Wojskowa Akademia Techniczna
Im. Jarosława Dąbrowskiego

Projekt z laboratorium NET

Wykonawcy:

- Krzysztof Kobyliński
- Jakub Kowalik

Grupa: WCY18IJ7S1

Prowadzący: mgr. inz. Kamil Małysz

Data wykonania: 25.01.2021

Link do repozytorium: https://github.com/jakub-kowalik/projektLabNet

Spis treści

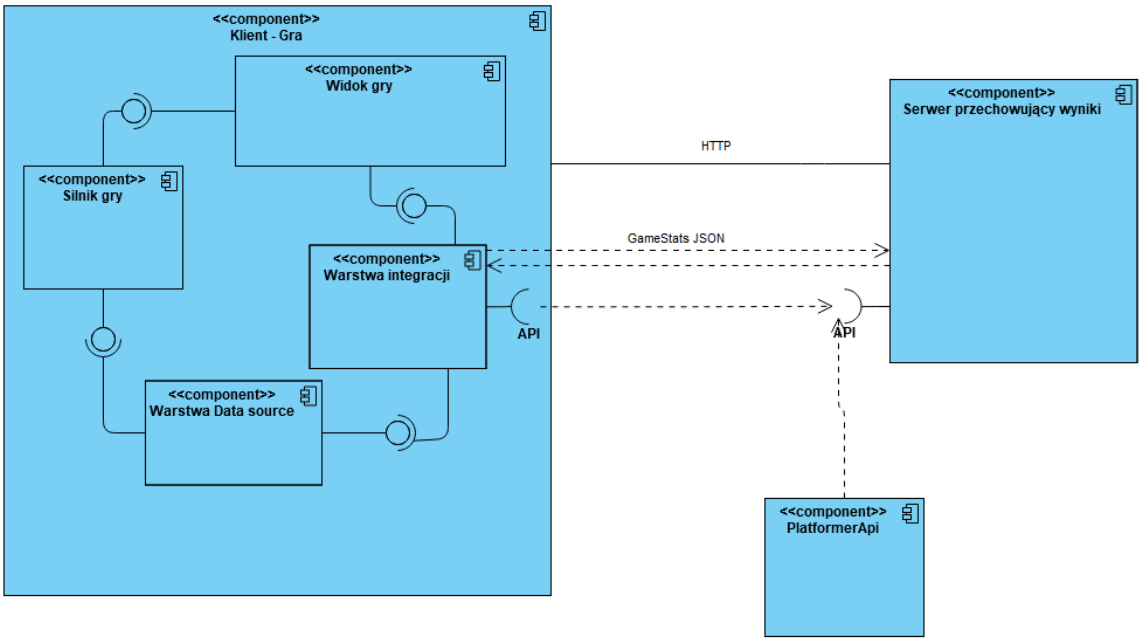
1 Treść zadania

Projektem który zrealizowała nasza grupa jest „gra platformowa w technologii Unity”

2 Opis aplikacji

Aplikacja składa się z 2 części – części właściwej, czyli gry platformowej 2D oraz serwera odpowiadającego za przechowywanie wyników uzyskanych przez graczy. Serwer ten oferuje usługę REST-API, z której korzysta klient gry.

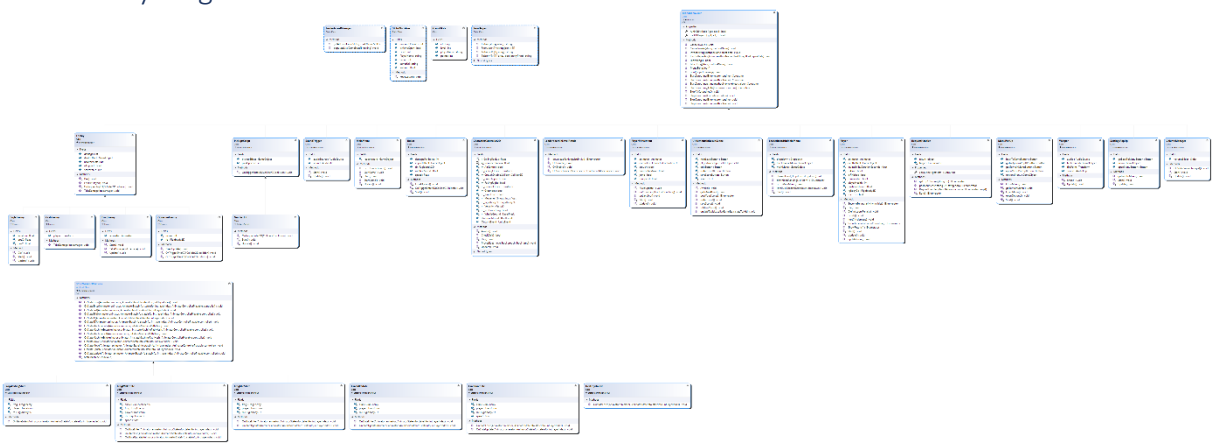
2.1 Diagram komponentów



Rysunek 2.11 Diagram komponentów aplikacji

2.2 Diagram klas gry

2.2.1 Cały diagram klas

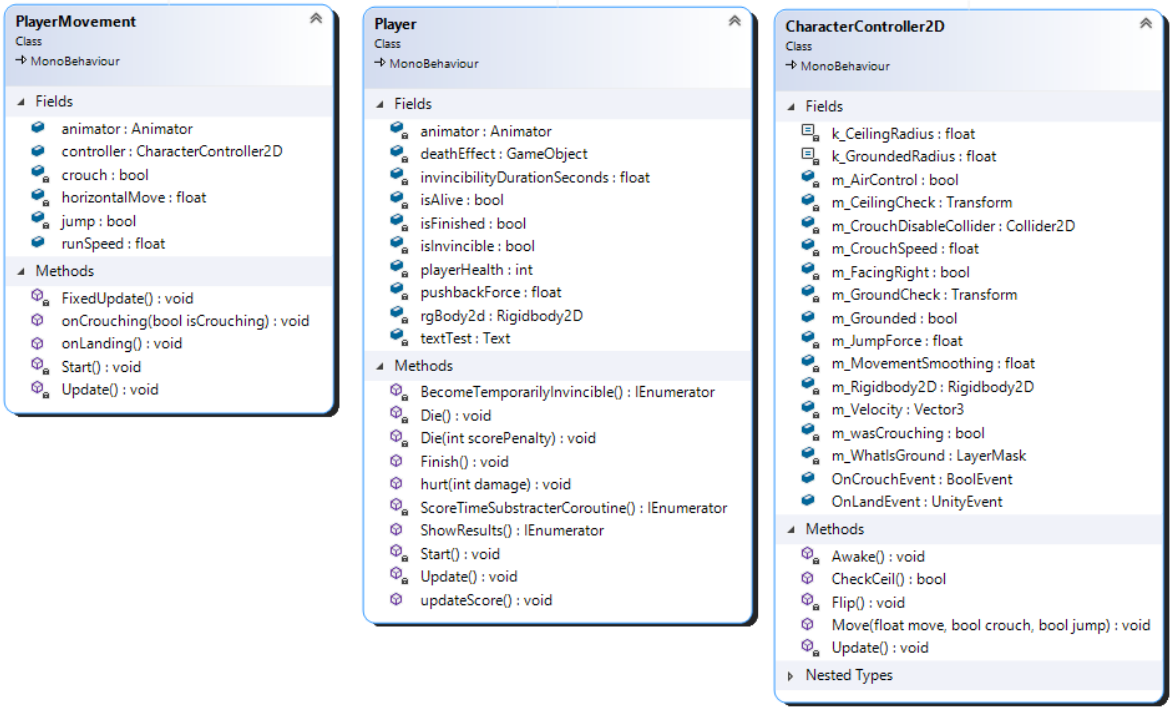


Rysunek 2.2 Diagram klas gry

2.2.2 Klasy obsługujące obiekt gracza

- Klasa CharacterController2D odpowiada za obsługę fizyki obiektu gracza

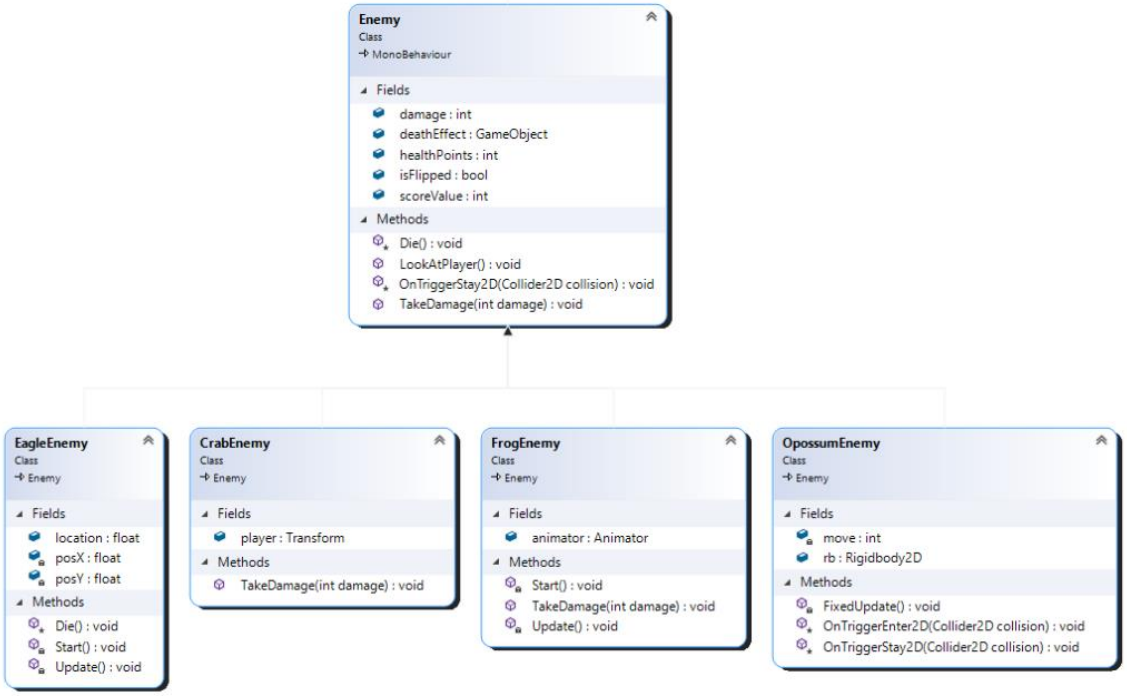
- Klasa PlayerMovement odpowiada za odpowiednie sterowanie obiektem gracza za pomocą klawiatury.
- Klasa Player odpowiada za atrybuty gracza podczas rozgrywki oraz jego interakcje z przeciwnikami.



Rysunek 2.3 Klasy obsługujące gracza

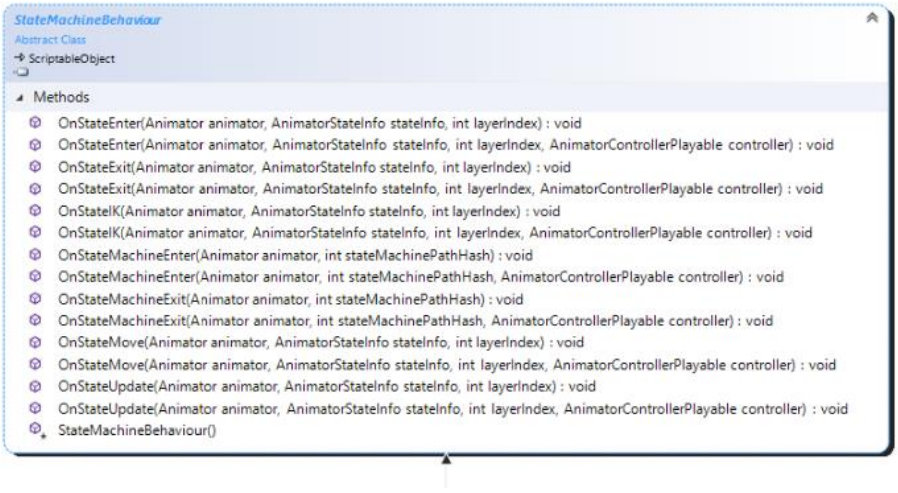
2.2.3 Klasy obsługujące obiekty wrogów

- Klasy dziedziczące po klasie „Enemy” - Klasy te obsługują eventy śmierci, odnoszenia obrażeń, obracania się w kierunku gracza oraz kolizje dla poszczególnych typów wroga.

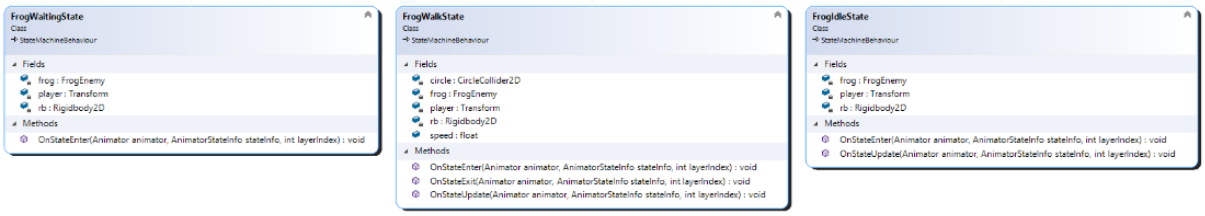


Rysunek 2.4 Klasy dziedziczące po "Enemy"

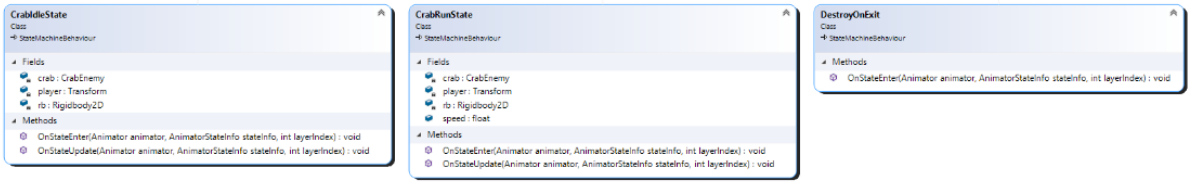
- Klasy dziedziczące po klasie „StateMachineBehaviour” – odpowiadają one za obsługę stanów w jakich znajdują się poszczególni przeciwnicy



Rysunek 2.5 Klasa „MachineStateBehaviour”



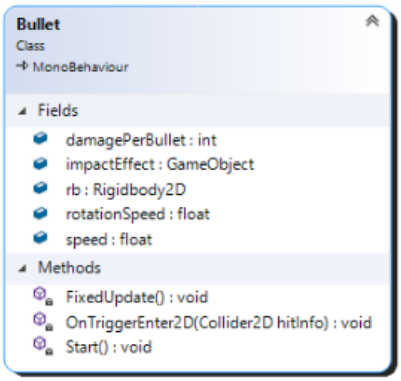
Rysunek 2.6 Klasy dziedziczące po „MachineStateBehaviour”



Rysunek 2.7 Klasy dziedziczące po „MachineStateBehaviour”

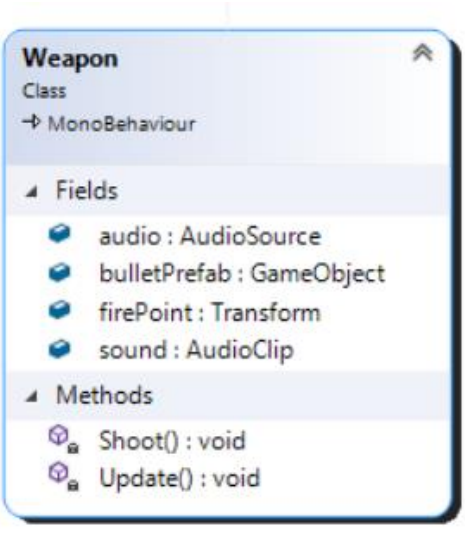
2.2.4 Inne klasy obiektów

- Klasa „Bullet” odpowiada za obsługę pocisków wystrzeliwanych przez gracza.



Rysunek 8 Klasa "Bullet"

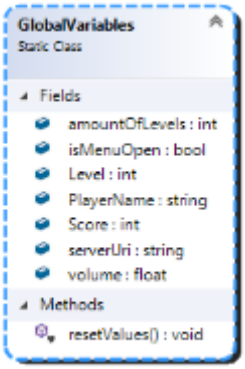
- Klasa „Weapon” odpowiada za obsługę tworzenia wystrzeliwanych pocisków.



Rysunek 9 Klasa "Weapon"

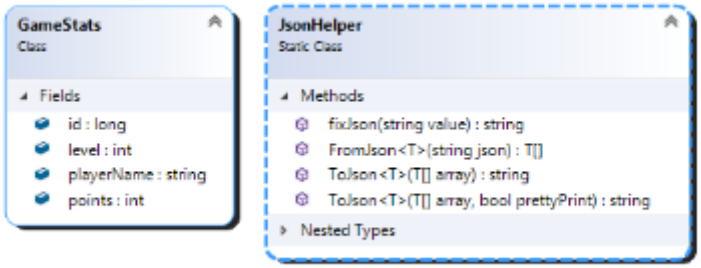
2.2.5 Klasy pomocnicze

- Klasa przechowująca informacje na temat rozgrywki oraz zmienne ogólne:



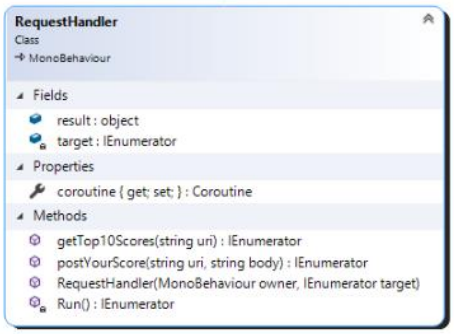
Rysunek 2.10 Klasa „GlobalVariables”

- Klasa „GameStats” przechowuje dane na temat statystyk jednej rozgrywki. To te dane przesyłane są do serwera i z serwera w postaci JSON. Klasa „JsonHelper” wspomaga konwersję z typów tablicowych (w tym przypadku typu „Gamestats[]”.



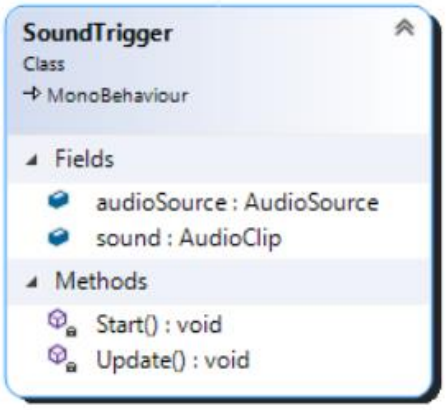
Rysunek 2.11 Klasa "GameStats" oraz "JsonHelper"

- Klasa „RequestHandler” odpowiada za przesyłania zapytań HTTP do serwera. Gra potrzebuje obsługiwać jedynie GET zwracający listę najlepszych wyników z danego poziomu oraz POST przesyłający wynik gracza na serwer.



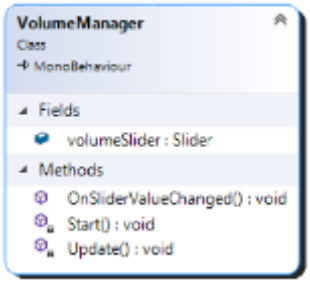
Rysunek 2.12 Klasa „RequestHandler”

- Klasa „SoundTrigger” odpowiada za odtwarzanie dźwięków



Rysunek 2.13 Klasa SoundTrigger

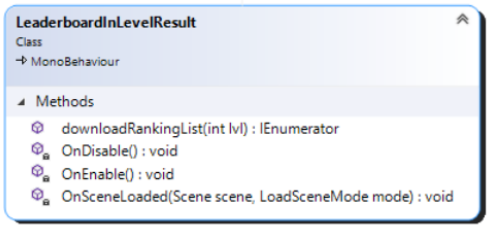
Klasa „VolumeManager” odpowiada za obsługę slidera głośności dźwięku.



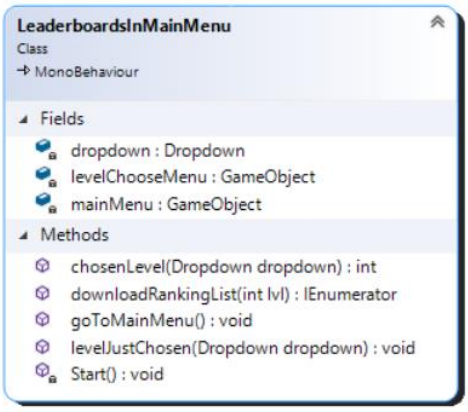
Rysunek 2.14 Klasa VolumeManager

2.2.6 Klasy obsługujące interfejs

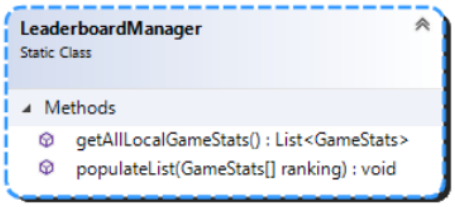
- Klasy odpowiadające za tablice wyników. Klasa „LeaderboardInLevelResult” odpowiada za obsługę tabeli w scenie pomiędzy poziomami. Klasa „LeaderboardsInMainMenu” odpowiada za obsługę tabeli wyników w menu głównym. „LeaderboardManager” natomiast jest klasą statyczną używaną przez obie te klasy w celu wypełnienia tabeli na interfejsie graficznym.



Rysunek 15 Klasa "LeaderboardInLevelResult"

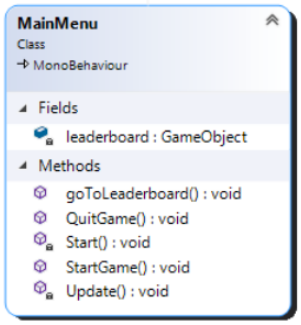


Rysunek 16 Klasa LeaderboardsInMainMenu



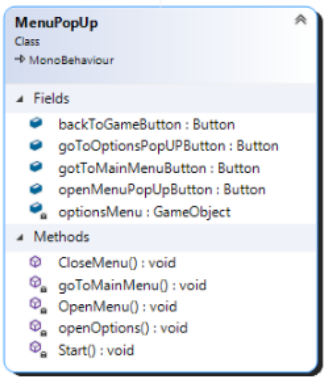
Rysunek 17 Klasa „LeaderboardManager”

- Klasa „MainMenu” odpowiada za obsługę graficznego interfejsu w scenie MainMenu:



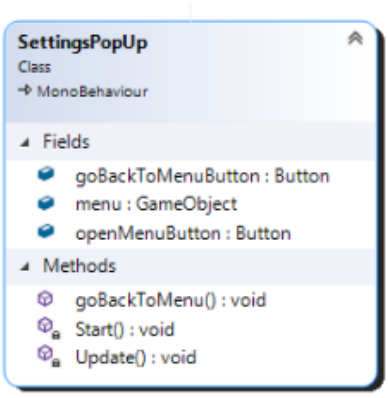
Rysunek 18 Klasa "MainMenu"

- Klasa „MenuPopUp” odpowiada za obsługę okna typu „popUp” wyświetlanego poprzez wciśnięcie przycisku w prawym górnym rogu, w trakcie rozgrywki.



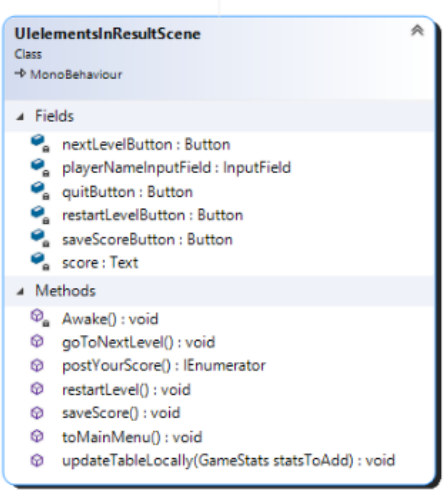
Rysunek 19 Klasa „MenuPopUp”

- Klasa „SettingsPopUp” odpowiada za obsługę okna typu „popUp” wyświetlanego podczas rozgrywki. Okno te zawiera opcje głośności.



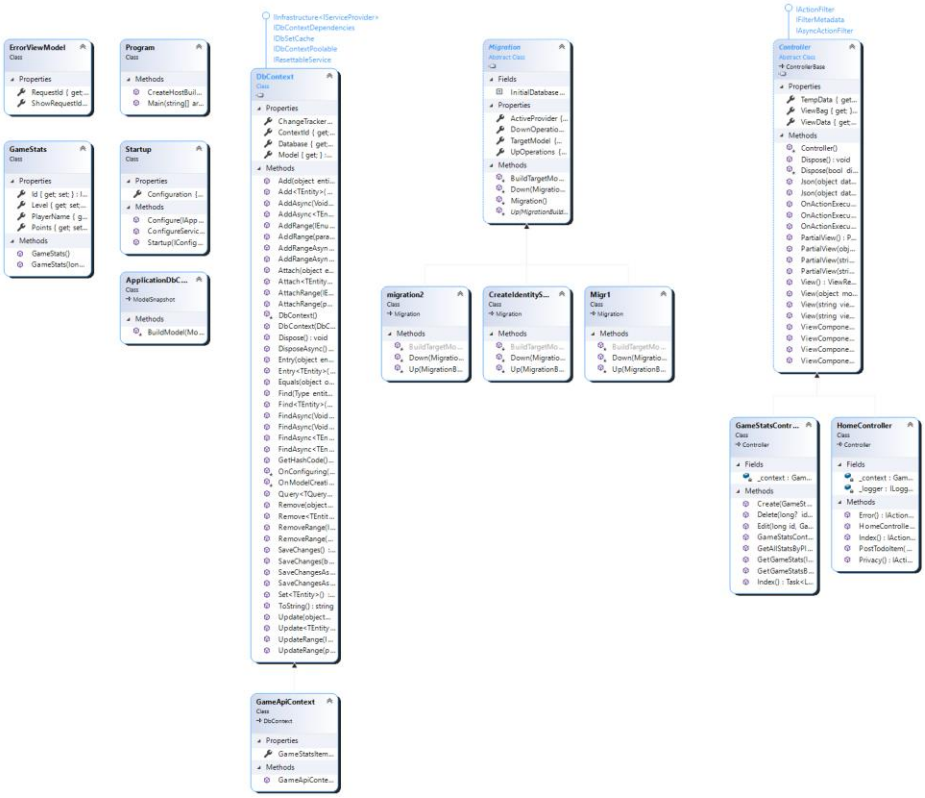
Rysunek 20 Klasa „SettingsPopUp”

- Klasa „UIelementsInResultScene” odpowiada za obsługę elementów interfejsu podczas sceny pomiędzy poziomami.



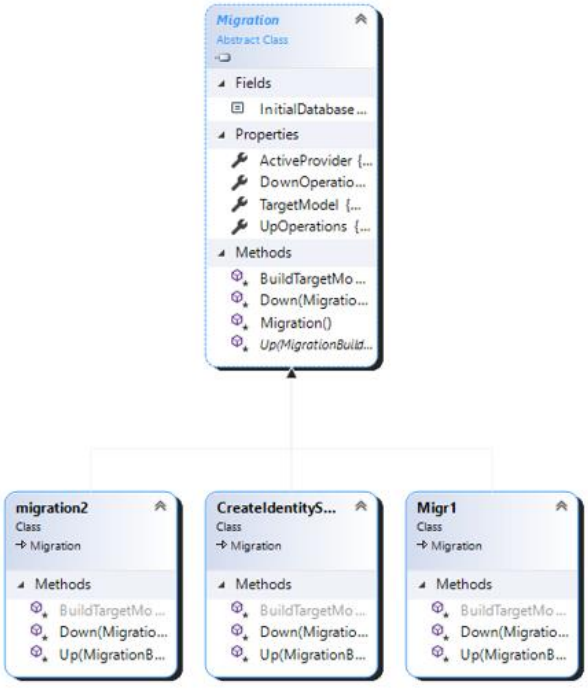
Rysunek 21 Klasa „UIelementsInResultScene”

2.3 Diagram klas serwera



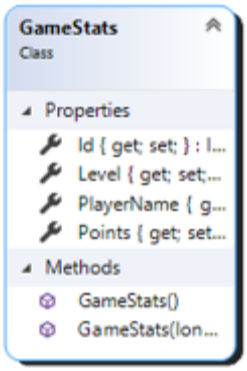
Rysunek 2.22 Diagram klas serwera

- Klasy migracji – dziedziczą one po klasie abstrakcyjnej „Migration”. Opisują one jak zmieniała się baza danych na przestrzeni tworzenia tej aplikacji.



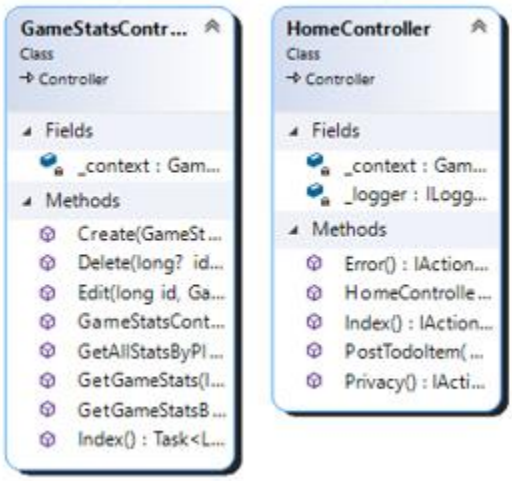
Rysunek 23 Klasy migracyjne

- Klasa modelu danych – w aplikacji istnieje tylko jeden model danych, czyli wyniki uzyskane przez gracza podczas jednej rozgrywki. Model ten opisuje klasa „GameStats”.



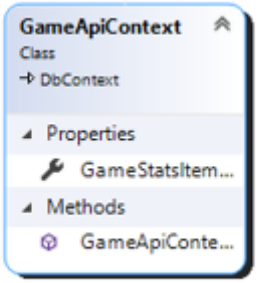
Rysunek 24 Klasa "GameStats"

- Klasy dziedzicząca po „Controller” - HomeController odpowiada za wyświetlanie widoków takich jak „Home” serwera. „GameStatsController” odpowiada natomiast za obsługiwanie zapytań HTTP związanych z modelem danych „GameStats”.



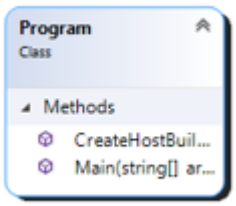
Rysunek 25 Kontrolery REST API

- Klasa dziedzicząca po „DbContext” – „GameApiContext” zawiera definicję DbSet-u przechowywującego dane „GameStats”.

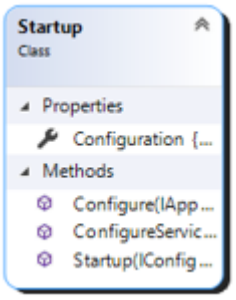


Rysunek 26 Klasa "GameApiContext"

- Klasa „Program” odpowiada za uruchomienie serwera, natomiast klasa „StartUp” odpowiada za jego początkową konfigurację.



Rysunek 27 Klasa "Program"

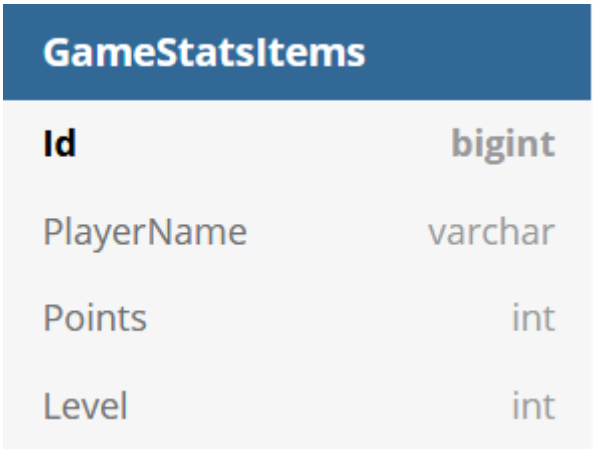


Rysunek 28 Klasa "StartUp"

2.3.1 Diagram bazy danych

W projekcie znajduje się tylko jedna baza danych zawierająca jedna tabele z polami:

- Id
- PlayerName
- Points
- Level



Rysunek 2.29 Diagram bazy danych

Tabela ta służy do przechowywania wyników graczy, które przechowywane są na oddzielnym serwerze, który z kolei przyjmuje je oraz udostępnia za pomocą API.

3 Prezentacja gry

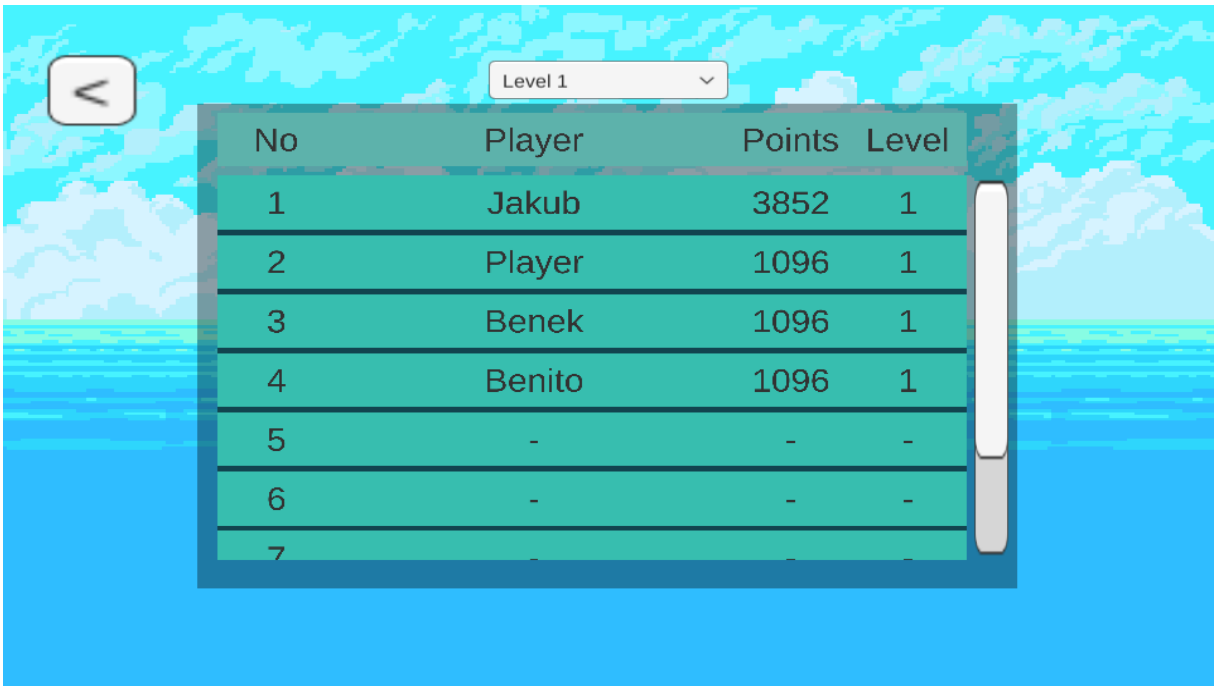
3.1 Ekran startowy

Przy uruchomieniu aplikacji pojawia się ekran menu głównego gry.

Rysunek 3.1 Ekran menu głównego gry.

Z ekranu głównego gry możemy wybrać trzy opcje, rozpoczęcie rozgrywki, tabele wyników oraz zakończenie aplikacji.

Po wybraniu opcji „LEADERBOARD”, pokaże się tabele najlepszych wyników dla wybranego poziomu, wyniki te, jeśli istnieje taka możliwość, pobierane sa z serwera udostepniającego API, jeśli serwer nie jest dostępny, żadne wyniki nie zostaną wyświetlone.



Rysunek 3.2 Ekran tabeli wyników z menu głównego.

3.2 Rozgrywka

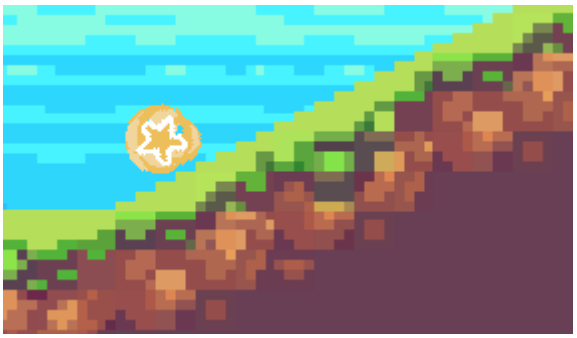
Po wybraniu przycisku „PLAY” rozpocznie się właściwa część gry.

Rysunek 3.3 Ekran rozgrywki

Rozgrywka polega na zebraniu kryształu z planszy przy uzyskaniu przy tym jak najwyższego wyniku. Wynik zmniejsza się z każda sekunda rozgrywki o 1, natomiast można go podnieść poprzez pokonywanie przeciwników oraz zbieranie wiśni.

Gracz w czasie rozgrywki może poruszać się za pomocą klawiszy A, D lub strzałek kierunkowych. Wykonywać skok za pomocą klawisza SPACJA, oraz strzelać za pomocą klawisza Z lub lewego przycisku myszki.

Rysunek 3.4 Strzelanie w grze.



Rysunek 3.5 Efekt rozbicia pocisku o powierzchnie.

Rysunek 3.6 Wiśnia którą gracz może zebrać.

Rysunek 3.7 Efekt zebrania wiśni

3.3 Przeciwnicy

W grze zostały zaimplementowane cztery typy przeciwników, są one następujące:

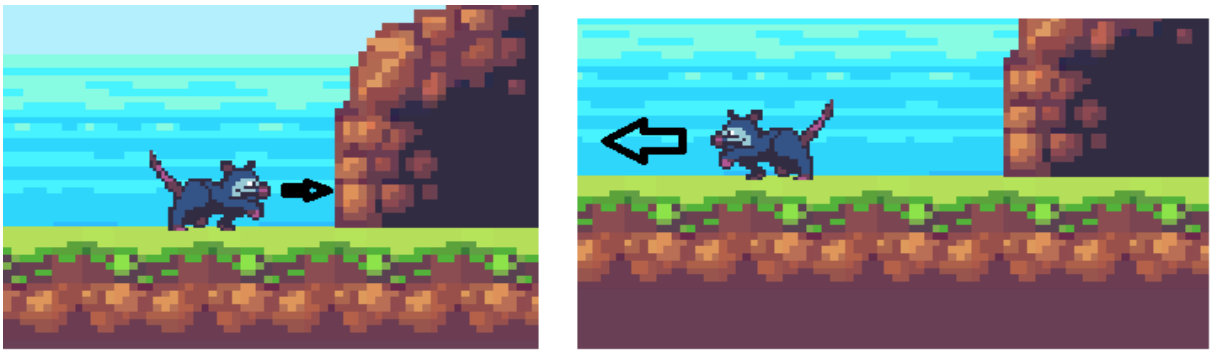
- Ptak – jest to przeciwnik który porusza się do góry i do dołu w stałych odstępach czasu.

Rysunek 3.8 Przeciwnik typu ptak.

- Żaba – jest to przeciwnik który po wykryciu gracza zaczyna podążać w jego stronę, jeśli gracz zniknie z jej pola widzenia, wtedy przestaje go ścigać.

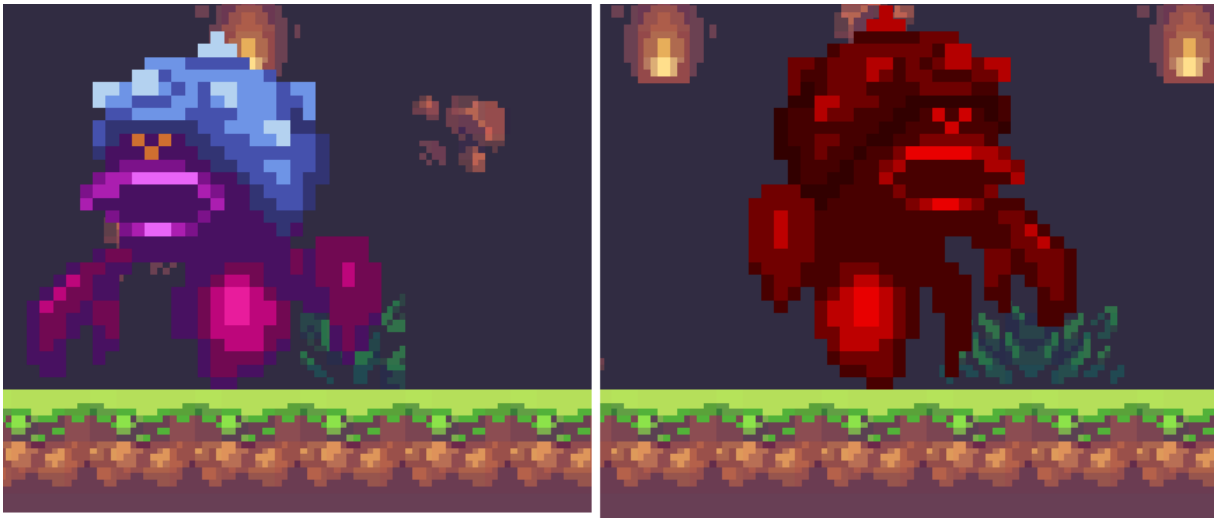
Rysunek 3.9 Przeciwnik typu żaba.

- Kot – jest to przeciwnik który porusza się poziomo po ekranie oraz odbija się od przeszkód i gracza, po odbicie kot zmienia kierunek swojego biegu na przeciwny.



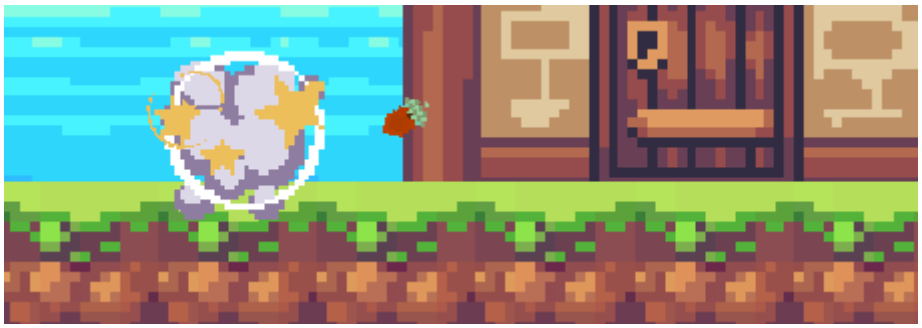
Rysunek 3.10 Przeciwnik typu kot.

- Krab – jest to ostatni i finalny typ przeciwnika. Krab po wykryciu gracza będzie nacierał w jego kierunku. Jeśli gracz zdoła mu uciec, krab przestanie go gonić do czasu aż znowu nie zjawi się w obszarze jego widzenia. Po pewnej ilości obrażeń otrzymanych przez kraba, przejdzie on do drugiej fazy w której zmieni się jego kolor oraz statystyki.



Rysunek 3.11 Przeciwnik typu krab

Ponadto każdy przeciwnik ma efekt śmierci który jest pokazywany po jego pokonaniu



Rysunek 3.12 Efekt śmierci pokonanego przeciwnika

3.4 Ekran wyników

Po zebraniu kryształu przez gracza poziom zostanie ukończony.

Rysunek 3.13 Ukończenie poziomu.

Po ukończeniu poziomu przez gracza pojawi się ekran wyników.

Rysunek 3.14 Ekran wyników

Na ekranie wyników gracz może wpisać swoje imię oraz przesłać swój wynik na serwer, aby został umieszczony w tabeli najlepszych wyników dla poziomu. Pokazane są także najlepsze wyniki dla danego poziomu pobrane z serwera (o ile jest z nim połączenie), przycisk restartu, przycisk następnego poziomu oraz powrót do menu głównego.

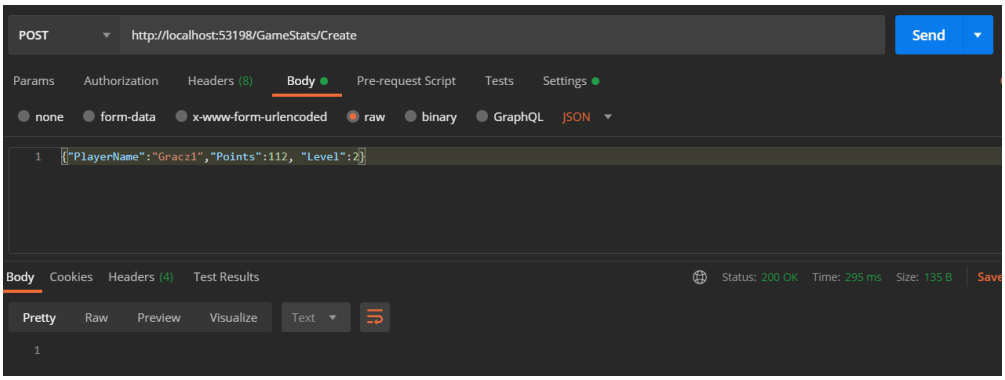
4 Prezentacja serwera

Przedstawimy teraz podstawowe funkcjonalności REST API serwera. Użyjemy do tego narzędzia Postman.

- POST przesyłający obiekt GameStats w postaci JSON

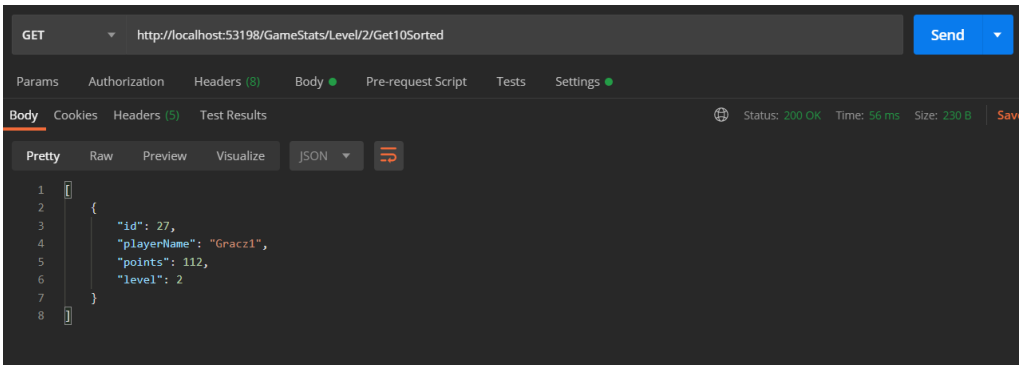
```
[HttpPost]
[Route("~/GameStats/Create")]
0 references
public async Task<IActionResult> Create([FromBody] GameStats gameStats)
{
    if (ModelState.IsValid)
    {
        System.Diagnostics.Debug.WriteLine("Item id= " + gameStats.Id + ","
        _context.Add(gameStats);
        await _context.SaveChangesAsync();
        return Ok();
    }
    else
    {
        return BadRequest();
    }
}
```

Rysunek 15 Kod obsługujący POST GameStats



Rysunek 16 Zapytanie POST GameStats

Poniżej przedstawiam dowód na zapisanie się danych.

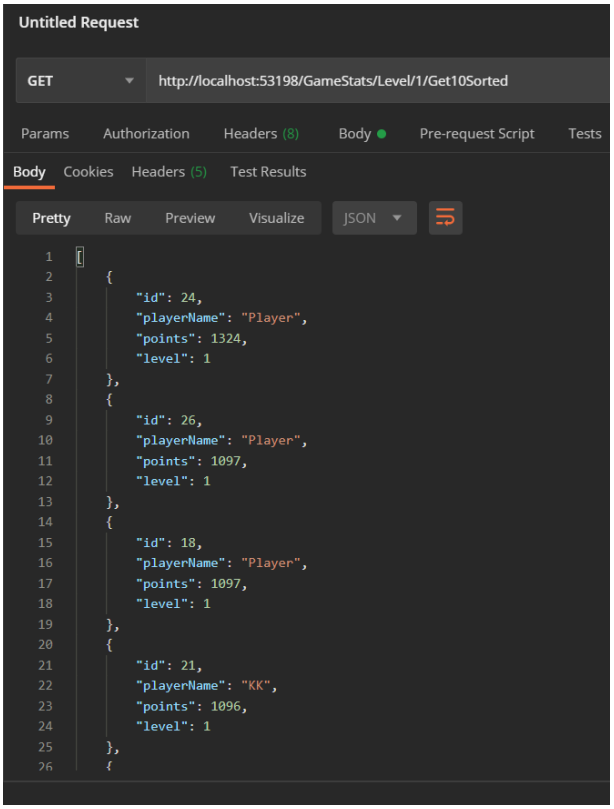


Rysunek 17 Rezultat POST GameStats

- GET zwracający Top 10 wyników z danego poziomu

```
[HttpGet]
[Route("~/GameStats/Level/{lvl}/Get10Sorted")]
0 references
public async Task<List<GameStats>> GetGameStatsByLvlDescendingTop10(string lvl)
{
    return await _context.GameStatsItems
        .Where(o => o.Level == int.Parse(lvl))
        .OrderByDescending(x => x.Points)
        .Take(10)
        .ToListAsync();
}
```

Rysunek 18 Kod obsługujący GET Top 10 GameStats na danym poziomie



Rysunek 19 Zapytanie GET Top 10 GameStats na poziomie 1