Dokumentacja projektu zespołowego

Agents in the Harbour

Lilianna Hełminiak, Maciej Kaim, Wojciech Kuprianowicz

Spis treści

Krótki opis programu

Agents in the Harbour to program symulujący proces załadowania statku w porcie przez dźwigi, które mają ograniczony zasięg ramienia oraz przez mobilne wózki widłowe. Jest to program wykorzystujący systemy inteligentnych agentów, gdzie każdy wózek i dźwig to agent mający możliwość komunikowania się z innymi agentami.

Celem agentów jest załadowanie statku w jak najszybszy sposób wykonując jak najmniej ruchów.

Charakterystyka problemu

Wprowadzenie

System agentowy to system złożony z agentów. Agent ma jakiś cel i stara się w czasie swojego działania przybliżyć do niego. Agenci potrafią badać środowisko w którym się znajdują i podejmować decyzje oraz kroki, które przybliżą ich do osiągnięcia celu. Potrafią komunikować się i współpracować z innymi agentami, tak aby realizować wspólne cele.

Systemy agentowe nadają się do rozwiązywania problemów o charakterze rozproszonym lub złożonych obliczeniowo.

Sformułowanie problemu

Problemem do rozwiązania jest załadowanie statku paczkami które znajdują się w porcie. Statek wysyła swoje żądania do agentów, którzy są rozproszeni po porcie. Są to dźwigi, które mają ograniczony zasięg ramienia oraz wózki widłowe które szukają paczek na polach do którego dźwigi nie mają dostępu.

Agenci muszą wspólnie współpracować, aby zapełnić statek. Tylko niektóre dźwigi mają dostęp do statku i to właśnie im trzeba podać paczki. Należy wykonać przy tym jak najmniej ruchów oraz wykonać to w jak najkrótszym czasie.

Problem nadaje się doskonale do rozwiązania agentowego. Agenci muszą się rozglądać po swoim otoczeniu aby móc zbadać środowisko. Następnie mają do wyboru parę akcji które mogą wykonać: nasłuchiwanie wiadomości, przesunięcie paczek ze stosu lub podanie paczki dalej. Muszą zdecydować jaką akcję podjąć aby załadować statek jak najszybciej.

Ważną cechą agentów oprócz badania otoczenia jest komunikacja z innymi agentami. Dzięki rozwiązaniu tego problemu systemem agentowym, agenci mogą się komunikować między sobą i ustalać poprzez negocjacje kto ma przenieść daną paczkę lub który z wózków ma się rozejrzeć po niedostępnym dla dźwigów polu. Dzięki temu agenci planują wspólnie akcje jakie mają wykonać i nie wykonują przy tym zbędnych zadań, lecz tylko te które zostały im przydzielone przez wygranie negocjacji.

Zastosowania programu

Program może posłużyć jako symulacja załadowania statku w porcie. Jest on dobrym przykładem problemu w którym czerpie się korzyści z użycia systemu agentowego.

Środowisko programu

Uruchomienie programu

Aplikacja napisana jest w języku Python 2.x. Aby ją uruchomic należy mieć zainstalowanego Pythona 2.x (http://www.python.org/) oraz pygame (http://pygame.org).

Po zainstalowaniu należy uruchomić plik main.py, opcjonalnie z nazwą wybranej planszy jako parametrem. Jeśli żaden paramentr nie jest podany, to program uruchomi się z domyślną planszą (map/map1).

Dane wejściowe

Wszystkie konfiguracje są czytane z pliku, który został podany przy uruchomieniu main.py.

Plik zawiera następujące informacje:

- Rozmiar planszy: wysokość, szerokość
- Liczba dźwigów
- Parametry dla każdego dźwigu: id dźwigu, współrzędne (y, x) zasięg widzenia, zasięg ramienia
- Liczba wózków
- Parametry dla każdego wózka: id wózka, współrzędne (y, x)
- Liczba paczek
- Parametry dla każdej paczki: id paczki, waga, współrzędne (y, x)
- Parametry dla statku: współrzędne przodu statku, współrzędne tyłu statku, liczba paczek w wiadomości dla agentów, przedział czasu co jaki statek ma wysyłać wiadomości agentom
- Id paczek, które statek będzie potrzebował.

Poprawność danych w pliku nie jest sprawdzana w programie. Razem z programem załączone są trzy pliki z trzema różnymi planszami. Ponieważ są przetestowane, uzytkownik powinien wybrać jeden z tych plików do uruchomienia programu.

Komunikacja z użytkownikiem

Symulacja wyświetla się użytkownikowi w nowym oknie graficznym. Po prawej stronie wyświetlana jest lista paczek jakie potrzebuje statek. Gdy odpowiednia paczka zostanie przeniesiona na statek, id paczki zniknie z listy.

Do obsługiwania okna programu można uzywać następujących klawiszy:

Strzałki góra i dół – do przesunięcia widoku planszy w odpowiednią stronę,

Lewy przycisk myszy – wyświetla informacje o kliknięty polu. Jeśli okno z informacjami jest aktualnie wyświetlane, to kolejne kliknięcie wyłączy to okno. Informacje mogą być wyświetlane dla następujących obiektów:

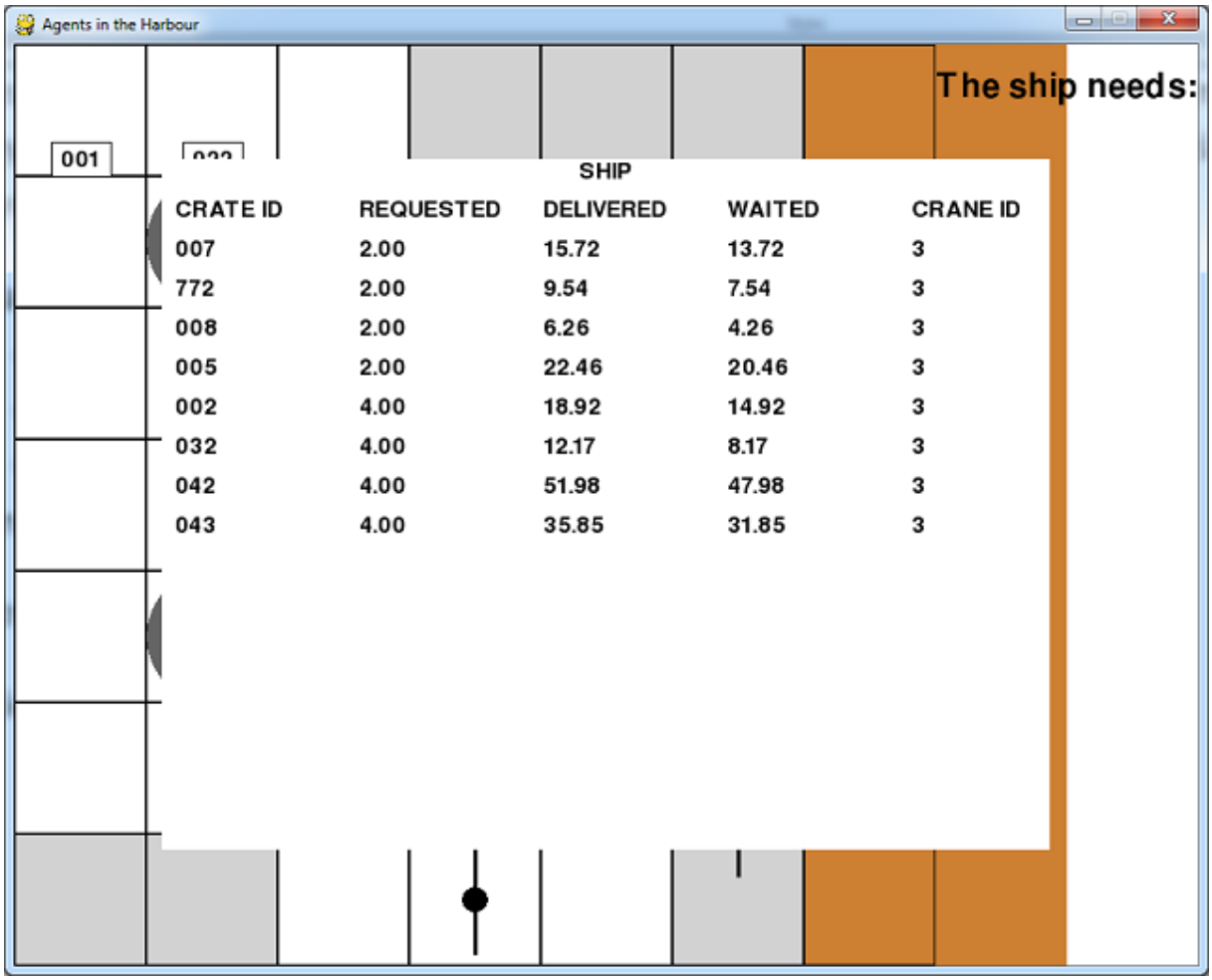
- Pola do magazynowania paczek:
    - storage fields – pola które są w zasięgu przynajmniej jednego dźwigu
    - road fields – pola które nie są dostępne przez żaden dźwig
    - ship fields – pola które należą do pokładu statku
- crane field – pole na którym stoi dźwig

Spacja – zatrzymaj/włącz symulację

ESC – wyjście z aplikacji.

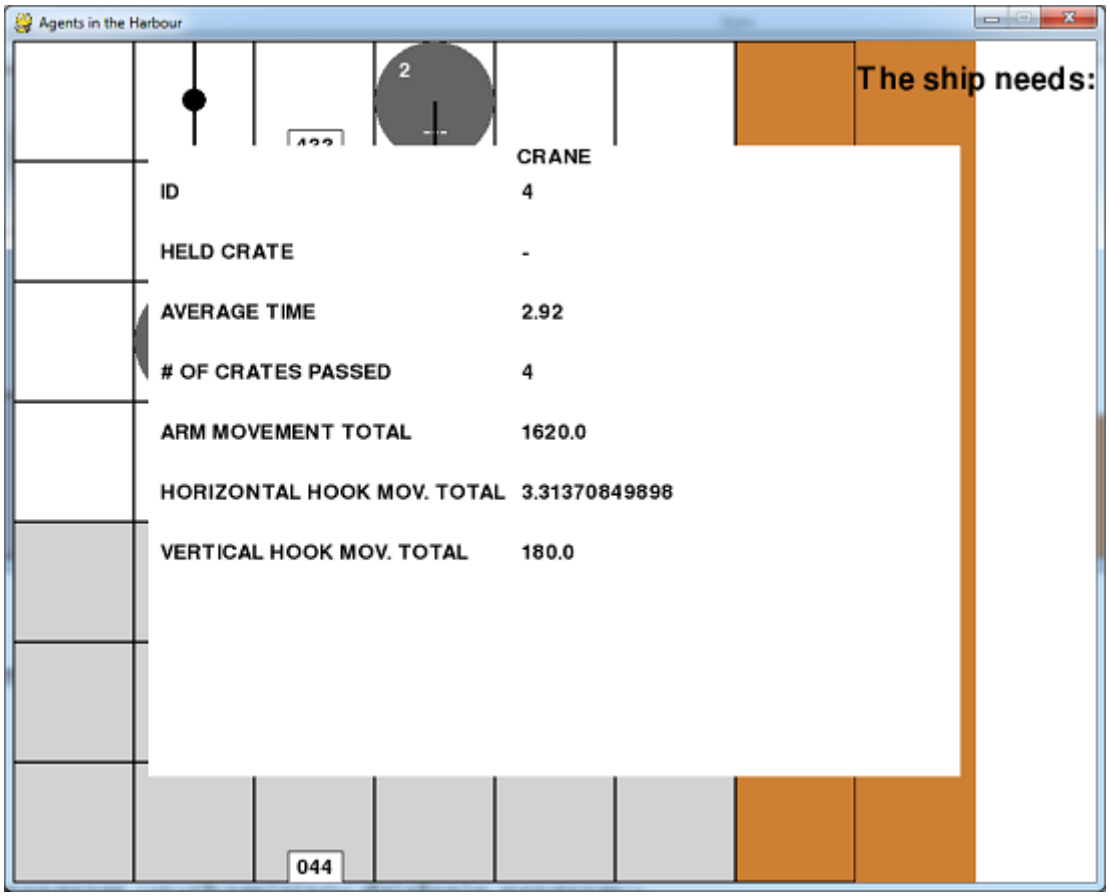
Wyniki

Po zakończeniu symulacji, gdy wszystkie statki są dostarczone na statek, można podejrzeć wyniki klikając lewym przyciskiem myszy na dowolne pole statku. Ukaże nam się lista paczek które były wymagane przez statek oraz czas kiedy zostały one zażądane oraz kiedy zostały dostarczone.



Rys1. Przykładowe wyniki dostarczenia paczek

Dodatkowo w każdym momencie programu możemy podejrzeć statystyki każdego z agentów. Dowiemy się ile paczek przeniósł, jaki jest jego średni czas dostarczenia oraz ile ruchów ramieniem wykonał.



Rys2. Przykładowe statystyki dla dźwigu

Moduły

Main

Główny program uruchamiający działanie programu.

Opis funkcjonalny modułu

Moduł jest przeznaczony do uruchomienia programu z właściwymi parametrami.
Zawiera jedynie konstruktor w którym tworzy obiekt map oraz wyświetla go.

Map

Moduł zawierający wszytkie konfiguracje programu.

Opis funkcjonalny modułu

Moduł jest przeznaczony do przechowywania wszystkich informacji na temat wyświetlanej mapy.

Map zawiera tablicę dwuwymiarową która symuluje port w 2D. Elementami tablicy są obiekty typu field.

Moduł zawiera następujące procedury:

stopThreads(): void – zatrzymuje wszystkie wątki aby program mógł wstrzymać działanie

inMapBounds(pos): bool – sprawdza, czy jakieś pole jest w obszarze mapy

distance(coordinates1, coordinates2): int – wylicza odległość między dwoma współrzędnymi.

commonArea(crane1, crane2): ((int y, int x), height, width) – wylicza wspólne pole o kształcie prostokąta dla dźwigów crane1 i crane2. Zwraca trójkę: współrzędne lewej górnej kratki, wysokość oraz szerokość wspólnego pola.

commonStorageFields(crane1, crane2): [(int y, int x)] – zwraca listę współrzędnych wspólnych pól dźwigów crane1 i crane 2 które są typu Storage, posortowaną po odległości od crane1.

drawMap() : void – wyświetla mapę

Map korzysta z innych modułów: crane, ship, forklift, display oraz field. Jest to potrzebne, ponieważ w tym module są tworzeni agenci.

Sytuacje niepoprawne

W poszczególnych metodach jest sprawdzane, czy dane wejściowe są poprawne. Jeśli tak nie jest, to jest zwracany obiekt None zamiast wyliczonej odpowiedzi.

Field

Moduł który określa typ danego pola na mapie.

Opis funkcjonalny modułu

Moduł jest przeznaczony do przechowywania informacji o danym polu na mapie, czyli jakiego jest typu oraz jakie obiekty się na nim aktualnie znajdują.

Może być następujących typów:
STORAGE_TYPE – na polu takiego typu można układać paczki,
ROAD_TYPE – po takiego typu polach mogą się przemieszczać wózki widłowe,
CRANE_TYPE – na polach tego typu znajdują się dźwigi,
SHIP_TYPE – na tych polach znajdują się pola należące do statku

Field zawiera listę, na której są przechowywane obiekty które leżą na danym polu.

Zawiera następujące procedury:

isStackable(): bool – sprawdza, czy można na danym polu położyć paczkę

countCrates(): int – zwraca liczbę obiektów mieszczących się na danym polu

getAllCratesIds(): [int] – zwraca id wszystkich paczek mieszczących się na polu

isCratePresent(crateId): bool – sprawdza, czy dana paczka znajduje się na liście obiektów pola

getCratePosition(crateId): int – jeśli paczka jest obecna na liści obiektów danego pola to jest zwracana jej pozycja

getTopCrateId(): int – zwraca id paczki leżącej na szczycie

removeCrateFromTop(): crate – ściąga ostatnią paczkę na liście obiektów pola i ją zwraca.

putCrateOnTop(crate): void – dołącza do listy obiektów na danym polu kolejną paczkę.

getCrane(): Crane – zwraca obiekt Crane który jest na danym polu.

isForkliftPresent(): bool – sprawdza, czy zawiera obiekt Forklift.

Moduł korzysta z modułu Crate w celu korzystania z informacji o id paczki jaka znajduje się na polu.

Sytuacje niepoprawne

W poszczególnych procedurach modułu, przed wykonaniem akcji jest sprawdzane jakiego typu jest pole. Jeśli typ nie zgadza się z daną akcją, to wywoływany jest wyjątek. Np. nie można zdobyć informacji o paczce, która miałaby leżeć na polu CRANE_TYPE.

Crate

Moduł który zawiera specyfikacje paczki mieszczącej się na mapie.

Opis funkcjonalny modułu

W module Crate znajduje się klasa tworząca obiekt paczki, który zawiera informacje o id paczki oraz o jej wadze.

Crane

Klasa która specyfikuje obiekt dźwigu.

Opis funkcjonalny modułu

Klasa jest przeznaczona do tworzania agentów stacjonarnych – dźwigów.

Każdy dźwig ma następujące pola:

Id: int – id różniące się dla każdego dźwigu

Position: (int y, int x) – pozycja dźwigu na mapie

Reach: int – zasięg dźwigu

Angle: float – kąt ramienia

hookDistance: float – odległość haku na ramieniu dźwigu

neighbours: [Crane] – lista dźwigów, z którym obiekt ma wspólne pole, a więc sąsiedzi dźwigu

map: Map – obiekt pomagający odczytać zawartość pól z okolicy dźwigu

crate: Crate – paczka która jest aktualnie przenoszona przez dźwig

averageTime: float – średni czas dla dźwigu jaki zajmuje mu dostarczenie paczki na statek

passedPackages: int – liczba paczek, które zostały podane przez dźwig

messages: Queue – kolejka do odczytywania wiadomości

negotiate: Queue – kolejka na odpowiedzi dotyczących negocjacji

tasks: [] – lista z zadaniami do wykonania przez dźwig (np. podnieć, upuść, obruć się...)

directToShip: bool – wartość boolowska mówiąca, czy dźwig jest bezpośrednio przy statku

hops – liczba przeskoków do statku, czyli liczba dźwigów między obiektem a statkiem na ścieżce do statku

wanted: [int] – lista id potrzebnych paczek przez statek

onMyArea: [int] – lista paczek na polach w zasięgu dźwigu

negotiations: bool – wartość, która mówi czy dźwig jest w trakcie negocjacji

Dźwig zawiera następujące metody:

mainLoop(): void – główna pętla obiektu w której jest zapisane jego działanie

doWork(): void – procedura w której są analizowane kolejne kroki jakie dźwig musi podjąć aby przemieścić paczkę

examineSurroundings(): void – dźwig sprawdza dzięki tej metodzie swoje otoczenie, tzn czy jest bezpośrednio przy statku oraz jakie paczki leżą w jego zasięgu

getPackageToDeliver(): void – metoda w której dźwig ustala kto ma się zająć daną paczką która jest w jego zasięgu i przenosi ją jeśli wygra negocjacje

readMessage() : void – odczytuje wiadomość z kolejki messages i podejmuje właściwe kroki w odpowiedzi na tą wiadomość

informOthers(): void – informuje sąsiadów o tym, że zna ścieżkę do statku

keepBusy(): void – w czasie kiedy nie ma więcej zadań do wykonania, dźwig zajmuje się samym sobą nie przeszkadzając innym

takeOff(pos): Crate – dźwig zabiera ze stosu paczkę z podanego pola

passOn(Crane): Crate – podaje wybranemu dźwigowi paczkę, którą aktualnie trzyma

loadShip(): Crate – jeśli dźwig znajduje się bezpośrednio przy statku to może załadować statek paczką którą aktualnie trzyma

Crane korzysta również z innych modułów: Message to przesyłania wiadomości oraz Field do sprawdzania typu pól i ich zawartości.

Charakterystyka działania modułu

Działanie agenta jest spisane w metodzie mainloop. Polega na trzech etapach:

1. Sprawdzeniu swojej okolicy. W celu dowiedzenia się jakich agent ma sąsiadów oraz jakie paczki leżą w jego polu zasięgu, dźwig przeszukuje swoje dostępne pola oraz dowiaduje się o jego otoczeniu.
2. Czytaniu wiadomości. Agent dowiaduje się od statku jakich paczek on potrzebuje poprzez odczytanie wiadomości z kolejki. Poprzez czytanie wiadomości, agent może się dowiedzieć o propozycjach jakie inni agenci mają do zaoferowania.

3. Wykonaniu pracy. Na podstawie informacji o swoim otoczeniu oraz wiadomości jakie agent dostał w potrzebnym kroku, agent może zacząć swoją pracę – ustalanie oraz przenoszenie odpowiednich paczek w kierunku statku.

Sytuacje niepoprawne

W celu uniknięcia sytuacji, gdzie dwa dźwigi chcą zabrać się jednocześnie za jakieś pole, przed rozpoczęciem ruchu dźwig sprawdza, czy dane pole nie jest zablokowane semaforem. Jeśli nie, to blokuje takie pole aby żaden inny dźwig nie przeszkadzał mu w pracy i zwalnia dopiero kiedy skończył rozładowywać stos paczek i uchwycił paczkę którą był zainteresowany. Jeśli pole jednak było zablokowane, to dźwig przechodzi do następnego zadania i wraca do tego pola później.

Forklift

Klasa która specyfikuje obiekt wózka widłowego.

Opis funkcjonalny modułu

Klasa jest przeznaczona do tworzania agentów stacjonarnych – wózków widłowych. Są to mobilni agenci, którzy poruszają się po polach gdzie dźwigi nie mają zasięgu. Gdy zauważą potrzebną paczkę, to przekazują ją dźwigowi.

Zawiera następujące pola:

Position: (int y, intx) – aktualna pozycja wózka na mapie

Angle: float – kąt ramienia wózka

Crate: Crate – obiekt paczki którą wózek aktualnie wiezie

Messages: Queue – kolejka wiadomości przychodzących.

Way: [(int y, int x)] - lista kolejnych współrzędnych na których ma poruszać się wózek

Wózek zawiera następujące metody:

mainLoop(): void – główna pętla działania obiektu. Zawiera pętlę z instrukcjami do działania.

continueWay(): void – przemieszcza się do następnego pola z listy Way

readMessages() : void – odczytuje wiadomości z kolejki

drop(): void – wózek upuszcza paczkę na pole które znajduje się tuż przed nim

grab(): void – wózek podnosi paczkę z pola które znajduje się tuż przed nim

turnRight: void – metoda która porusza wózek w prawo

turnLeft(): void – metoda która porusza wózek w lewo

forward(): void – metoda która porusza wózek do przodu

Charakterystyka działania modułu

Główne działanie wózka jest opisane w pętli mainLoop. Jest ono następujące:

1. Wózek nasłuchuje wiadomości o potrzebnych paczkach.
2. Rozgląda się po polu do którego został przydzielony. Szuka za potrzebnymi paczkami zgłoszonymi przez statek.
3. Gdy paczka zostanie znaleziona, wózek negocjuje z najbliższymi dźwigami o tym komu powinien przekazać paczkę.
4. Wózek komunikuje się z innymi wózkami dopytując się o ich pozycję i negocjując kto powinien podjechać na pola które nie są w zasięgu żadnego dźwigu i na których nie ma żadnego wózka.
5. Jeśli wózek wygra negocjacje to przemieszcza się do takiego pola. Algorytm przechodzi w pętli do punktu 1.

Sytuacje niepoprawne

W celu uniknięcia wjechania na niedostępne pole (na którym jest paczka lub na którym znajduje sie dźwig), każde pole jest sprawdzane przed przemieszczeniem się. Jeśli jest to pole typu Crane, to wózek nie może tam wjechać, jeśli jest typu Storage to musi być puste (lub z ewentualnie jedną paczką, którą wózek podniesie) oraz zablokowane przez wózek semaforem, tak aby nikt nie przeszkadzał wózkowi w trasie.

Ship

Klasa która specyfikuje obiekt statku.

Opis funkcjonalny modułu

Klasa tworzy obiekt statku, który komunikuje się z pozostałymi agentami w czasie trwania programu.

Statek zawiera następujące pola:

cratesPerMessage: int – liczba paczek które będą przekazywane agentom w kolejnych turach informowania o potrzebnych paczkach

timeInterval: int – przedział czasu co jaki statek powinien wysyłać kolejne zapytania o paczki

Crates: [int] – lista id paczek jakie są statkowi potrzebne

Cranes: [Crane] – lista wszystkich dźwigów

neededCrates: [int] - część listy paczek które są statkowi potrzebne. Lista ta jest wysyłana pozostałym agentom.

infoData: [(crateId, requestTime, deliveryTime, waitTime, craneId)] – Lista zawierająca informacje na temat poszczególnych paczek: id paczki, czas kiedy o nią zapytano, czas kiedy ją dostarczono, czas jaki na nią czekano oraz id dźwigu który ją bezpośrednio podał.

Ship zawiera następujące procedury:

mainLoop(): void – główna pętla obiektu

readMessage(): void – odczytuje wiadomość z kolejki wiadomości i podejmuje odpowiednie akcje

Ship korzysta z modułu Message w celu wysyłania i odbierania wiadomości.

Charakterystyka działania modułu

Działanie statku jest zapisane w pętli metody mainLoop. Jest ono następujące:

1. Statek wpisuje sobie do listy neededCrates tyle paczek z listy crates ile jest zdefiniowane w cratesPerMessage.
2. Lista neededCrates jest wysłana pozostałym agentom.
3. Statek zapisuje sobie czas wysłania wiadomości.
4. Statek odczytuje wiadomości od innych agentów – np. o dostarczonej przez kogoś paczce.
5. Jest sprawdzane ile czasu minęło od ostatniej wysyłki. Jeśli minęło przynajmniej tyle ile jest zadeklarowane w zmiennej timeInterval, to przechodzi w pętli spowrotem do punktu 1. Jeśli nie minęło odpowiednio dużo czasu bądź nie ma już żadnych wymaganych paczek do dodania, to statek jedynie nasłuchuje wiadomości.

Display

Moduł który zawiera szczegóły dotyczące grafiki programu.

Opis funkcjonalny modułu

Moduł służy to narysowania grafiki programu, animacji oraz rysowaniu informacji potrzebnych użytkownikowi. Uruchamia moduł pygame który rysuje symulację.

Moduł zawiera następujące metody:

normalizeSize(): void – dopasowywuje rozmiar okna do liczby kratek na mapie

drawCraneBody(Crane, rect): void - rusuje dźwig

drawCranesArms([Crane]): void – każdemu dźwigowi rysuje ramię pod odpowiednim kątem

drawForklift(Forklift, rect): void – rysuje wózek widłowy

drawCratesOnField([id], rect): void – rysuje paczki z ich numerem id na danym polu w kolejności jakiej są na liście

drawInformationWindow(map): void - umożliwia wyświetlanie informacji o kliknięty przez użytkownika obiekcie

Message

Moduł który określa rodzaj przesyłanych wiadomości między agentami.

Opis funkcjonalny modułu

W module zawiera się klasa która tworzy obiekty wiadomości. Każda wiadomość ma swojego nadawcę, typ oraz dane do przesłania. Wiadomość może mieć następujące typy:

SEARCH_PACKAGE: wysyłana przez statek do pozostałych agentów z id paczek które są mu potrzebne

PACKAGE_DELIVERED: wysyłana przez statek z informacją jaka paczka została właśnie dostarczona

PACKAGE_LOADED: wysyłana przez dźwig statkowi o paczce którą właśnie załadował na pokład

HAVE_SHIP_PATH: wysyłana przez dźwig do jego sąsiadów o tym, że zna ścieżkę do statku

NEGOTIATE_FIELD: wysyłana jako propozycja negocjacji o tym na jakie pole odstawić paczkę

NEGOTIATE_OWNERSHIP: wysyłana jako propozycja negocjacji o tym kto ma podnieść paczkę

NEGOTIATE_ANSWER: wiadomość wysyłana jako odpowedź na negocjacje

Protokoły komunikacji

Wiadomości

Statek – Agenci: Potrzebuję paczek

Statek ma podaną w konstruktorze listę ze wszystkimi paczkami jakie będzie potrzebował. Co jakiś czas wysyła informacje do pozostałych agentach o części paczek jakie są mu potrzebne. Dzięki temu, że informacje o potrzebnych paczkach są wysyłane w partiach, pozostali agenci muszą być czujni przez całe działanie programu oraz rozglądać się za nowymi paczkami.

Statek ma ustaloną liczbę paczek w zapytaniu (**cratesPerMessage**) oraz przedział czasu co jaki ma wysyłać zapytania (**timeInterval**). Tworzy odpowiednią podlistę z id paczke które potrzebuje (**neededCrates**) oraz wysyła tą listę do każdego agenta (dźwigów i wózków).

Dźwig – dźwig: Mam dostęp do statku

Dźwig w swojej głównej pętli działania rozgląda się po otoczeniu metodą **examine-Surroundings**. W tej metodzie nie tylko rozgląda się za paczkami jakie ma w swoim polu zasięgu, ale również sprawdza, czy jego zasięg wystarcza aby załadować statek. Jeśli tak jest to wartość zmiennej **directToShip** jest zmieniana z 0 na 1. Następnie od razu informuje swoich sąsiadów o tym, że ma ścieżkę na statek oraz o tym, że odległość do statku jest 0 (bo jest bezpośrednio przy statku).

W odpowiedzi na tą informację, dźwigi przekazują to wszystkim swoim sąsiadom poza nadawcą. Wraz z wiadomością przekazują też informację o odległości do statku w przeskokach. Ta informacja będzie potrzebna po to aby każdy z dźwigów mógł rozstrzygnąć jak daleko znajduje się od statku i mógł tym uargumentować negocjacje o paczki.

Negocjacje

Dźwig – dźwig: podnieś paczkę

Gdy jakaś paczka leży na polu więcej niż jednego dźwigu, to dźwigi muszą ustalić który z nich ma podnieść paczkę i podać ją dalej.

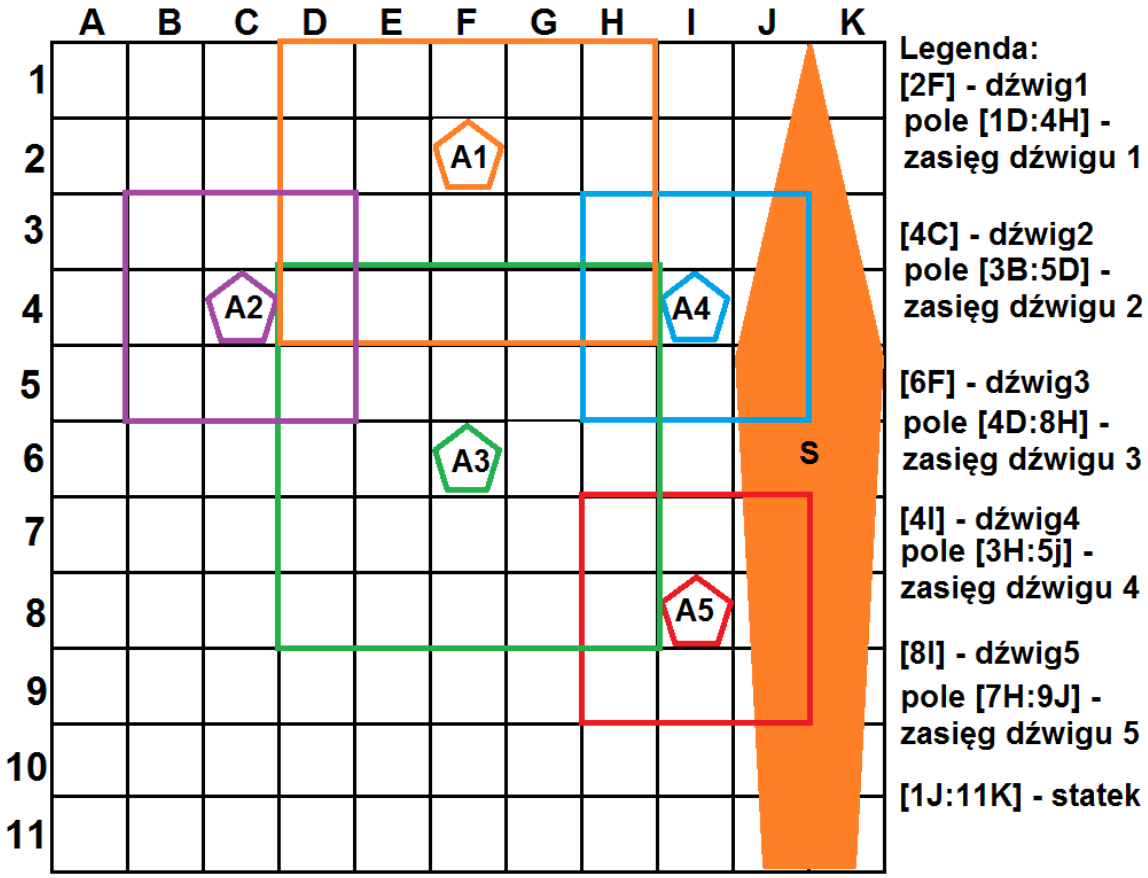
Zainteresowany paczką dźwig pyta sąsiadów paczki czy może sobie ją przywłaszczyć. W wiadomości podaje id paczki, swoją odległość do statku oraz swój średni czas dostarczenia paczki. Dźwig który ma odpowiedzieć na nagocjacje porównuje swoją odległość do statku z odległością nadawcy. Jeśli jego odległość jest dłuższa, to się zgadza, a jeśli jest krótsza to odmawia.

Jeśli jest taka sama, to porównuje swój średni czas dostarczenia paczki oraz nadawcy. Ten który ma krótszy średni czas dostarczenia wygrywa własność nad paczką.

Dźwig – dźwig: podaj paczkę

Gdy dźwig wygrał negocjacje nad paczką, to musi zdecydować któremu z sąsiadów najlepiej opłaca mu się przekazać paczkę dalej.

Rozważmy sytuację z rys.2, gdzie dźwig A3 jest w posiadaniu paczki, którą chce przekacać któremuś z jego sąsiadowi. Sąsiad A2 odpada, bo ma aż dwa przeskoki do statku, a sąsiad A1 ma tyle samo co A3, więc nie przybliży to paczki ku statku. Agent rozważa dźwigi A 4 i A5. Oba leżą bezpośrednio przy statku. Ponieważ odległość w przeskokach jest równa dla obu sąsiadów, o tym kto dostanie paczkę decyduje ich średni czas dostarczenia paczki na statek. Czas jest liczony z dużą dokładnością, więc zapewne będzie się różnił dla obu sąsiadów, lecz na początku programu, gdy żaden z nich nie przekazał jeszcze paczki na statek to średni czas dostarczenia jest równy 0. W takim wypadku obaj sąsiedzi czas dostarczenia paczki na statek jest porównywalny, więc dźwig A3 podaje tą paczkę dowolnemu z nich.



Rys 2. Negocjacje komu podać paczkę

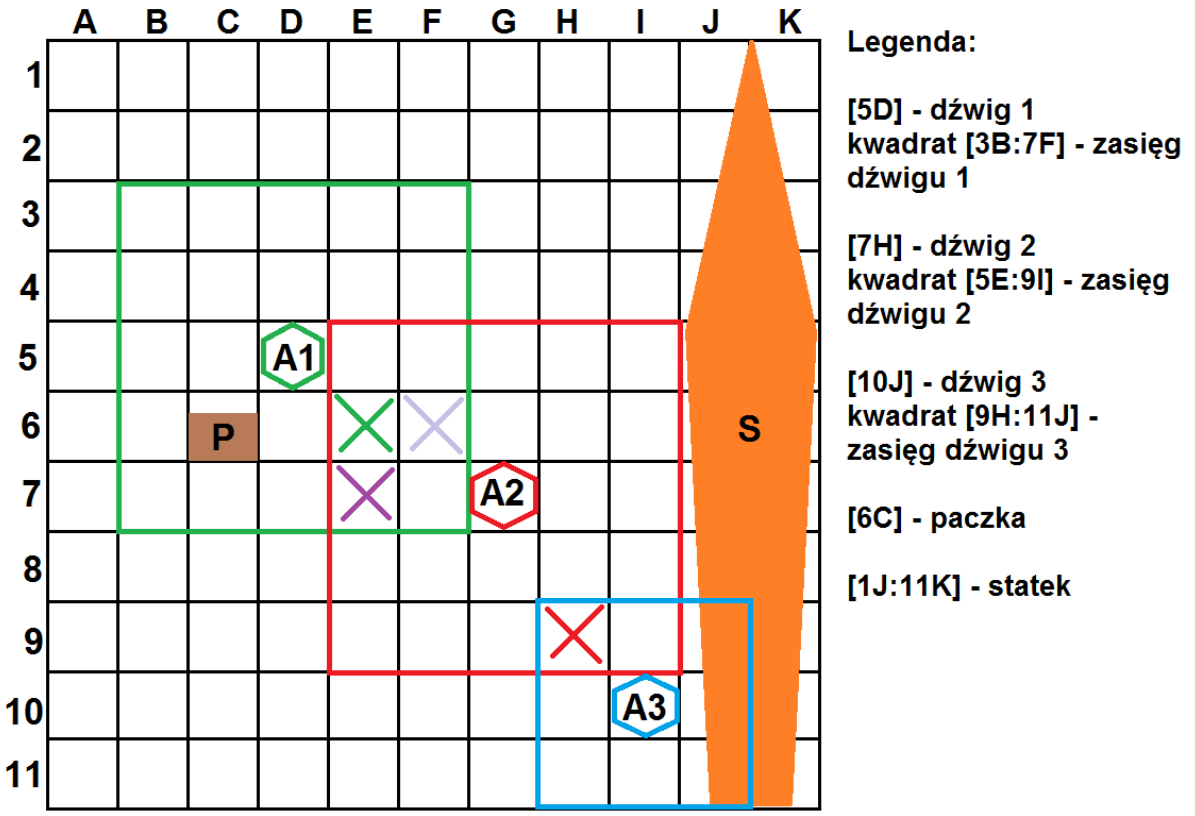
Dźwig – dźwig: na jakie pole podać paczkę?

Gdy dźwig z paczką wie już komu chce ją przekazać to musi ustalić na jakie ze wspólnych pól ma ją położyć. Dźwig który podaje paczkę podałby najchętniej na najbliższe wspólne pole od paczki, jednak dźwig który ma dostać paczkę na swoje pole wolałby żeby była ona tak podana aby nie musiał wykonywac zbyt dużego ruchu kiedy będzie ją podawał dalej.

Na początku wspólne pole na którym ma zostać umieszczona paczka jest losowane (Rys 2., pole 6F). Następnie dźwig który chce podać paczkę proponuje pola które są bliższe pola z paczką. Na rysunku 1 widać, że dźwig A1 zaproponuje pole 6E, ponieważ wtedy przeniesie paczkę z pola 6C jedynie o 2 pola w prawo. Gdyby miał podać na pole wylosowane, to musiałby ją przemieścić o 3 pola.

Jednak agent A2 się nie zgadza, ponieważ z zaproponowanego pola do najbliższego wspólnego pola z sąsiadującym dźwigiem musiałby przenieść paczkę o $\sqrt{3^2 + 3^2} \approx 4.24$ pól, tymczasem z wylosowanego pola przeniesie on paczkę jedynie o $\sqrt{3^2 + 2^2} \approx 3.60$ pola.

Dźwig A1 wysyła kolejne propozycje które są dla niego nie gorsze od wylosowanego pola. Zaznaczone pole 7E powinno zostać wybrane, ponieważ zostanie zaproponowane przez dźwig A1 (odległość od pola z paczką to $\sqrt{1^2 + 2^2} \approx 2.24 < 3$), a dźwig A2 się zgodzi (przemieści paczkę o $\sqrt{3^2 + 2^2} \approx 3.60$, a więc tyle samo co w przypadku pola wylosowanego).



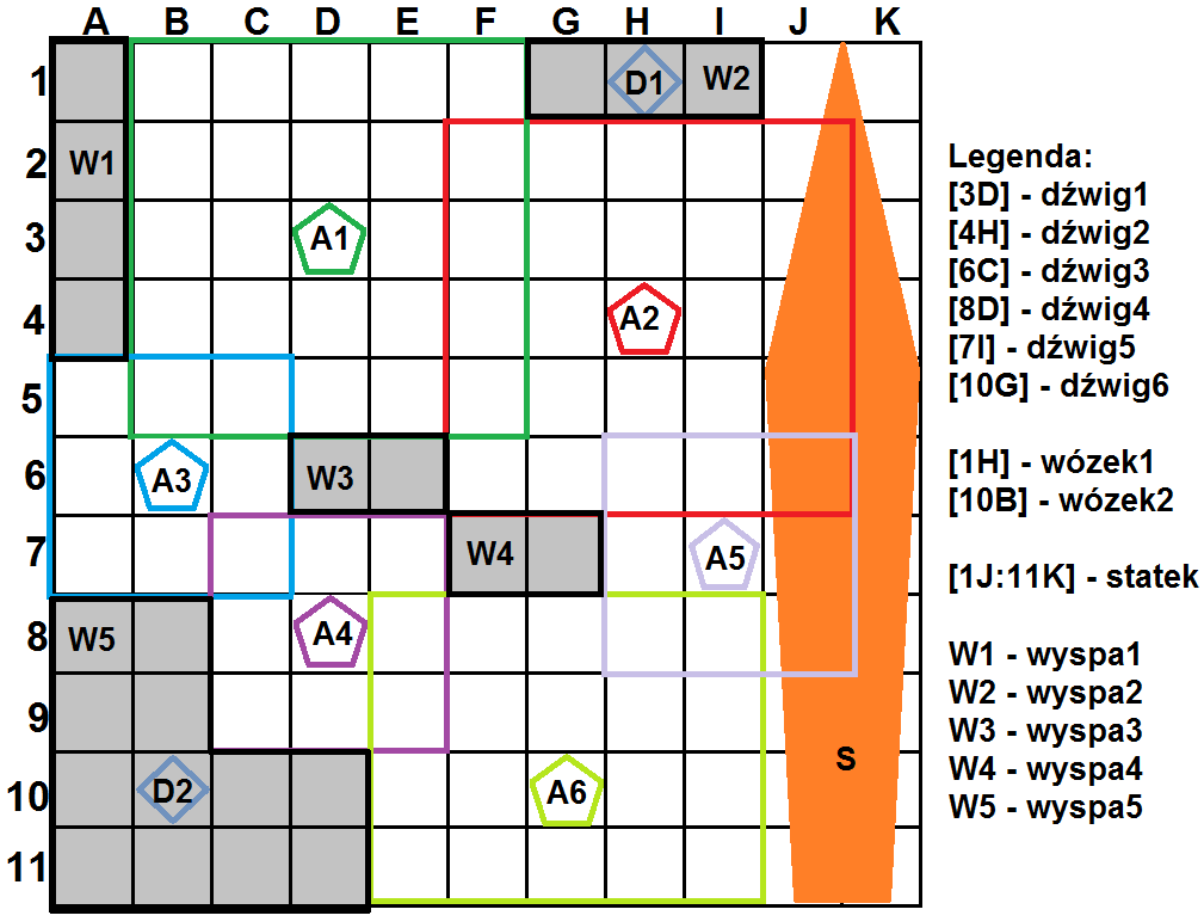
Rys 2. Negocjacje gdzie położyć paczkę

Wózek – wózek: czy mogę wjechać na wyspę?

Port podzielony jest na różne rodzaje pola: CRANE_FIELD, tam gdzie stoją dźwigi, STORAGE_FIELD tam gdzie dźwigi mają zasięg oraz ROAD_FIELD tam gdzie dźwigi nie mają zasięgu, a więc wózki muszą się poruszać po tych polach aby zaobserwować czy tam nie ma potrzebnych paczek. Na rysunku 3 pokazano port gdzie białe pola są w zasięgu dźwigu, a szare nie.

Gdy wózek D1 stojący na polu 1H skończył rozglądać się po swojej wysepce, to chciałby rozejrzeć się po następnej. Znajduje się on najbliżej wyspy W2, więc negocjuje z pozostaływmi wózkami (w tym przypadku z D2) wysyłając komunikat W2. Jeśli D2 zwiedzał tą wyspę wcześniej, to odmawia

wózkowi wjechania na tą wyspe – bo wie, że nie ma takiej potrzeby, ta wyspa jest już zwiedzona. Jeśli D2 nie był na tej wyspie wcześniej, to podaje swoją odległość do wyspy 2. Ponieważ D2 jest dalej od W2 niż D1, to D1 wygra negocjacje i przejedzie się na wyspę 2 poszukać tam potrzebnych paczek.



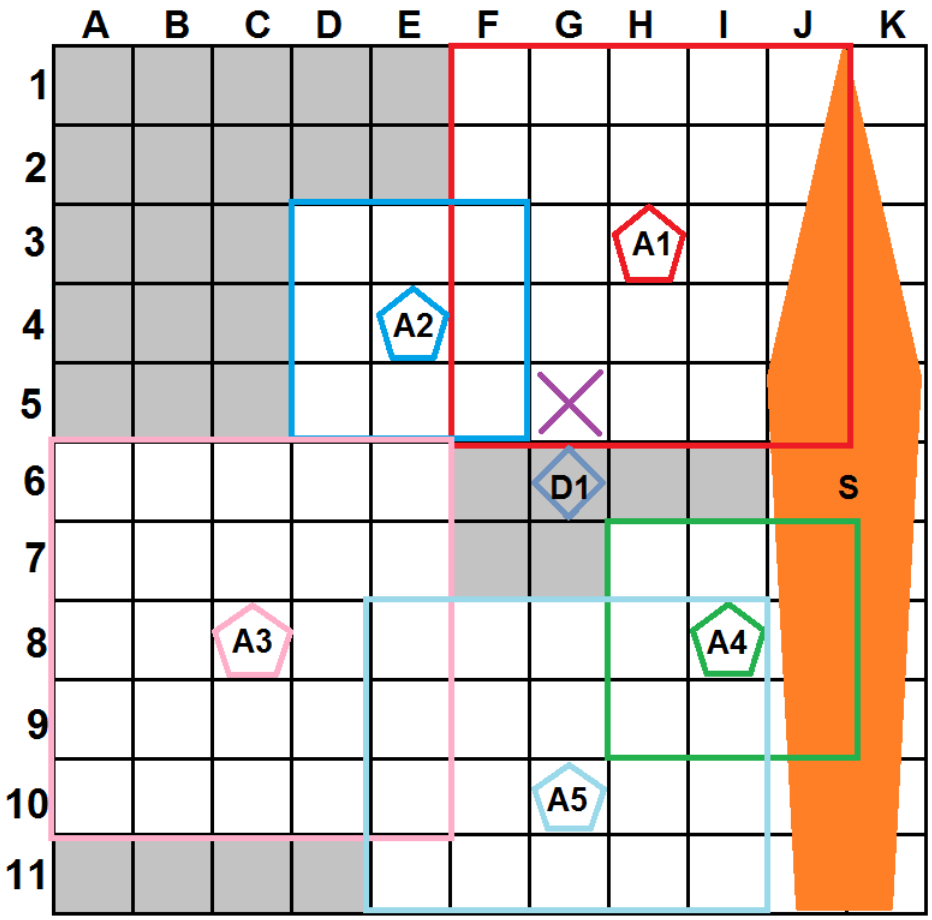
Rys. 3 Negocjacje o wysepki

Wózek – dźwig: podaję paczkę

Kiedy wózek już zdobędzie paczkę, która jest potrzebna statkowi, to musi ją dostarczyć dźwigowi, tak a by ten już mógł pdac ją dalej znaną mu drogą do statku. Wózek powinien wynegocjować któremu z sąsiadujących dźwigów należy się paczka.

Popatrzmy na sytuację z rysunku 4. Wózek D1 na polu 6G jest w posiadaniu paczki, którą chce statek. Rozważa dźwigów – sąsiadów, których pola zasięgu są dookoła wyspy na której znajduje się wózek.

Agent pyta kolejno każdego z dźwigów jaką mają odległość do statku. Dźwigi odpowiadają odległością podaną w przeskokach. Wygrywa dźwig, który podał najmniejszą liczbę przeskoków. Jak widać na rysunku, mamy dwa dźwigi które są bezpośrednio przy statku, A1 oraz A4. W takim przypadku wózek uzna, że negocjacje wygrał ten którego skrajne pole z wysepką jest bliżej wózka. Jest to pole 5G zatem agent A1 wygra paczkę którą chce przekazać wózek.



Legenda:
[3H] - dźwig1
[4E] - dźwig2
[8C] - dźwig3
[8I] - dźwig4
[10G] - dźwig5

[6G] - wózek1

[1J:11K] - statek