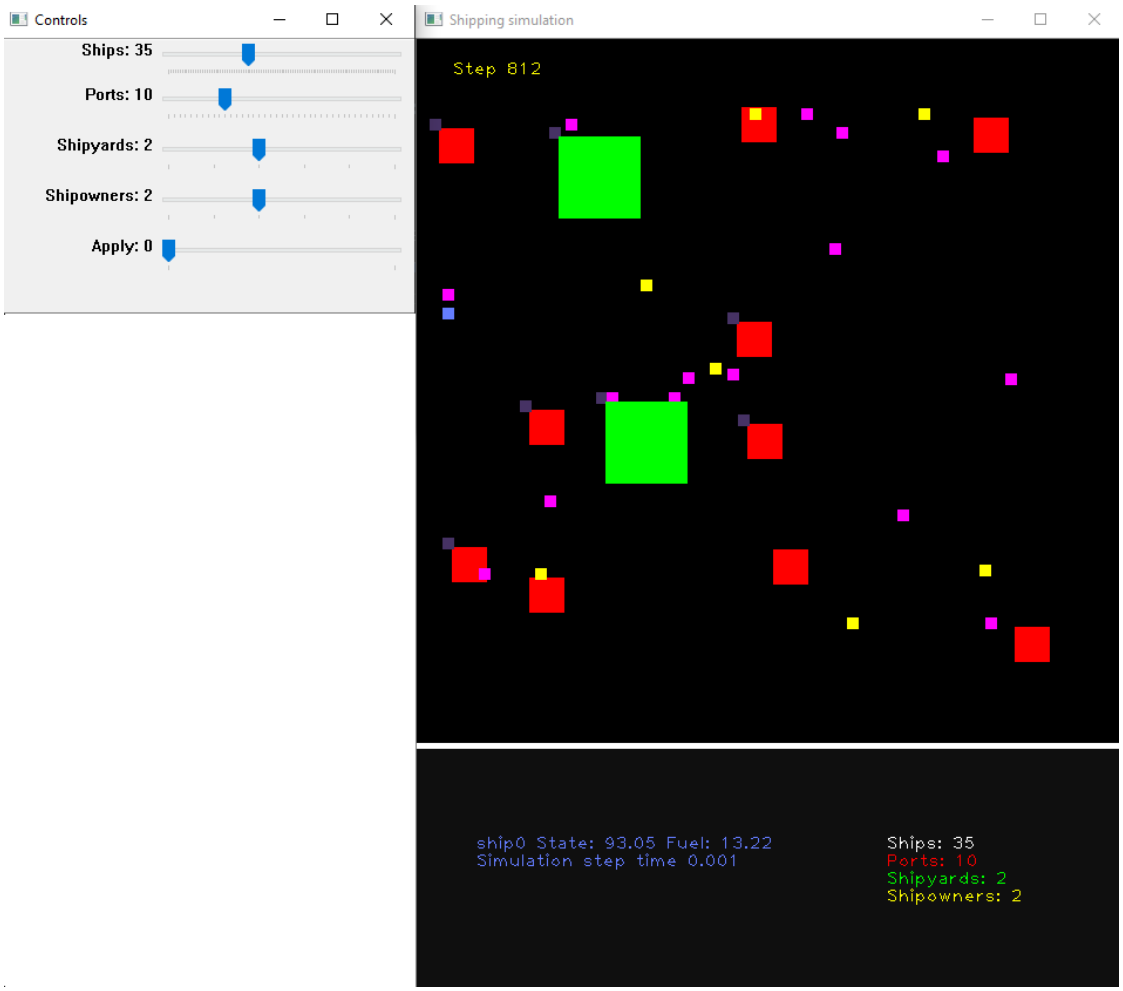
Okręty

Wstęp

Projekt przedstawia aplikację napisaną w języku *Python*. Symuluje ona zachowanie okrętów na morzu, które poruszają się pomiędzy portami i przewożą kontenery. Do wykonania prostego interfejsu graficznego wykorzystana została biblioteka *OpenCV*, do obliczeń na macierzach biblioteka *NumPy*. Wykorzystano również bibliotekę *time* do sterowania szybkością symulacji.

Interfejs graficzny

Korzystając z możliwości graficznych biblioteki *OpenCV* wyświetlane są dwa okna. Jedno do prezentacji pozycji obiektów symulacji, a drugie dające możliwość sterowania ilością statków, portów, stoczni i armatorów obecnych w symulacji.



W związku z brakiem możliwości implementacji przycisku w podanej bibliotece zaimplementowano go w postaci paska przesuwnego o nazwie *Apply*.

Sterowanie symulacją

Aby zmienić liczbę statków, portów, stoczni i armatorów należy za pomocą odpowiednich pasków przesuwnych ustawić właściwą liczbę obiektów. Następnie, aby wprowadzić zmiany i zainicjalizować symulację i mapę należy przesunąć suwak o nazwie *Apply*.

Do sterowania przebiegiem symulacji wykorzystano klawisze klawiatury. Poniżej opisano działanie każdego z nich:

- Klawisz ‘r’ – Restart symulacji
- Klawisz ‘k’ – Zmniejszenie czasu symulacji jednego kroku o 0.0005 s
- Klawisz ‘l’ – Zwiększenie czasu symulacji jednego kroku o 0.0005 s
- Klawisz ‘q’ – Zakończenie symulacji
- Klawisz ‘s’ – Zapisuje stan symulacji do pliku pickle
- Klawisz ‘x’ – Zapisuje stan symulacji do pliku o formacie csv
- Klawisz ‘g’ – Wczytuje stan symulacji z pliku pickle
- Klawisz ‘b’ – Wczytuje stan symulacji z pliku o formacie csv

Klasy wykorzystywane w symulacji

Każda klasa została zaimplementowana w osobnym pliku o rozszerzeniu .py, a następnie zaimportowana w skryptach, które korzystają z tej klasy.

Lista klas

1. Simulation
2. SeaMap
3. Ship
4. Port
5. Shipyard
6. Shipowner
7. SeaCan
8. Dock
9. Warehouse
10. FuelMagazine

Klasa Simulation

Odpowiada za kontrolę przebiegu symulacji.

Metody:

- **__init__()** – Inicjalizacja parametrów symulacji
- **run()** – Główna pętla, kontroluje jak długo trwa krok symulacji
- **next_step()** – Wywołuje metody odpowiedzialne za wykonanie obliczeń odnośnie stanu symulacji w następnym kroku
- **move_ships()** – Odpowiada za przemieszczanie się statków i nadawaniem im kolorów w zależności od stanu w jakim się znajdują
- **give_orders()** – Metoda odpowiedzialna za sterowaniem stanu statków i ich kolejnych docelowych portów w zależności od stanu w jakim się znajdują
- **show_map()** – Uaktualnia i wyświetla mapę, pobiera informacje podawane przez użytkownika

Klasa SeaMap

Przechowuje informacje o obiektach znajdujących się w symulacji: statkach, portach, stoczniach i armatorach, ich liczbie, pozycji itp.

Metody:

- **is_place_empty()** – Sprawdza czy podana pozycja jest zajęta przez inny obiekt
- **find_place()** – Znajduje wolne miejsce dla obiektów symulacji podczas ich inicjalizacji
- **init_map()** – Inicjalizacja mapy
- **clean_map()** – Resetuje informacje przechowywane przez mapę
- **setup_shipowners()** – Odpowiada za przyporządkowanie obiektów do właściwych armatorów

Klasa Ship

Implementacja obiektów jakimi są okręty, ich stanu, poziomu paliwa, docelowych portów, listy kontenerów do przewiezienia itp.

Metody:

- **__init__()** – Inicjalizacja właściwości okrętu
- **degrade()** – Funkcja symulująca pogarszanie się stanu okrętu podczas przewożenia kontenerów
- **use_fuel() -** Funkcja symulująca zużywanie paliwa przez okręt podczas przewożenia kontenerów
- **is_at_destiny_port()** – Zwraca informację czy statek dotarł już do miejsca docelowego dostarczenia kontenera
- **fill_up()** – Funkcja symulująca tankowanie okrętu
- **unload_seaCan()** – Funkcja symulująca rozładunek przewożonych kontenerów

Klasa Port

Odpowiada za implementacje obiektu jakim jest port morski i jego składowych czyli doków i magazynów załadunkowych i rozładunkowych do obsługi kontenerów.

Metody:

- **__init__()** – Inicjalizacja właściwości portu
- **create_docks()** – Inicjalizacja doków w ilości wygenerowanej losowo
- **create_warehouses()** – Inicjalizacja magazynów do załadunku i rozładunku magazynów
- **fill_up_ship()** – Metoda umożliwiająca tankowanie paliwa okrętom przybyłym do portu

Klasa Shipyard

Odpowiada za implementacje obiektu jakim jest port morski i jego składowych czyli doków i magazynów załadunkowych i rozładunkowych do obsługi kontenerów.

Metody:

- **__init__()** – Inicjalizacja właściwości stoczni
- **create_docks()** - Inicjalizacja doków w ilości wygenerowanej losowo
- **repair()** – Funkcja symulująca naprawę okrętu
- **has_empty_slots()** – Sprawdza czy istnieje wolny dok w stoczni
- **is_still_being_repaired()** – Sprawdza czy podany statek jest nadal naprawiany
- **fill_up_ship()** - Metoda umożliwiająca tankowanie paliwa okrętom przybyłym do stoczni
- **make_new_ship() –** Umożliwia konstruowanie nowych statków
- **destroy_ship() –** Umożliwia rozbiórkę okrętów umożliwiając odzysk środków właścicielowi danego okrętu

Klasa Shipowner

Udostępnia metody potrzebne do funkcjonowania klasy właściciela okrętów.

Metody:

- **__init__()** – Inicjalizacja stanu właściciela okrętów
- **schedule_seaCans()** – Metoda określająca porty docelowe transportu kontenerów
- **buy_ship()** – Metoda implementująca możliwość zakupu okrętu przez armatora
- **repair_ship()** – Umożliwia naprawę wybranego statku
- **fill_up_ship()** - Metoda umożliwiająca zlecenie zatankowania statku
- **destroy_ship() –** Metoda umożliwiająca zlecenie rozbiórki statku

Kolejki

Do przechowywania informacji o kolejnych portach docelowych używano w projekcie list dostępnych w języku *Python*. Aby dostać pierwszy element wykorzystywano metodę **list_name.pop(0**). Ostatni element otrzymywany był dzięki poleceniu **list_name.pop(-1)**.

```
class Ship:
    def __init__(self, name, capacity, shipowner, seaCans):
        self.position = [0, 0]
        self.seaCans = seaCans
        self.src_port_queue = []
        self.dest_port_queue = []
        self.shipowner = shipowner
        self.capacity = capacity
        self.fuel = 20
        self.state = 100
        self.name = name
```

Przykładowe wykorzystanie kolejki **dest_port_queue** i funkcji **pop()** w metodzie klasy **Ship** :

```
def unload_seaCan(self):
    self.is_unloading_seaCan = True
    current_port = self.dest_port_queue[0]
    current_port.loading_warehouse.load_seaCan(self.seaCans.pop(0))
```

Zapis i odczyt stanu

W celu zapisu aktualnego stanu symulacji i jego późniejszego odczytu, do klasy **Simulation** dodane zostały następujące funkcjonalności:

1. **save_state() –** tworzy nowy folder i zapisuje do niego plik pickle przechowujący informację o stanie symulacji
2. **load_state() –** wczytuje z pliku pickle informacje o zachowanym stanie symulacji
3. **save_state_in_csv() –** tworzy nowy folder i zapisuje do niego plik format csv przechowujący informację o stanie symulacji
4. **load_state_from_csv() –** wczytuje z pliku o formacie csv informacje o zachowanym stanie symulacji