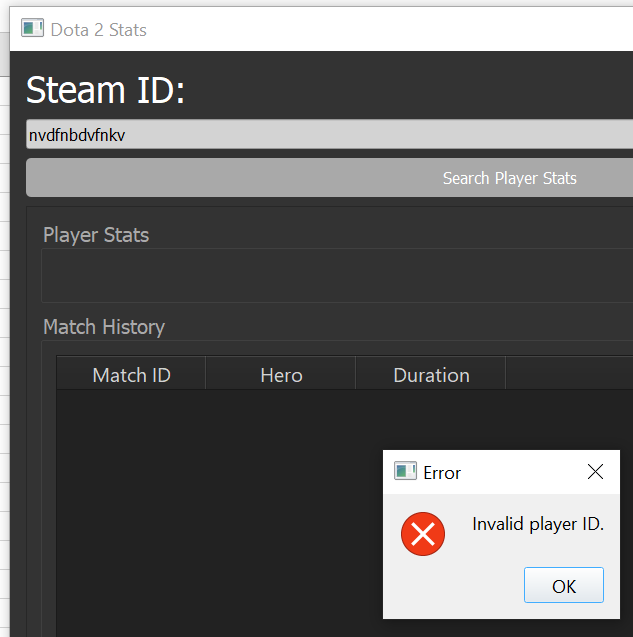
****

****

****

****

**Контроль помилкових введень:**

****