Projekt z Programowania w języku Python oraz Baz Danych

Kamil Pluciński, Tomasz Koszarek

Cel projektu

Celem projektu jest stworzenie dwuwymiarowej gry o walce czołgami w widoku z lotu ptaka.
Gra będzie stanowić pełne oprogramowanie pozwalające na rozgrywkę w sieci lokalnej, gdzie serwer będzie na komputerze z systemem operacyjnym Linux z zainstalowanym mongoDB.

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 możliwość
   - 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 siebie (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
    - biblioteka pygame, pygame_menu, threading
    - 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

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.

Opis architektury kodu wraz z wymienionymi najważniejszymi fragmentami, oraz ich akcje.

Logika gry

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

Przykładowe systemy wraz z krótkim opisem odpowiedzialności:

CollisionSystem
-wykrywanie kolizji SAT
-statyczne rozwiązywanie kolizji

IntegrationSystem:
-masa pojazdu wpływa na to
-zmiana pozycji wypadkowa -> ruch
-wyzerowanie wypadkowej

ControlSystem:
-wczytanie klawiszy
-wygenerowanie sił
-rotacja obiektów
-ustawia flagę is_dirty

RestianecesSystem:
-wyliczenie oporów (opór gruntu)
-aktualizacja wypadkowej

HitboxSystem:
-aktualizuje stransformowane wierzchołki w oparciu o flagę
-is_dirty ustawia na false

PhysicsSystem:
-collision system
-control system
-restiances system
-integration system

Serwer

Logika gry wykonywana jest na serwerze bądź u klienta w zależności od trybu rozgrywki (multi, single). W przypadku serwera obsługiwanych jest dwóch klientów na osobnych wątkach (nałożona blokada na sockety), oprócz trybu rozgrywki gdzie gracze współdzielą wątek z logiką gry (odblokowane sockety). Przy pomocy biblioteki socket zapewniamy komunikacje w sieci lokalnej.

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

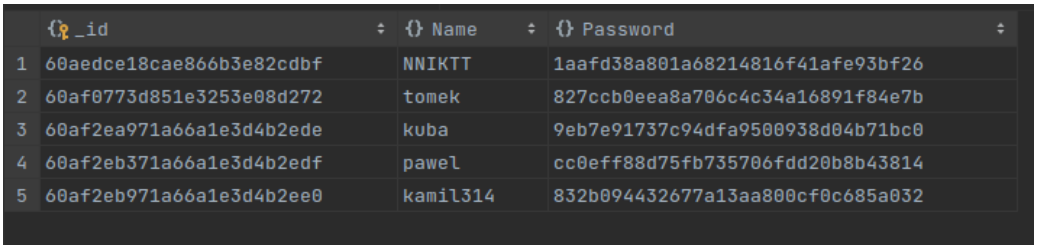
Kolekcje będą modelowane za pomocą klas dziedziczących z klasy Document z biblioteki mongoengine:

```
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

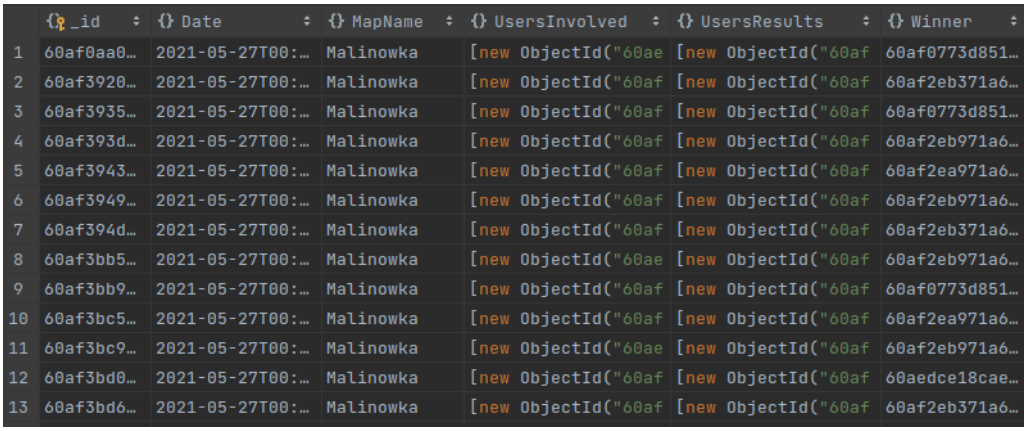
Bazę wypełniłem przykładowymi danymi (screeny poniżej):

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

filter {} sort {}

	_id	InitPos1	InitPos2	MapName	SrcPath
1	60aef1942…	{"type": "Point", "coordinates": [new NumberInt("0")	{"type": "Poin	Malinowka	some_ex_src

UserInBattle

	_id	GetDamage	GivenDamage	NumOfShots	NumOfShotsOnTarget	OpponentHp	UserId
1	60af0aa09…	1000	750	10	3	1000	60aedce18cae866b3e82cdbf
2	60af0aa09…	750	1000	4	3	1000	60af0773d851e3253e08d272
3	60af3920a…	1000	255	7	1	1000	60af2ea971a66a1e3d4b2ede
4	60af3920a…	255	1000	7	1	1000	60af2eb371a66a1e3d4b2edf
5	60af3935a…	1000	213	6	1	1000	60af2eb971a66a1e3d4b2ee0
6	60af3935a…	213	1000	4	1	1000	60af0773d851e3253e08d272
7	60af393da…	1000	104	3	3	1000	60af2ea971a66a1e3d4b2ede
8	60af393da…	104	1000	4	3	1000	60af2eb971a66a1e3d4b2ee0
9	60af3943a…	1000	735	7	1	1000	60af0773d851e3253e08d272
10	60af3943a…	735	1000	1	1	1000	60af2ea971a66a1e3d4b2ede
11	60af3949a…	1000	429	9	3	1000	60af2eb371a66a1e3d4b2edf
12	60af3949a…	429	1000	4	3	1000	60af2eb971a66a1e3d4b2ee0
13	60af394da…	1000	479	1	1	1000	60af2ea971a66a1e3d4b2ede
14	60af394da…	479	1000	6	1	1000	60af2eb371a66a1e3d4b2edf

Operacje na bazie danych

Metody operujące na bazie danych pozwalają serwerowi na:

my_battle_history

Metoda zwraca listę obiektów z informacjami o bitwach które się odbyły:

```
    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_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 haśle 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 hasla sa 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 = {
```

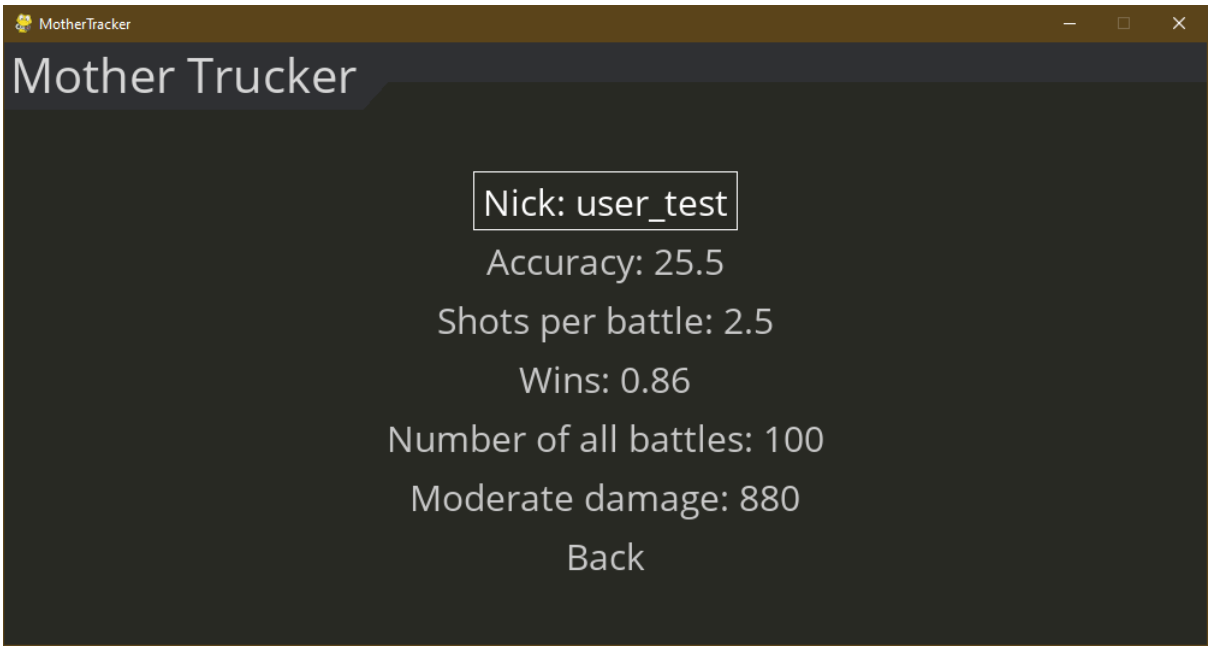
```
                "user_id": user2id,
                "num_of_shots": num_of_shots,
                "num_of_shots_on_target": num_of_shots_on_target,
                "given_damage": 1000,
                "my_damage": damage1,
                "opponent_hp": 1000
            }
            add_battle(map_name, player_result, player2_result)
```

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

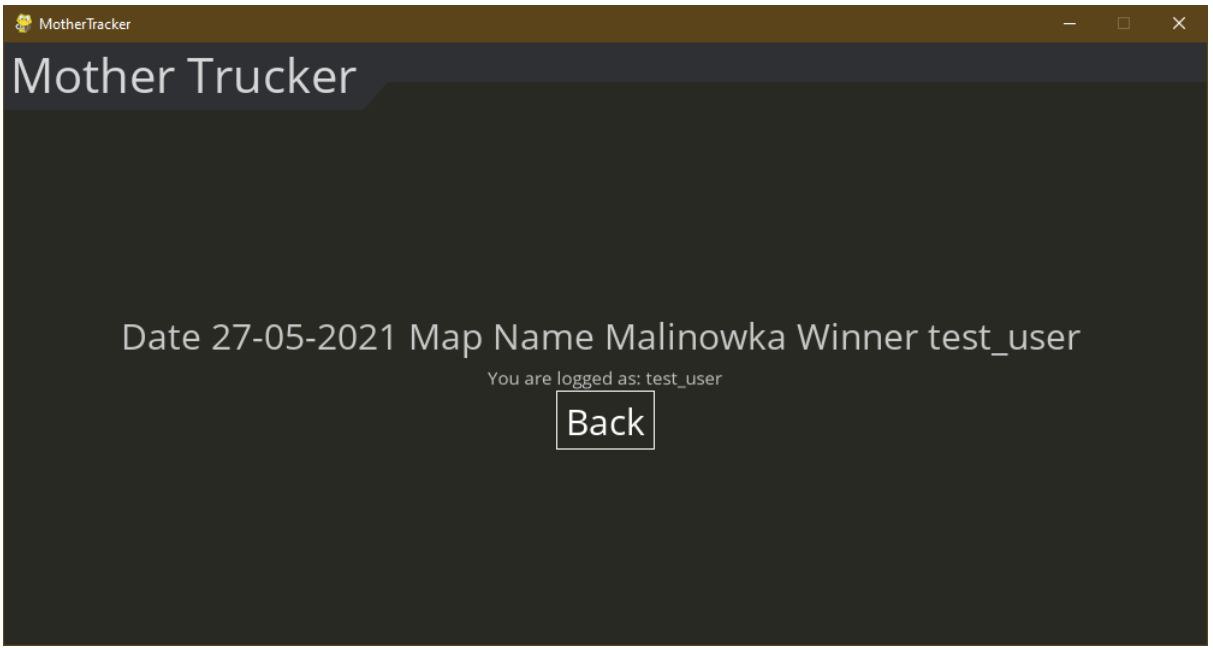
Widoki

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

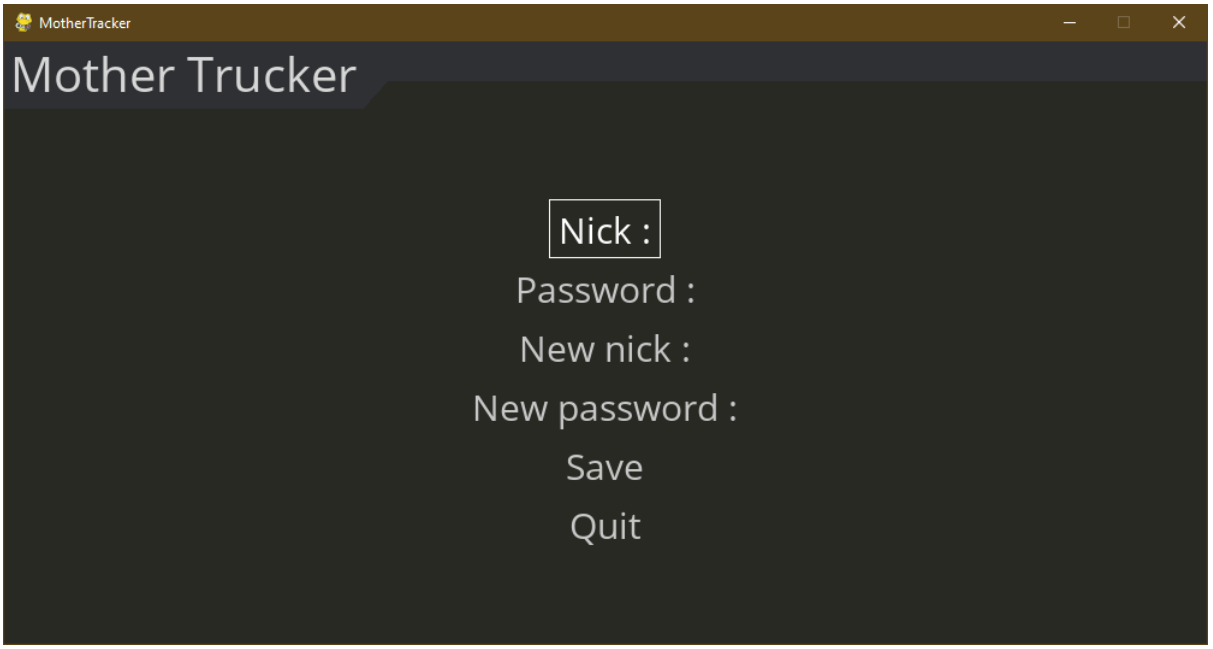
Statystyki



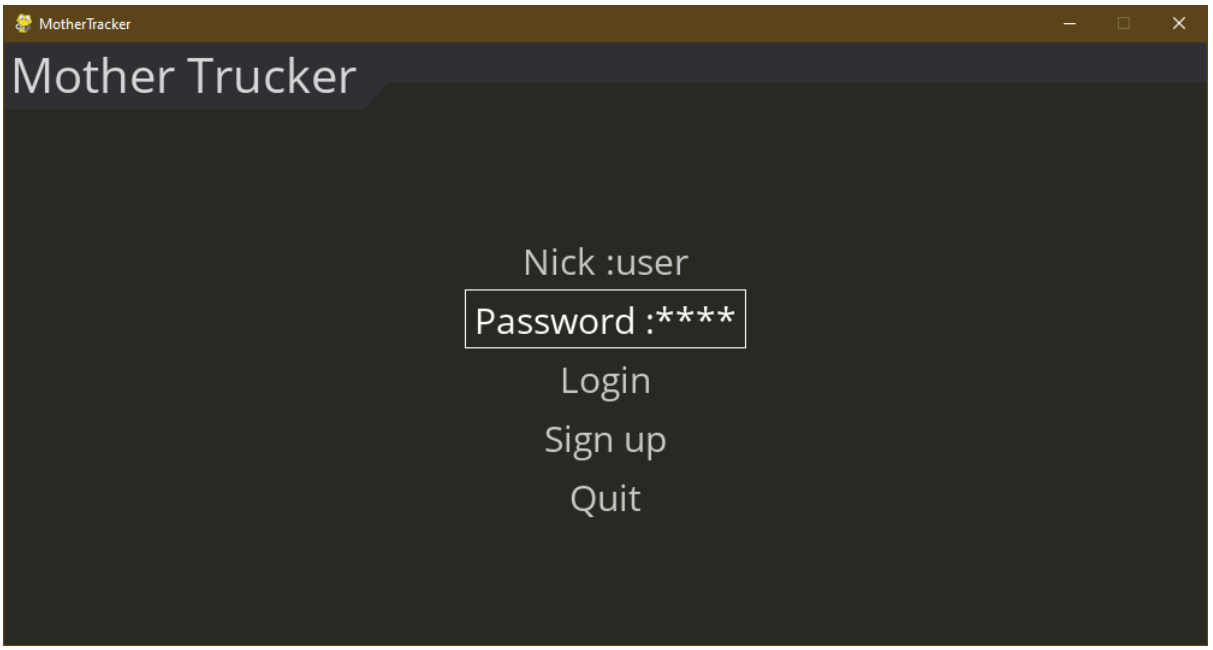
Historia bitew



Zmiana hasła lub nicku



Logowanie



Kolejna funkcjonalnoscia jest tworzenie dokumentow 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 ogolnosci w projekcie za komunikacje z baza 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 zmienna 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.