

Projekt z Programowania w języku Python oraz Baz Danych

Kamil Pluciński, Tomasz Koszarek

Spis treści

I.	Cel projektu	2
II.	Cechy projektu	2
III.	Organizacja czasowa projektu	3
IV.	Architektura gry	4
V.	Logika biznesowa	4
VI.	Baza danych (autor Kamil Pluciński)	6

I. Cel projektu

Celem projektu jest stworzenie dwuwymiarowej gry o walce czołgami w widoku z lotu ptaka. Rozgrywka polega na sterowaniu czołgiem i walczeniu z przeciwnikami

II. Cechy projektu

1. Główne cechy projektu, które chcemy zaimplementować
 - poruszanie się czołgami
 - strzelanie do innych czołgów
 - mechanika odbijania pocisków (w zależności od wartości pancerza oraz kąta padania pocisku)
 - różne rodzaje broni (działo, miny, rakiety)
 - różne typy pojazdów (ciężkie i wolne, szybkie, bezwieżowe)
 - rozwijanie pojazdu w trakcie walki (tzn. znajdowanie i zbieranie różnych bonusów “leżących” na mapie)
 - przesuwanie kamery zależne od pozycji pojazdu, spowodowane rozmiarem mapy. Domyślnie chcemy, by była większa niż ekran gracza
 - mechanika “znikania” wrogich pojazdów za przeszkodami (znikanie czyli nie będą widoczne dla gracza)
2. Opcjonalne cechy projektu, które chcemy zaimplementować w drugiej kolejności, w zależności od czasu i możliwości
 - własnoręczne grafiki (zaprojektowane od podstaw przez nas, zamiast pochodzących z internetu)
 - dźwięki w grze
 - więcej niż jedna mapa
 - autentykacja graczy umożliwiająca m. in. zbieranie statystyk i rankingi
3. Warianty gry, które rozważamy i które są zależne od naszych możliwości i trudne do określenia teraz
 - wariant z jednym graczem przeciwko botom
 - wariant z dwoma graczami grającymi przeciwko sobie (na jednym komputerze, symultanicznie)
 - wariant multiplayer w sieci lokalnej
 - wariant multiplayer w sieci globalnej (1 vs 1 lub kilku vs kilku)
4. Technologie, biblioteki, z których będziemy korzystać
 - Python
 - biblioteki: *pygame*, *pygame_menu*, *threading*, *socket*
 - MongoDB z biblioteką *mongoengine*

5. Szata graficzna (warianty)

- darmowe tekstury, które znajdziemy w internecie
 - pseudo “z wizją” styl line-art, same kontury
 - cokolwiek, co nam przyjdzie do głowy, np stylistyka modernistyczna, drugowojenna, etc.
- Generalnie nie ma na to konkretnego pomysłu, nie jest to też ważne z punktu widzenia wymagań naszego projektu

III. Organizacja czasowa projektu

lab 3 (29 marca)

silnik gry w miarę gotowy, podstawowe mechaniki są zaimplementowane. Czołg istnieje, wyświetla się, jeździ, strzela

lab 4 (19 kwietnia)

istnieje mapa, na której mieści się czołg, przeszkody, pociski zatrzymują się na ścianach oraz system kolizji obiektów ogólnie działa, system rozwiązywania kolizji również.

Na mapie są przedmioty, które czołg może “podnieść” i zyskać bonus do jakiejś statystyki.

Kamera się przesuwa, podążając za poruszającym się czołgiem.

lab 5 (17 maja)

podjęta jest decyzja o tym, czy w grze są boty, lokalny multiplayer, czy globalny z serwerem. Zaimplementowane są części z tych mechanik.

lab 6 (31 maja)

wszystko z pozostałych rzeczy działa, gra jest prawie gotowa, zrobione są opcjonalne feature’y, wszystkie jakie zdołamy zrobić.

IV. Architektura gry

Podczas tworzenia gry zdecydowaliśmy się na zrealizowanie dwóch wariantów rozgrywki: singleplayer (gra na jednej maszynie) oraz multiplayer (gra przez sieć na dwóch maszynach). Pierwszy wariant umożliwia starcie dwóch graczy korzystając z jednego komputera, przy użyciu z klawiszy strzałek (jeden gracz) oraz klawiszy WSAD (drugi gracz).

Wariant sieciowy korzysta z architektury klient-serwer. Opiera się na programie-serwerze, który musi zostać uruchomiony przed uruchomieniem klientów. Program ten umożliwia połączenie klientów i rozpoczęcie rozgrywki, by finalnie wykonywać wszystkie obliczenia logiki gry, jakimi są przykładowo obliczanie pozycji poruszających się obiektów, wykrywanie kolizji, obliczanie zadanych obrażeń.

Program-klient natomiast udostępnia użytkownikowi graficzny interfejs oraz odpowiada za zebranie informacji o naciśniętych klawiszach i wysłaniu ich do serwera.

System umożliwia połączenie dwóch klientów z serwerem i rozpoczęcie rozgrywki.

Komunikacja sieciowa odbywa się przy pomocy socketów.

Wstępne obsługiwanie klientów na serwerze odbywa się przy pomocy wątków.

V. Logika biznesowa

Czyli krótki opis wzorca Entity-Component-System

Gra oparta jest na systemie ECS (Entity Component System). Wykorzystywany on jest głównie przy tworzeniu gier komputerowych ze względu na dużą elastyczność. Wzorec ten stawia kompozycję ponad dziedziczeniem co odróżnia go od podejścia czysto obiektowego.

Dlaczego zdecydowaliśmy się na takie podejście?

Przy konstruowaniu dużej gry, bogatej w wiele różnorodnych obiektów (postacie, maszyny, otoczenie, interfejs) powstałaby potrzeba zrealizowania skomplikowanej struktury dziedziczenia oraz wielokrotnego sprawdzania właściwego typu obiektu. Przykładowo zdolność do oddziaływania z fizyką jest cechą wspólną zarówno pocisków, czołgów jak i ścian. Każdy z tych obiektów natomiast inaczej na tę fizykę reaguje, bo kolicja ściany z czołgiem nie powinna skutkować odebraniem ścianie punktów życia i wybuchu czołgu, natomiast kolizja pocisku z innym pociskiem nie powinna zatrzymać jednego z nich niczym muru. Naiwnym rozwiązaniem byłoby osobne poruszanie o kolidowanie pocisków, ścian, czołgów itp. Z oczywistych względów jest to złe podejście.

Rozwiązanie stanowi ECS, definiujący każde obiekty w grze jako *Entity*, które poza numerem nie posiadają właściwie żadnych innych informacji.

Są im natomiast przypisane *Komponenty*, (każde entity jest kompozycją kilku komponentów).

Dzięki temu możemy w czasie wykonania podłączać i odłączać różne komponenty do różnych obiektów, oraz, co ważniejsze, możemy łatwiej realizować poszczególne *Systemy* logiki.

Przykładowym systemem jest *System Fizyki*, który iteruje po wszystkich *Komponentach Fizyki*. Nie ma on informacji, czy komponenty nie należą do czołgów czy ścian, dla każdego z komponentów wykonuje odpowiednie operacje.

Podsumowując, logika naszej gry przebiega w czasie każdej klatki w następujący sposób: Każdy z systemów wykonuje swoje zadanie na kolekcji komponentów, które „go interesują”. Dzięki temu unikamy sprawdzania z jakiego typu obiektem mamy do czynienia, równocześnie zyskując możliwość odłączania i dodawania komponentów (cech) obiektom gry.

Opis przykładowego systemu w naszej grze

Najciekawszym systemem jest system fizyki.

Jest on też najbardziej obszerny, z tego względu podzieliliśmy jego działanie na kilka podsystemów:

- Hitbox System (aktualizujący kształty obiektów zdolnych do kolizji)
- Collision Detection System (wykrywający kolizje)
- Collision Resolve System (rozwiązujący kolizje)
- Control System (oddziałujący na pojazd, gdy naciśnięte są klawisze)
- Resistance System (dodający do obiektom opory gruntu i powietrza)
- Integration System (dokonujący zmian położenia obiektów)

Pierwszy z podsystemów zajmuje się uaktualnieniem pozycji wszystkich wierzchołków składających się na figury geometryczne reprezentujące nasze obiekty w symulowanej fizyce.

Następnie wykrywane są kolizje. Zrealizowaliśmy tę funkcjonalność w oparciu o matematyczne twierdzenie SAT (Separated Axis Theorem) pozwalające na rozstrzygnięcie czy dwa wielokąty wypukłe się przecinają (kolidują).

Rozwiązywanie kolizji polega na wybuchach, odejmowaniu HP, odbijaniu, zależnie od obiektów, których kolizję wykryliśmy.

Następnie na wszystkie ciała działają odpowiednie siły, czyli siła silnika i gąsienic (Control System) dla czołgów, czy opory gruntu i powietrza (Resistance System) dla wszystkich obiektów.

Ostatnim krokiem jest obliczenie siły wypadkowej.

Na jej podstawie znajdujemy przyspieszenie wypadkowe, które służy do obliczenia prędkości i finalnie położenia ciała. Obliczenia te z punktu widzenia fizyki są całkowaniem (ze względu na czas), stąd nazwa Integration System.

VI. Baza danych (autor Kamil Pluciński)

Baza danych wraz z serwerem docelowo będą się znajdować na komputerze z Linuxem, ze względu na łatwość zarządzania bazą w lokalnej sieci.

Kolekcje

W bazie będą się znajdowały kolekcje:

- User
- Battle
- Map
- UserInBattle

Kolekcje będą modelowane za pomocą klas dziedziczących z klasy Document z biblioteki mongoengine:

```
class User(me.Document):
    Name = me.StringField(required=True)
    Password = me.StringField(max_length=50)
    def __repr__(self):
        return str(self.id)
```

```
class Battle(me.Document):
    MapName = me.StringField(required=True, max_length=50)
    Date = me.DateField(required=True)
    Winner = me.ReferenceField(User)
    UsersResults = me.ListField(me.ReferenceField(UserInBattle))
    UsersInvolved = me.ListField(me.ReferenceField(User))
```

```
class UserInBattle(me.Document):
    UserId = me.ReferenceField(User)
    NumOfShots = me.DecimalField(min_value=0, max_value=100)
    NumOfShotsOnTarget = me.DecimalField(min_value=0, max_value=100)
    GivenDamage = me.DecimalField(min_value=0, max_value=1000)
    GetDamage = me.DecimalField(min_value=0, max_value=1000)
    OpponentHp = me.DecimalField(min_value=0, max_value=1000)
```

```
class Map(me.Document):
    MapName = me.StringField(max_length=50, unique=True)
    SrcPath = me.StringField(required=True)
```

```
InitPos1 = me.fields.PointField()
InitPos2 = me.fields.PointField()
```

Przykładowe dane

Bazę wypełniłem przykładowymi danymi (screeny poniżej):

User

	{_id	Name	Password
1	60aedce18cae866b3e82cdbf	NNIKTT	1aafd38a801a68214816f41afe93bf26
2	60af0773d851e3253e08d272	tomek	827ccb0eea8a706c4c34a16891f84e7b
3	60af2ea971a66a1e3d4b2ede	kuba	9eb7e91737c94dfa9500938d04b71bc0
4	60af2eb371a66a1e3d4b2edf	pawel	cc0eff88d75fb735706fdd20b8b43814
5	60af2eb971a66a1e3d4b2ee0	kamil314	832b094432677a13aa800cf0c685a032

Battle

	{_id	Date	MapName	UsersInvolved	UsersResults	Winner
1	60af0aa0...	2021-05-27T00:...	Malinowka	[new ObjectId("60ae...	[new ObjectId("60af...	60af0773d851...
2	60af3920...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2eb371a6...
3	60af3935...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af0773d851...
4	60af393d...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2eb971a6...
5	60af3943...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2ea971a6...
6	60af3949...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2eb971a6...
7	60af394d...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2eb371a6...
8	60af3bb5...	2021-05-27T00:...	Malinowka	[new ObjectId("60ae...	[new ObjectId("60af...	60af2eb971a6...
9	60af3bb9...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af0773d851...
10	60af3bc5...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2ea971a6...
11	60af3bc9...	2021-05-27T00:...	Malinowka	[new ObjectId("60ae...	[new ObjectId("60af...	60af2eb971a6...
12	60af3bd0...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60aedce18cae...
13	60af3bd6...	2021-05-27T00:...	Malinowka	[new ObjectId("60af...	[new ObjectId("60af...	60af2eb371a6...

Map


```

        j = []
        for his in history:
            if user in his.UsersInvolved:
                j.append(his)
        res = []
        for bat in j:
            winner_id = bat.Winner.id
            winner_nick = User
            .objects(id=winner_id)
            .first()
            .Name
        res.append({
            "Date": bat.Date,
            "Map Name": bat.MapName,
            "Winner": winner_nick
        })
    return res

```

get_nick

Metoda zwraca nick użytkownika o danym id:

```

def get_nick(my_id):
    return str(User.objects(id=my_id).first().Name)

```

add_battle

Metoda dodaje wynik bitwy do bazy danych (dodaje bitwę do Battle oraz wyniki graczy do UserInBattle):

```

def add_battle(map_name, player1result, player2result):
    pl1res = UserInBattle()
    pl2res = UserInBattle()
    pl1res.UserId = User
    .objects(id=player1result["user_id"])
    .first()
    pl1res.NumOfShots = player1result["num_of_shots"]
    pl1res.NumOfShotsOnTarget =
    player1result["num_of_shots_on_target"]

```

```

        pl1res.GivenDamage = player1result["given_damage"]
        pl1res.GetDamage = player1result["my_damage"]
        pl1res.OpponentHp = player1result["opponent_hp"]
    pl2res.UserId = User
    .objects(id=player2result["user_id"])
    .first()
    pl2res.NumOfShots = player2result["num_of_shots"]
    pl2res.NumOfShotsOnTarget =
    player2result["num_of_shots_on_target"]
    pl2res.GivenDamage = player2result["given_damage"]
    pl2res.GetDamage = player2result["my_damage"]
    pl2res.OpponentHp = player2result["opponent_hp"]

    pl1res.save()
    pl2res.save()

    pl1_id = User
    .objects(id=player1result["user_id"])
    .first()
    pl2_id = User
    .objects(id=player2result["user_id"])
    .first()

    winner = None
    if pl1res.GivenDamage == pl1res.OpponentHp:
        winner = pl1_id
    else:
        winner = pl2_id
    bat = Battle()
    bat.MapName = map_name
    bat.Date = datetime.utcnow()
    bat.Winner = winner
    bat.UsersResults = [pl1res, pl2res]
    bat.UsersInvolved = [pl1_id, pl2_id]
    bat.save()

```

my_battle_history

Metoda zwraca listę obiektów reprezentujących rozegrane bitwy dla danego gracza:

```
def my_battle_history(user_id):
    history = Battle.objects()
    user = User.objects(id=user_id).first()
    j = []
    for his in history:
        if user in his.UsersInvolved:
            j.append(his)
    res = []
    for bat in j:
        winner_id = bat.Winner.id
        winner_nick = User
        .objects(id=winner_id)
        .first()
        .Name
        res.append({
            "Date": bat.Date,
            "Map Name": bat.MapName,
            "Winner": winner_nick
        })
    return res
```

get_stat

Metoda zwraca wyliczone statystyki dla konkretnego użytkownika:

```
def get_stat(user_id):
    data = UserInBattle.objects(UserId=user_id)
    history = Battle.objects()
    user = User.objects(id=user_id).first()
    battles = []
    for his in history:
        if user in his.UsersInvolved:
            battles.append(his)
    wins = 0
    wins_effectiveness = 0
    all_battles = len(battles)
    for bat in battles:
        if bat.Winner.id == user_id:
```

```

        wins += 1
    if all_battles > 0:
        wins_effectiveness = wins / all_battles
        accuracy = 0
        shots_in_target = 0
        shots = 0
        shots_per_battle = 0
        num_of_battles = len(data)
        all_opponent_hp = 0
        all_given_damage = 0
        moderate_damage = 0
        for bat in data:
            shots_in_target += bat.NumOfShotsOnTarget
            shots += bat.NumOfShots
            all_opponent_hp += bat.OpponentHp
            all_given_damage += bat.GivenDamage
        if shots > 0:
            accuracy = shots_in_target / shots
            if num_of_battles > 0:
                shots_per_battle = shots / num_of_battles
            if all_opponent_hp > 0:
                moderate_damage = all_given_damage / all_opponent_hp
        return {
            "nick": User.objects(id=user_id).first().Name,
            "accuracy": accuracy,
            "shots_per_battle": shots_per_battle,
            "wins_effectiveness": wins_effectiveness,
            "all_battles": all_battles,
            "moderate_damage": moderate_damage
        }

```

login

Metoda pozwala na zalogowanie się użytkownika i zwraca jego id:

```

def login(nick, password):
    if nick and password:
        my_user_info = User
        .objects(Name=nick, Password=code(password))
        if my_user_info:

```

```

        res = my_user_info[0]
        return res.id
    return None

```

sign

Metoda pozwala na dodanie nowego użytkownika do bazy, zwraca False gdy użytkownik o podanym nicku lub hasle istnieje już:

```

def sign(nick, password):
    l1 = User.objects(Name=nick)
    l2 = User.objects>Password=code(password))
    if l1 or l2:
        return False
    User(Name=nick, Password=code(password)).save()
    return True

```

change_password

Metoda pozwala na zmianę nicku lub hasła w momencie gdy argumenty starego nicku oraz starego hasła są poprawne:

```

def change_password(old_nick,
                    old_password,
                    new_nick=None,
                    new_password=None):

    if (not old_nick) or (not old_password):
        return False
    log = login(old_nick, old_password)
    if not log:
        return False
    if (not new_nick) and (not new_password):
        return False
    if (not new_nick) and new_password:
        new_nick = old_nick
    if (not new_password) and new_nick:

```

```

        new_password = old_password
    User
    .objects(Name=old_nick)
    .first()
    .update(Name=new_nick, Password=code(new_password))
    return True

```

get_new_battle

Metoda dostarcza rozpoczętej nowej rozgrywce informacje na temat pozycji początkowych 2 graczy oraz mapy na której odbędzie się walka:

```

def get_new_battle():
    map_list = Map.objects()
    if map_list:
        chosen_map = random.choice(map_list)
        return {
            "MapName": chosen_map.MapName,
            "SrcPath": chosen_map.SrcPath,
            "Pos1": chosen_map.InitPos1["coordinates"],
            "Pos2": chosen_map.InitPos2["coordinates"],
        }
    return False

```

Metoda dostępna dla administratora, która pozwala na dodanie nowej mapy:

```

def add_map(src_path, name, init_pos1, init_pos2):
    if Map.objects(MapName=name):
        return False

```

```

mp = Map()
mp.MapName = name
mp.SrcPath = src_path
mp.InitPos1 = [init_pos1[0], init_pos1[1]]
mp.InitPos2 = [init_pos2[0], init_pos2[1]]
mp.save()

```

Metoda dostępna dla administratora, która pozwala na usunięcie mapy:

```

def delete_map(self, map_name):
    Map.objects(MapName=map_name).first().delete()

```

Metoda która posłużyła do wypełnienia bazy przykładowymi danymi odnośnie stoczonych walk:

```

def i_want_to_play():
    # get init map
    map_name = get_new_battle()["MapName"]
    # get random object
    user1id = random.choice(User.objects()).id
    user2id = random.choice(User.objects()).id
    while user1id == user2id:
        user2id = random.choice(User.objects()).id
    num_of_shots = int(random.uniform(1, 10))
    num_of_shots_on_target = int(random.uniform(1, 5))
    damage1 = int(random.uniform(1, 1000))
    player_result = {
        "user_id": user1id,
        "num_of_shots": num_of_shots,
        "num_of_shots_on_target": num_of_shots_on_target,
        "given_damage": damage1,
        "my_damage": 1000,
        "opponent_hp": 1000
    }
    num_of_shots = int(random.uniform(1, 10))
    player2_result = {
        "user_id": user2id,
        "num_of_shots": num_of_shots,
        "num_of_shots_on_target": num_of_shots_on_target,
        "given_damage": 1000,
        "my_damage": damage1,
        "opponent_hp": 1000
    }

```

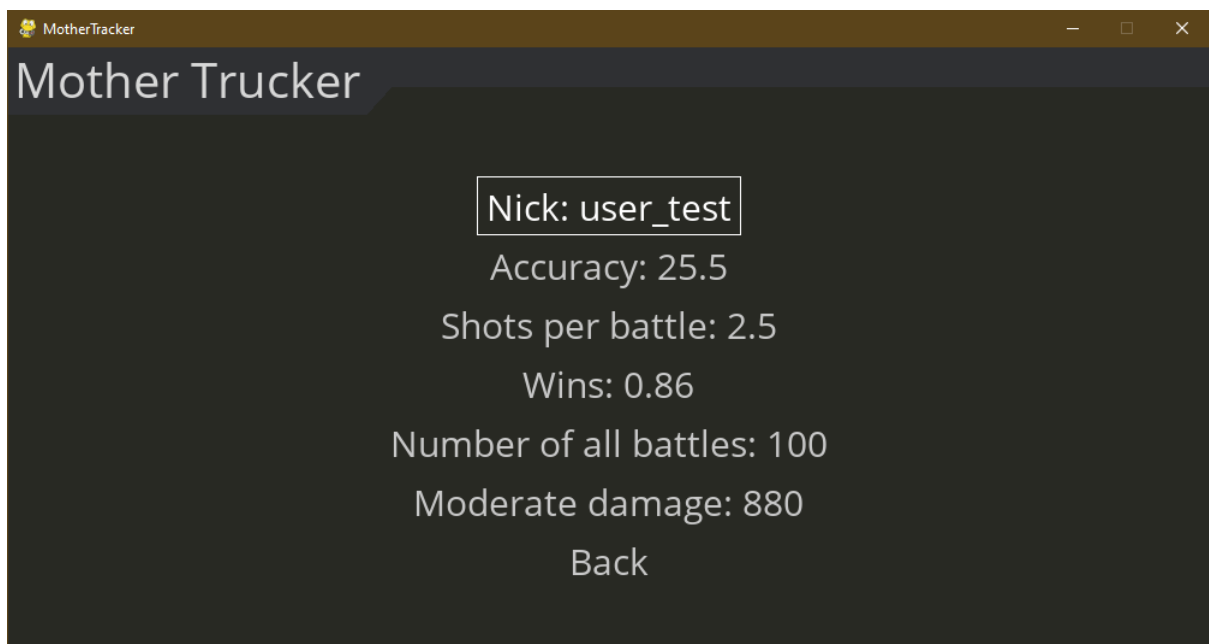
```
add_battle(map_name, player_result, player2_result)
```

Serwer komunikuje się za pomocą powyżej przedstawionych metod z baza danych, co pozwala na przesyłanie tych informacji do użytkowników.

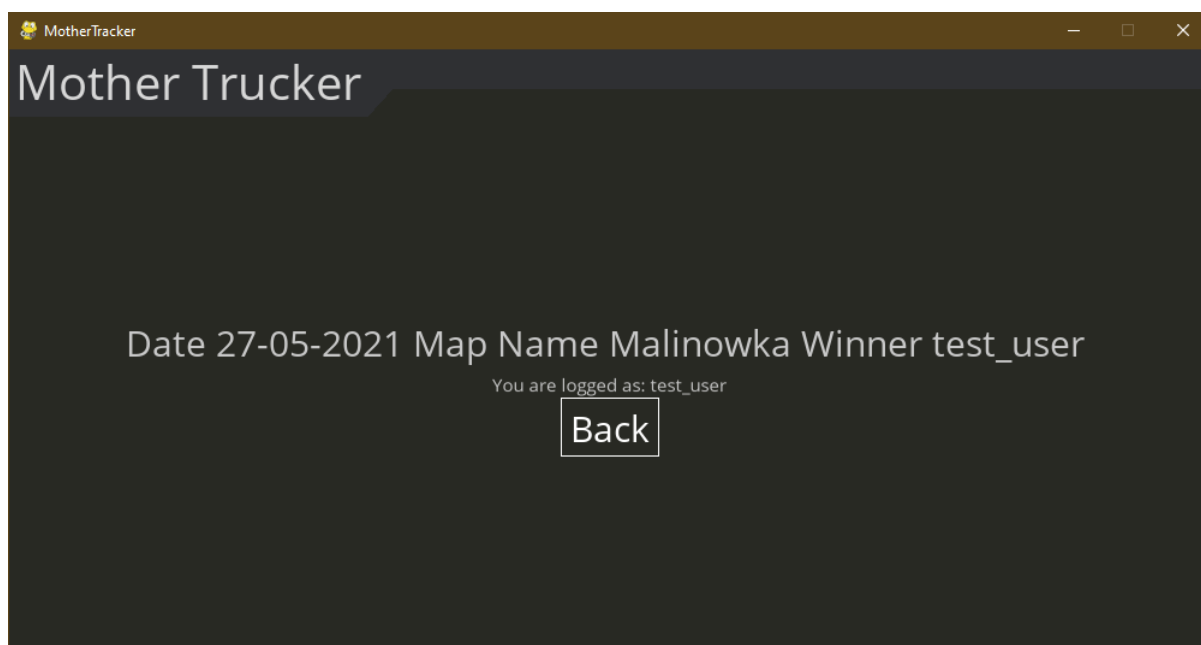
Widoki

Po stronie klienta mamy możliwość wyświetlenia naszych statystyk oraz historii bitew jak również możemy zmodyfikować nasze hasło i nick:

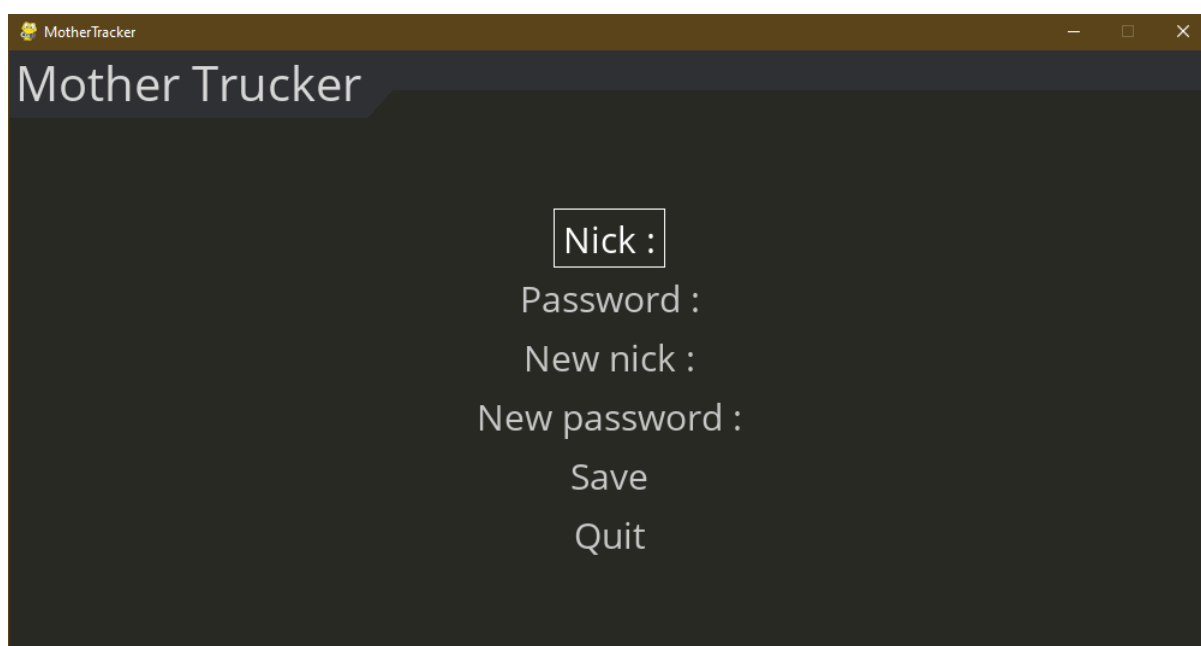
Statystyki



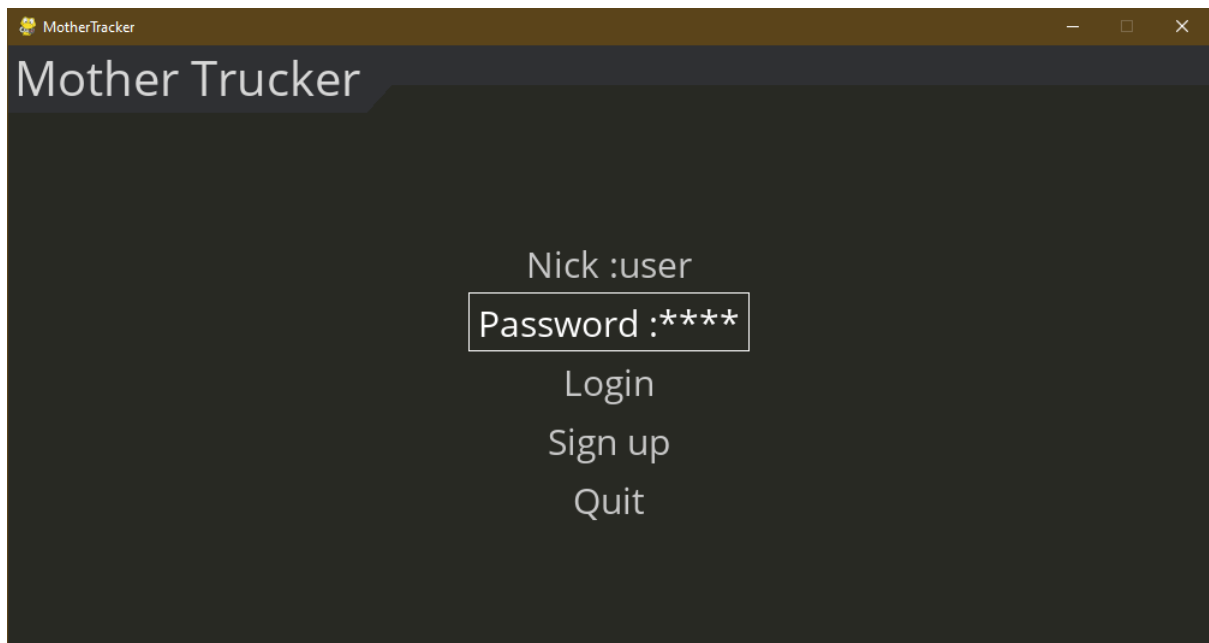
Historia bitew



Zmiana hasła lub nicku



Logowanie



Kolejną funkcjonalnością jest tworzenie dokumentów w bazie na podstawie stoczonej walki. Gdy dwóch graczy wybierze 'Play' i rozgrywka się rozpocznie, zostaje wybrana mapa oraz pozycje startowe pojazdów. Po zakończeniu bitwy ta zostaje utrwalona w bazie.

W ogólności w projekcie za komunikację z bazą danych i kierowanie zapytan odpowiada skrypt 'DBManager.py' w katalogu serwera. Istnieje również mock bazy 'MockDB.py' który na etap testów zwracał przykładowe dane nie istniejące faktycznie w bazie. Projekt w pliku 'settings.py' posiada zmienną DATA_BASE która determinuje czy korzystamy z faktycznej bazy czy z Mocka w projekcie.

W najbliższym czasie powstanie osobne GUI dla administratora aby ten mógł zarządzać danymi. Na githubie w katalogu DataBase znajdują się pliki JSON z obecnym losowo wygenerowanym stanem.