Systemy agentowe:
Wieloagentowy symulator rynku
Dokumentacja końcowa

Aleksandra Dzieniszewska
Jakub Łyskawa
Eryk Warchulski
Prowadzący: dr inż. Dominik Ryżko

28 stycznia 2020
wer. 1.0

Spis treści

1 Wprowadzenie

Celem niejszego projektu była implementacja symulatora rynku dóbr konsumpcyjnych w oparciu o paradygmat systemów wieloagentowych. Dokument ten składa się z trzech sekcji: w sekcji (2) omówiony jest model agenta oraz modelu rynku, którego dotyczą symulacje. Sekcja (3) zawiera opis architektury systemu, tj. modułów oraz sposobu ich działania, które składają się na system. Sekcja ostatnia – (4) – zawiera opis przykładowych symulacji, które można przeprowadzać w ramach dostarczonego systemu.

2 Model zjawiska

Przedstawiony w tej sekcji model zjawiska jest spójny z dokumentacją wstępną.[1]

2.1 Model rynku

Rynek determinowany jest poprzez strukturę połączeń między agentami przy czym struktura połączeń jest generowana przez wybrany graf losowy (Barabasi-Albert, dowolony inny lub zadany przez użytkownika). Rynek działa ciągle i po czasie t jego stan jest archiwizowany, co jest niezależne od podmiotów-agentów na rynku.

2.2 Model Agenta

2.2.1 Zasoby

- agent A_i w chwili t posiada $Z^{A_i}(t)$ zasobu i ma możliwość wygenerować większą jego ilość, która będzie go kosztowała $g(z)$, gdzie z jest przyrostem zasobu
- agent może przechowywać zasób lub go sprzedać, wchodząc w negocjacje handlowe z pozostałymi agentami na rynku, z którymi agent jest połączony (patrz struktura połączeń)
- produkcja agenta jest ograniczona przez $P^{A_i}_{max}(t, \delta t)$
- każdy agent posiada maksymalny stan magazynowy zasobu Z, którego nie może przekroczyć, i wynosi on M^{A_i}
- jeśli agent przekroczy maksymalny stan posiadania M^{A_i}, to zobligowany jest do zapłacenia kosztu utylizacji nadmiarowej ilości zasobu Z
- agenty mają potrzeby konsumpcyjne $C^{A_i}(t, \delta t)$, które chcą zaspokoić
- jeśli agent nie zaspokoi swoich potrzeb konsumpcyjnych po czasie T od ich wygenerowania, to zobligowany jest do zapłacenia kosztu
- agenty posiadają na starcie określoną ilość środka wymiany K^{A_i}, który jest im przydzialny w sposób losowy lub zdeterminowany przy inicjalizacji systemu
- agent otrzymuje środek wymiany zgodnie z funkcją $f^{A_i}()$

2.2.2 Polityka decyzyjna

Polityka decyzyjna określa zachowanie agentów na rynku.

Przyjmuje się, że polityka decyzyjna agenta sparametryzowana jest następującymi wielkościami:

- obecne zapotrzebowanie agenta $R \geqslant 0$
- czas, w którym agent musi zaspokoić swoje zapotrzebowanie T_s liczony od czasu startu sesji

[1] Dokumentacja wstępna zamieszczona jest pod niniejszym adresem.

- obecny stan agenta S, który jest liczbą posiadanych jednostek zasobu Z przez agenta
- obecny budżet agenta B, który jest liczbą posiadanych jednostek wymiany K przez agenta
- funkcją kosztu produkcji $g(z)$
- funkcją limitu produkcji $P(t, \delta t)$
- kosztem utylizacji dóbr nadmiarowych M_c
- kosztem niezaspokojenia potrzeb konsumpcyjnych C
- limitem posiadanych jednostek zasobu M

Agent w oknie czasowym T_w, wyznaczającym czas trwania negocjacji, generuje oferty sprzedaży (obiekt `Os`) oraz kupna (obiekt `Ob`), na które nałożone są limity:

- `Ob.value` $\leqslant B \wedge$ `Ob.n` $\leqslant M - S$, które kolejno oznaczają: cena zakupionej ilości towaru nie może przekraczać budżetu agenta oraz ilość zakupionego towaru nie może być większa od dostępnej jeszcze liczby jednostek zasobu, które agent może przechowywać.

- `Os.n` $\leqslant S$, tj. ilość sprzedanego towaru nie może być większa pod stan posiadania agenta.

Na podstawie powyższych ustaleń proponowana polityka decyzyjna agenta może być wyglądać następująco:

- `Os`, `Ob` są aktualnymi ofertami kupna i sprzedaży
- `Ns` jest agentem inicjalizującym transakcję

```
buyer initializes
  initial buy offer = (rand(R - S, M - S), 0 if Ob empty else
                  min(Ob).value * rand(0, 1))

  seller counter offer:
      n = min{S, O'b(Ns).number}
      value = random with boundaries:
          value >= max(g(S), O'b(Ns).value)
          if O's not empty:
              value < min{O's(n) where n is not self}
  buyer counter offer:
      n = keep previous w.r.t. limits
      if n = 0 then withdraw
      value = random with distribution depending on Ts and boundaries:
          value <= min{O's} & value >= previous
```

2.2.3 Protokół komunikacyjny

Komunikacja bazuje na protokole konwersacji o akcji i może się odbywać w jednym z dwóch trybów:

- `1-1`, tj. agent formułuje ofertę `O` kupna lub sprzedaży (patrz polityka decyzyjna) i przekazuje ją wyłącznie do jednego agenta
- `1-m`, tj. komunikacja typu *broadcast*, w której agent formułuje ofertę `O` kupna lub sprzedaży i rozsyła ją do co najmniej dwóch różnych agentów.

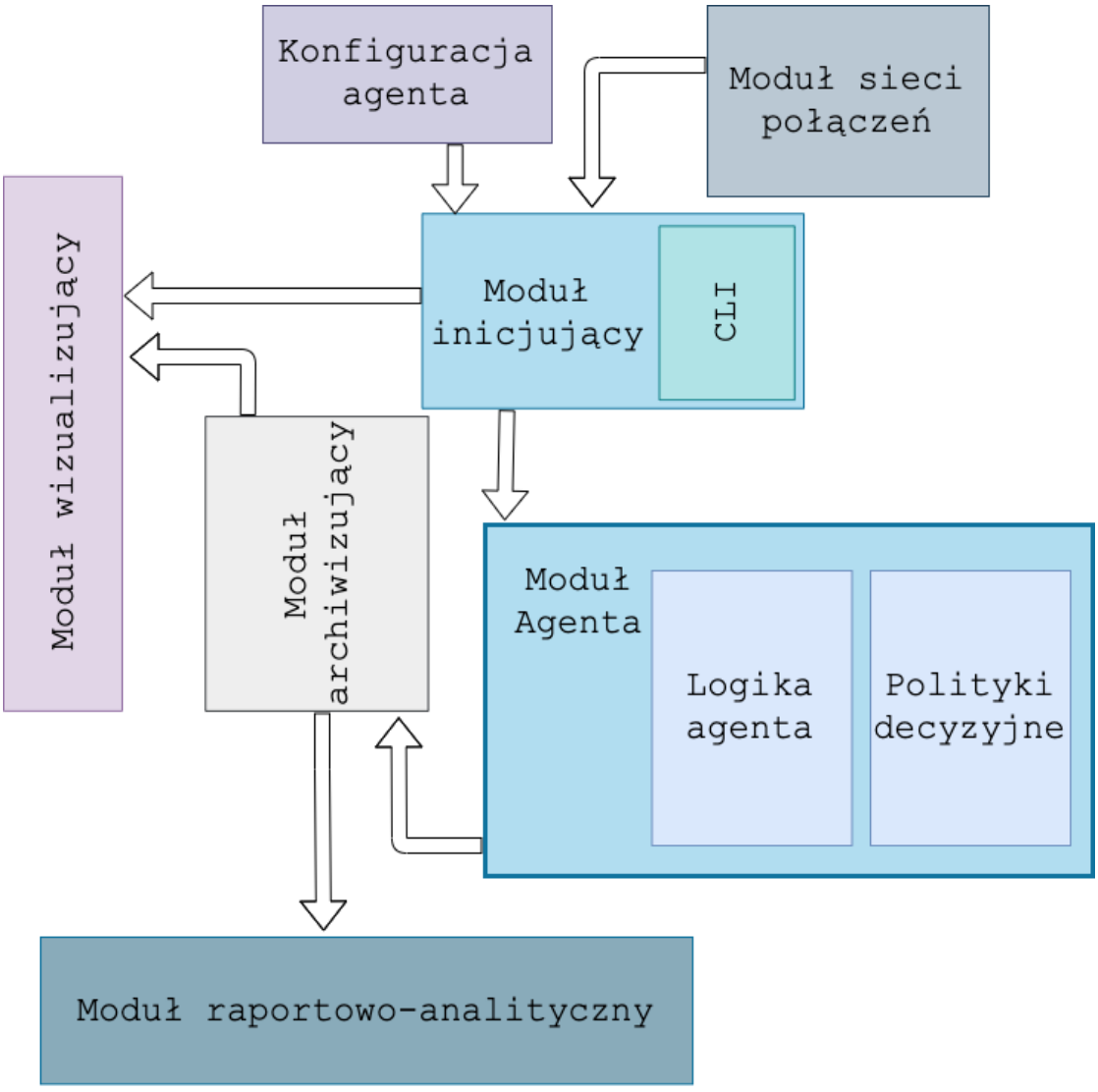
Oferta jest parą (i, p), na którą składa się:

- i liczba jednostek zasobu Z, które agent chce sprzedać lub kupić w trakcie transakcji z kontrahentami, tj. agentami przyjmującymi ofertę sprzedaży lub kupna od agenta inicjalizującego komunikację
- p oferowana cena kupna lub sprzedaży

Jeśli agent-kontrahent nie przystąpi do negocjacji z agentem oferującym po czasie t_{out}, to komunikacja między tymi agentami jest zerwana. Pozostałe warunki zerwania komunikacji między agentami wyznaczone są przez parametry polityki decyzyjnej agentów lub maksymalny czas oczekiwania na odpowiedź τ. Jeśli odpowiedź kontrahenta w trakcie negocjacji przyjdzie po czasie τ, to agent inicjujący negocjacje zrywa ją.

3 Architektura systemu

Rysunek (1) przedstawia architekturę implementowanego systemu. W sekcji tej znajduje się omówienie każdego modułu składowego systemu i przedstawienie wzajemnych relacji między nimi. Do części z nich podane są diagramy klas UML – w niektórych przypadkach z niepełnym interfejsem, co wynika z dużej liczby metod oferowanych przez daną klasę. Ponadto w dostarczonym kodzie źródłowym znajduje się dokumentacja każdej używanej funkcji lub metody klas wraz opisem parametrów wejściowych. Opis zawarty w tej sekcji abstrahuje od szczegółów implementacyjnych.



Rysunek 1: Architektura systemu z zaznaczanonymi modułami składowymi.

3.1 Moduł inicjujący

Jest to moduł odpowiedzialny za inicjalizację wszystkich pozostałych składowych systemu oraz uruchomienie środowiska symulacji. Moduł ten:

- uruchamia moduł archiwizujący, w którym zapisywane są zdarzenia generowane przez agentów oraz stan systemu
- wczytuje z plików konfiguracyjnych strukturę połączeń między agentami oraz ich parametry wewnętrzne
- włącza interfejs graficzny pozwalający na komunikację z systemem za pomocą zbioru komend
- włącza moduł wizualizaujący sieć połączeń oraz zdarzenia, która zachodzą w trakcie symulacji.

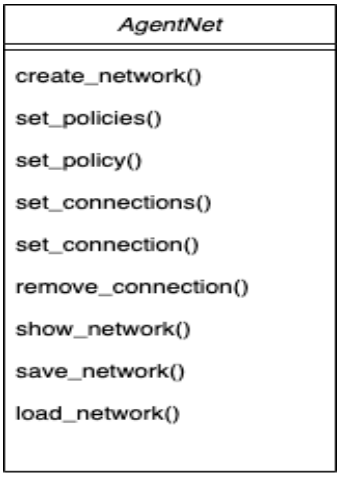
3.1.1 Sposób użycia

W celu aktywacji modułu inicjalizującego, a tym samym całego systemu symulacyjnego należy użyć poniższej komendy
`python run.py [-h] -config CONFIG [-log LOG] [-log_stream out,err] [-vis VIS]`
w której należy podać:

- ścieżkę do pliku z konfiguracją agentów `CONFIG`
- ścieżkę do pliku, w którym będzie zapisywane archiwum zdarzeń zaszłych w trakcie symulacji `LOG`
- strumień na który moduł archiwizujący będzie wypisał zdarzenia zaszłe w trakcie symulacji
  - `err`
  - `out`
- flagę aktywującą moduł wizualizacyjny `VIS`.

3.2 Moduł sieci połączeń

Moduł ten odpowiada za stworzenie sieci połączeń między agentami. Na rysunku (2) zawarty jest diagram klas UML przedstawiający metody dostarczane przez ten moduł.



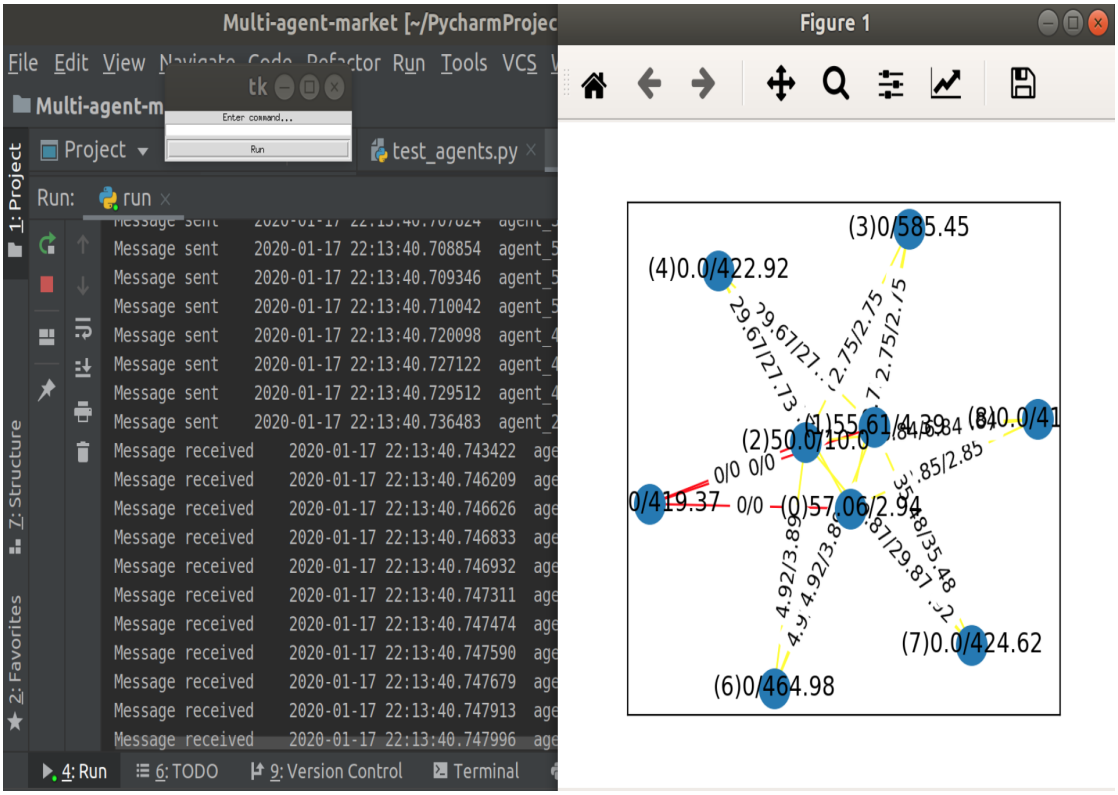
Rysunek 2: Diagram klas modułu sieci połączeń.

Poza stworzeniem sieci połączeń moduł również ładuje polityki decyzyjne agentów oraz umożliwia zapis/odczyt sieci i jej parametrów do/z plików `YAML`.

Moduł inicjujący odpowiedzialny jest również za aktywowanie interfejsu linii komend, w której użytkownik może komunikować się z systemem w trakcie trwania symulacji. Dostępne możliwe komendy są następujące:

- wyłączenie agenta z symulacji komendą o składni `kill [agent id]`
- ponowne włączenie agenta do symulacji komendą o składni `restore [agent id]`
- wyłączenie symulacji komedną `shutdown`.

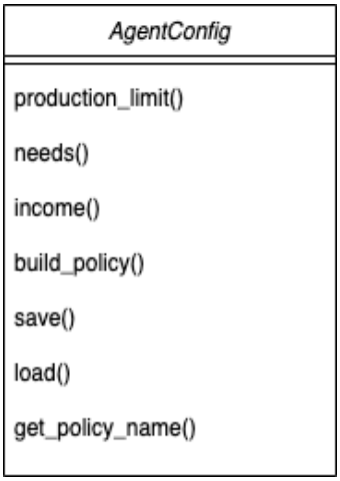
W celu wprowadzania komend dedykowane jest specjalne okno, które widnieje na rysunku (3) prezentującym działanie systemu.



Rysunek 3: System wypisuje archiwum zdarzeń na strumień standardowego wyjścia oraz widoczne jest działanie modułu wizualizaującego. Okno przy lewym górnym rogu zapewnia `CLI`.

3.3 Moduł konfiguracyjny

Moduł konfiguracyjny odpowiedzialny jest za ustawienie wszystkich wartości parametrów agenta, które zostały opisane w sekcji (2). Na rysunku (4) zawarty jest diagram klas UML przedstawiający metody dostarczane przez ten moduł.



Rysunek 4: Diagram klas modułu konfiguracyjnego.

Metody:

- `production_limit()`
- `needs()`
- `income()`

zwracają na podstawie aktualnego czasu symulacji oraz pewnego kroku czasowego *dt* skalarną wartość odpowiedzialną za – kolejno – limit produkcji, potrzeby konsumpcyjne agenta oraz przychód.

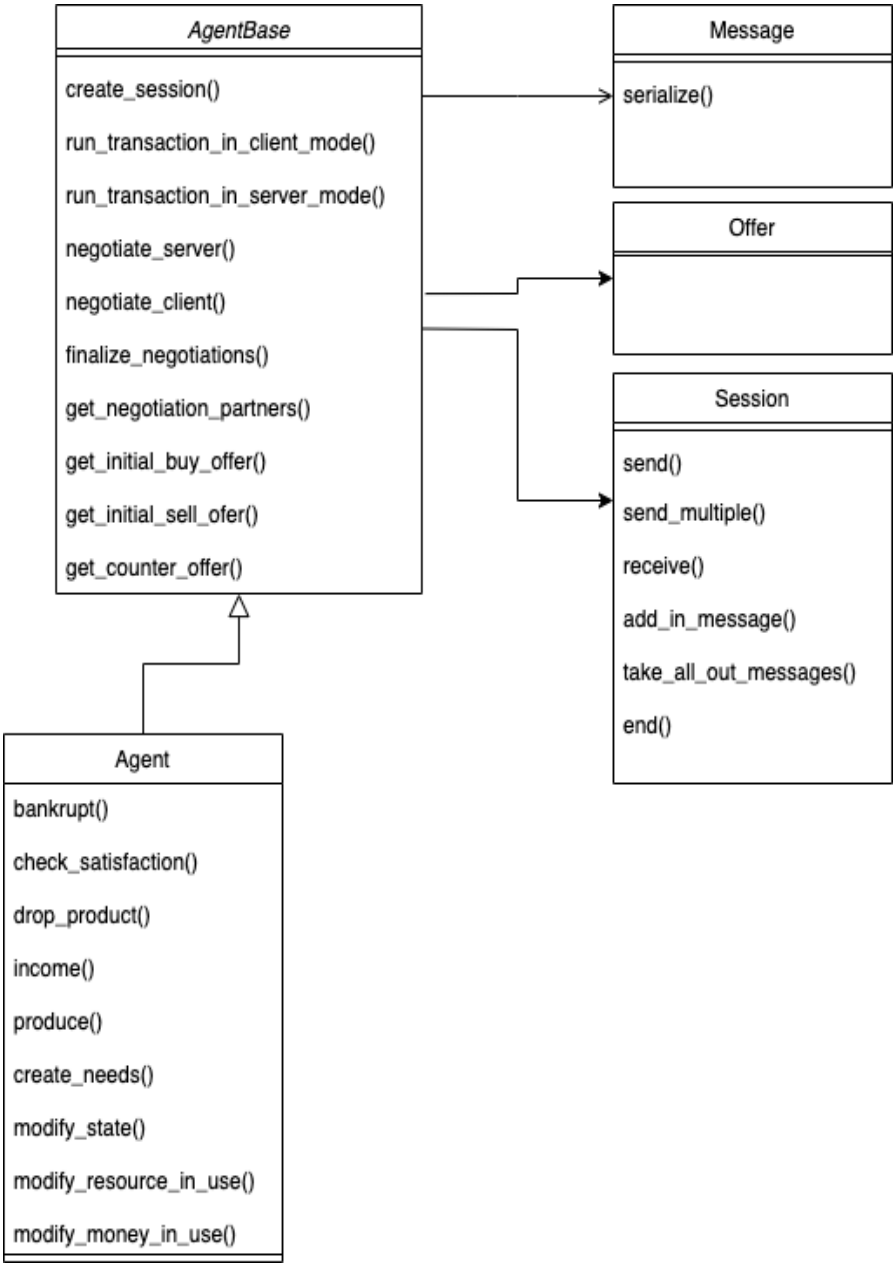
Pozostałe metody odpowiadają za wywołanie konstruktura polityk decyzyjnych oraz operacje zapisu/odczytu parametrów konfiguracyjnych z plików `YAML`.

3.4 Moduł Agenta

Moduł ten jest najbardziej złożonym elmentem systemu i składa się z dwóch podmodułów, które zostaną krótko scharakteryzowane:

- modułu logiki agenta
- modułu polityk decyzyjnych agenta.

Na rysunku (5) znajduje się diagram klas przedstawiający elementy składowe modułu agenta, tj.:



Rysunek 5: Diagram klas modułu agenta.

- klasę `AgentBase` implementującą model agenta zgodny z (2) oraz zachowania agenta związane z generacją oraz wysyłaniem wiadomości
- klasę `Agent`, tj. klasę dziedziczącą po `AgentBase`, w której zaimplementowana jest logika działania agenta
- klasę `Offer`, która reprezentuje ofertę formułowaną i nadawaną przez agenta

- klasę `Message`, która reprezentuje wiadomości wysyłane między agentami
- klasę `Session`, która modeluje sesję utrzymywaną przez agenta w ramach jego działania na rynku.

3.4.1 Logika agenta

Logika agenta jest implementowana w klasie `Agent` i składają się na nią wszystkie możliwe akcje podejmowane przez agenta w trakcie trwania sesji.
Zbiór akcji agenta składa się z następujących elementów:

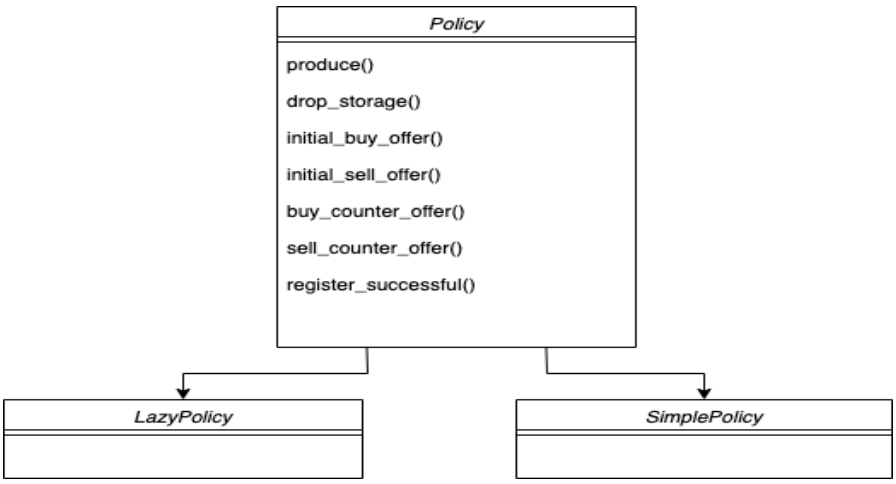
- utworzenie oferty (klasa `CreateOffers`)
- generacja produktu (klasa `GenerateProduct`)
- usunięcie produktu (klasa `DropProduct`)
- generacja potrzeb (klasa `ManageNeeds`)
- generacja przychodu (klasa `GenerateIncome`)

Wszystkie zachowania są modelowane jako zachowania cykliczne (*CyclicBehaviour*) w sensie definiowanym przez bibliotekę `spade`.

3.4.2 Polityki decyzyjne

Podmoduł polityk decyzyjnych jest odpowiedzialny za przebieg wykonania akcji definiowanych w ramach logiki agenta.

Na rysunku (6) zawarty jest diagram klas UML przedstawiający metody dostarczane przez ten moduł.



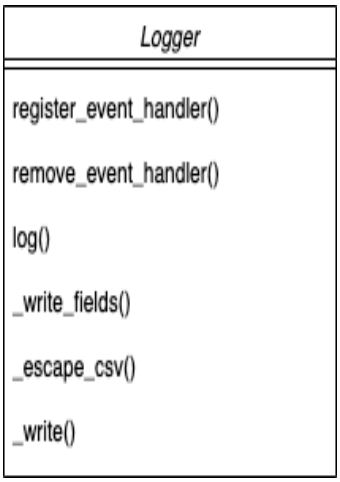
Rysunek 6: Diagram polityk decyzyjnych.

Klasa `Policy` jest klasą abstrakcyjną po której dziedziczą klasy `LazyPolicy` oraz `SimplePolicy`.

Użytkownik posiada dowolność w definiowaniu klas polityk agentów z dokładnością do zgodności typów zwracanych przez funkcje.

3.5 Moduł archiwizujący

Moduł ten pozwala na wypisywanie informacji o stanie systemu na standardowe wyjście. Na rysunku (4) zawarty jest diagram klas UML przedstawiający metody dostarczane przez ten moduł.



Rysunek 7: Diagram klas modułu archiwizującego.

Umożliwia on także zapisywanie tych informacji do pliku. Na podstawie danych z tego modułu tworzona jest wizualizacja symulacji oraz część analityczna. W tabeli (1) zawarte są wszystkie zdarzenia, które są przekazywane do modułu archwizującego.
Zdarzenia zostały podzielone na domeny, których dotyczą, tj.:

- domena **Logger** dotyczy zdarzeń związanych z modułem archiwizującym
- domena **Agent** dotyczy zdarzeń związanych ze stanem agenta
- domena **Sesja** dotyczy zdarzeń związanych z sesją handlową prowadzoną przez agenta
- domena **System** dotyczy zdarzeń związanych z działaniem systemu

Domena	Zdarzenia
Logger	*EVENT_LOGGER_INITIALIZED*
Agent	*EVENT_AGENT_INITIAL_OFFER,* *EVENT_AGENT_OFFER_CHANGED,* *EVENT_AGENT_OFFER_ACCEPTED,* *EVENT_AGENT_OFFER_REJECTED,* *EVENT_AGENT_STATE_CHANGED,* *EVENT_AGENT_BANKRUPTED*
Sesja	*EVENT_MESSAGE_SENT,* *EVENT_MESSAGE_RECEIVED,* *EVENT_SERVER_NEGOTIATION_BREAKDOWN,* *EVENT_CLIENT_NEGOTIATION_BREAKDOWN,* *EVENT_SERVER_FAILED_TO_RESPOND*
System	*EVENT_SYSTEM_INITIALIZED,* *EVENT_SYSTEM_CLOSE,* *EVENT_EXCEPTION,* *EVENT_CLI*

Tabela 1: Tabela ze wszystkimi zdarzeniami, które odnotowuje moduł archiwizujący.

3.6 Moduł wizualizujący

Moduł ten pozwala na wyświetlanie symulacji w czasie rzeczywistym.

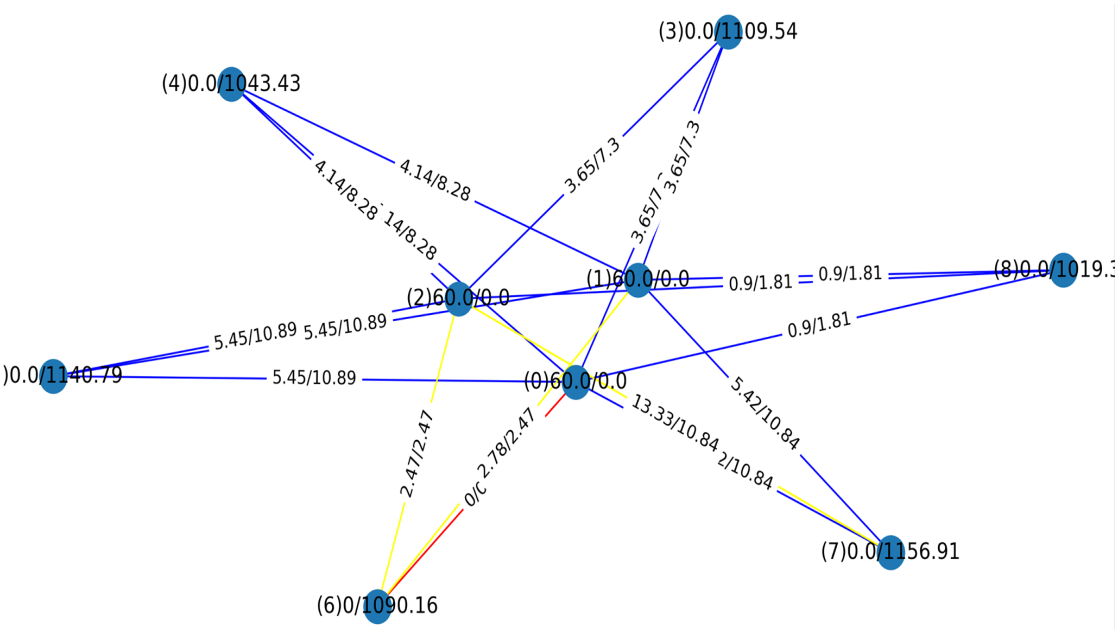
W węzłach grafu sieci połączeń przechowwany jest obecny stan agenta w formacie `liczba produktu/liczba środków wymiany`.

Kolor krawędzi oznacza jaka akcja wykonywana jest w danym momencie przez agentów w ramach negocjacji handlowych:

- czarny – bezczynny
- niebieski – wysłanie oferty
- żółty – wysłanie kontroferty
- czerwony – zerwanie negocjacji
- zielony – zaakceptowanie oferty

Krawędzie opisane są liczbą produktu oraz środka wymiany, które są przedmiotem oferty w identycznym foramcie jak stan agenta w węźle.

Na rysunku (8) zaprezentowana jest przykładowa wizualizacja symulacji.



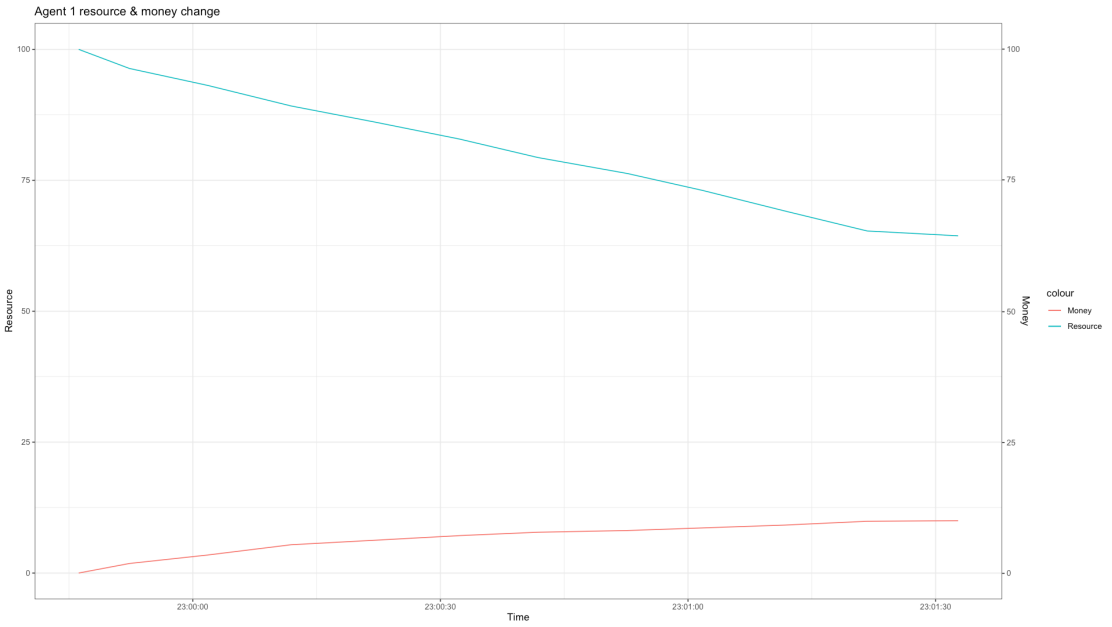
Rysunek 8: Wizualizacja symulacji złożonej z 9 agentów. Agent (3) posiada 0 jednostek produktu oraz 1109.54 jednostek środka wymiany.

3.7 Moduł raportowo-analityczny

Moduł ten składa się z szeregu funkcji napisanych w języku `R`, które na podstawie archiwum utworzonego w trakcie po przeprowadzeniu symulacji, pozwalają na:

- ekstrakcje ustandaryzowanych ramek danych
- wizualizacji stanu agentów w czasie trwania symulacji, tj. ich liczby jednostek produktu oraz środka wymiany
- wizualizacji przebiegu ceny produktu w ramach ofert sprzedaży i kupna.

Przykładowy przebieg utworzony na podstawie archiwum symulacji znajduje się na rysunku (9).



Rysunek 9: Przebieg liczby posiadanych jednostek produktu oraz środków wymiany przez agenta.

4 Eksperymenty

W celu zaprezentowania możliwości zaprojektowanego i zaimplementowanego systemu w ramach projektu przeprowadzono trzy eksperymenty:

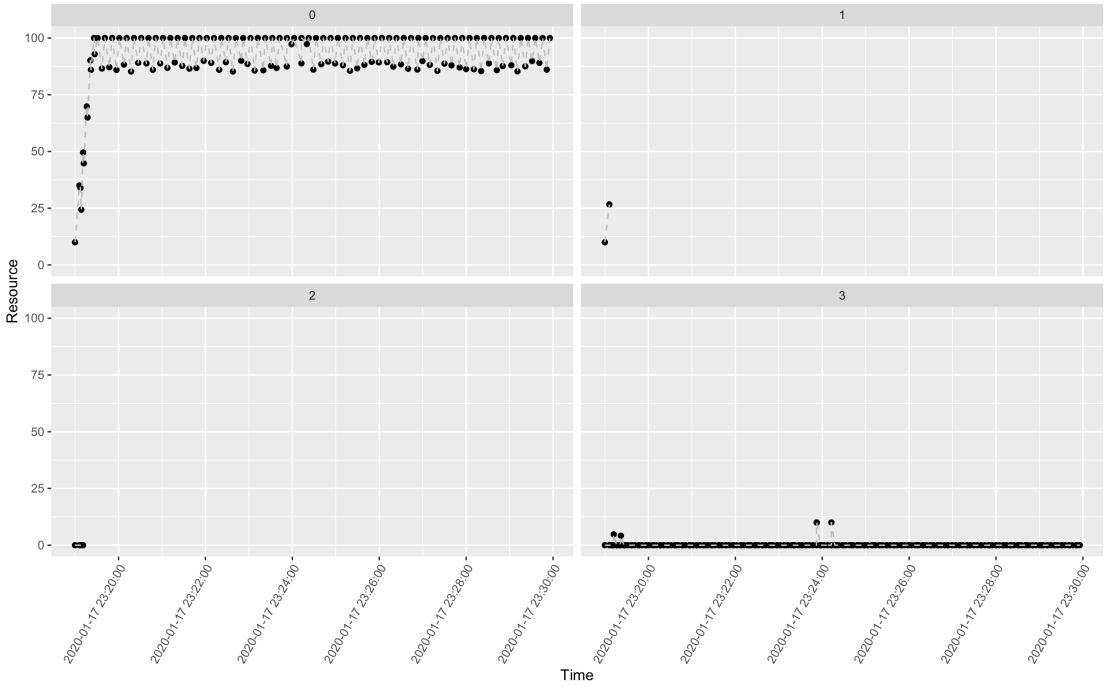
- generacja monopolu, tj. eksperyment mający na celu zbadanie wpływu kosztu produkcji na możliwość utworzenia się monopolu na rynku dóbr konsumpcyjnych
- działanie sieci wielkoskalowej, tj. eksperyment, w którym sprawdzano jak działa system w obliczu przeprowadzania symulacji dla wielu agentów oraz reakcje systemu na wyłączenie jednego z agentów
- odporność systemu na awarie, tj. eksperyment, w którym sprawdzono zachowanie systemu po wyłączeniu agenta ze zbioru agentów aktywnych oraz jego włączenia.

Ze względu na konieczną wiedzę ekspercką zakresu funkcjonowania rynków dóbr konsumpcyjnych i zachowań realnych agentów w tym środowisku – poniższe eksperymenty mają jedynie charakter demonstracyjny.

4.1 Generacja monopolu

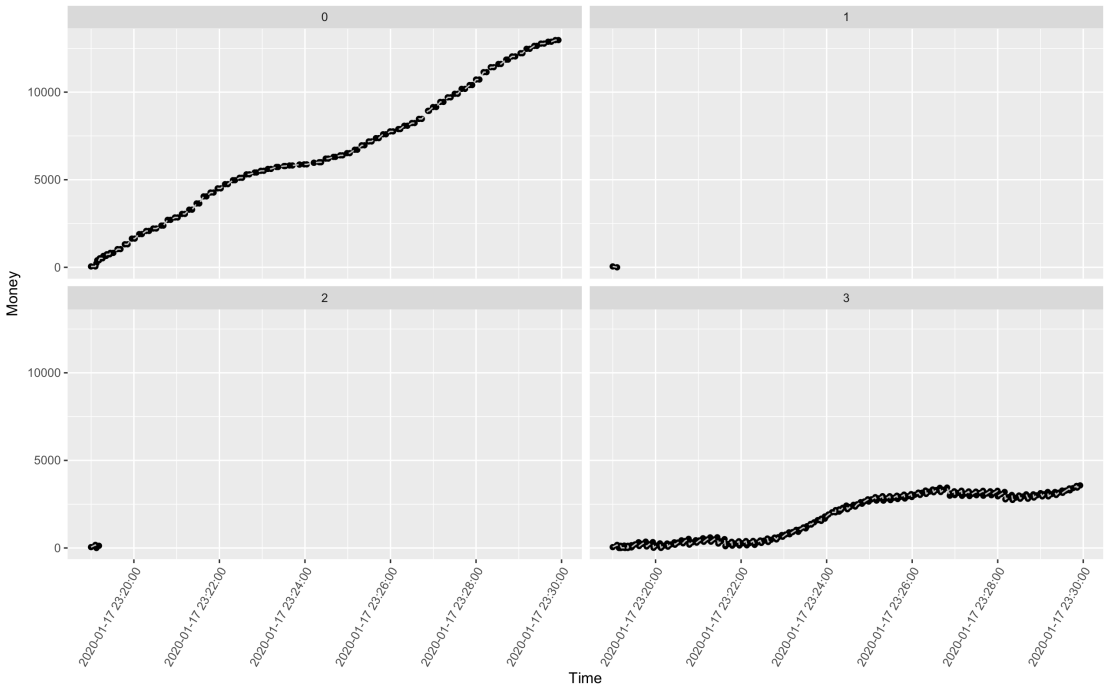
W celu zbadania zjawiska generacji monopolu na rynku dóbr konsumpcyjnych wykorzystano konfigurację agentów, w której wyróżniono łącznie 4 agentów z czego dwóch sprzedających dobra oraz dwóch wyłącznie kupujących je. Jeden z agentów sprzedających dobra (agent o numerze ID `0`) posiadał ponadto bliska zera koszt produkcji, co było – uproszczonym – warunkiem powstania monopolu.
Sieć połączeń między agentami stanowiła graf pełny. Rysunek przedstawia (10) przedstawia stan

posiadania jednostek produktów, które przypadały na agenta. Na jego podstawie można zawaużyć, że negocjaje handlowe przebiegały głównie między agentem od numerze ID równym `0` oraz agentem o numerze ID równym `3`. Brak chęci udziału uczestniczenia w sesji handlowej pozostałych agentów wynikał z ustawienia parametrów ich polityk decyzyjnych oraz parametrów początkowych przy czym brak zaznaczonego przebiegu dla agentów o ID `1` oraz `2` oznacza samoistne wyłączenie się z sesji.



Rysunek 10: Przebieg liczby posiadanych jednostek produktu przez agentów. Brak zaznaczonego przebiegu dla agentów o ID `1` oraz `2` oznacza samoistne wyłączenie się z sesji.

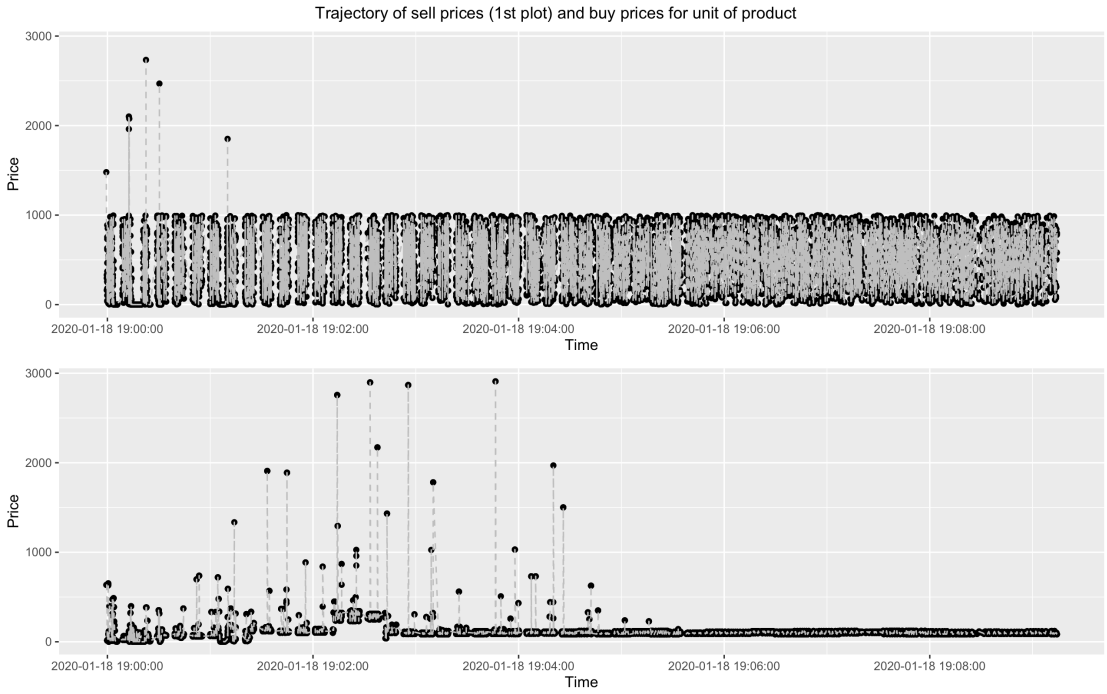
Dominująca pozycja na rynku podmiotu o ID równym `0` ukazała się również w fakcie posiadanych przez niego jednostek środka wymiany, co obrazuje rysunek (11)



Rysunek 11: Przebieg liczby posiadanych środków wymiany przez agentów

4.2 Działanie sieci wielkoskalowej

W ramach eksperymentów na systemie sprawdzono również w jaki sposób działa on, gdy liczba agentów jest duża – w przypadku przyjętej konfiguracji było to dziewięciu agentów. Sieć połączeń między nimi jest tożsama z siecią połączeń przedstawioną na rysunku (8).
Rysunek (12) przedstawia zmianę cen produktu w ramach ofert sprzedaży oraz kupna. Ze względu na dużą liczbę agentów, a tym samym bardzo dużą liczbę zdarzeń na rynku cena podlegała częstym zmianom, co zauważalne jest po zagęszczeniu punktów na obu wykresach.



Rysunek 12: Przebieg liczby posiadanych środków wymiany przez agentów

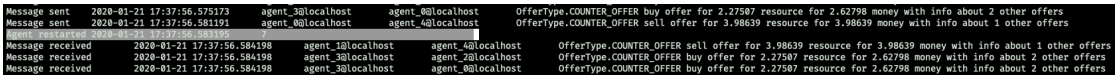
4.3 Odporność systemu na awarie

W celu zbadania odporności systemu na awarie zbadano jego zachowanie w obliczu wyłączenia jednego z agentów. W związku z faktem, że na infrastrukturę systemu składają się wyłącznie agenty i to na podstawie ich stanu generowana jest sieć połączeń lub przebieg symulacji, przeprowadzenie takiego eksperymentu i przeanalizowanie jego wyników powinno zwrócić kompletną informację o t.zw. *fault-tolerant* zaimplementowanego systemu.

Warunki początkowe eksperymentu są równoważne warunkom z eksperymentu opisanego w sekcji (4.2). W trakcie jego trwania przy pomocy dostarczanej linii komend wyłączono agenta o ID równym 7, co zostało odnotowane przez moduł archiwizujący i jest widoczne na rysunku (13). Ponadto widoczne jest tam, że agenty dalej podejmują dostępne akcje z tą różnicą, że z ich sieci połączeń wyłączony jest agent o ID równym 7.

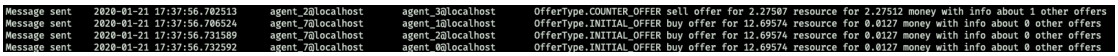
Rysunek 13: System poprawnie odnotowuje wyłączenie jednego z agentów.

Po pewnej chwili wyłączony wcześniej agent został przywrócony, co jest widoczne na rysunku (**??**).



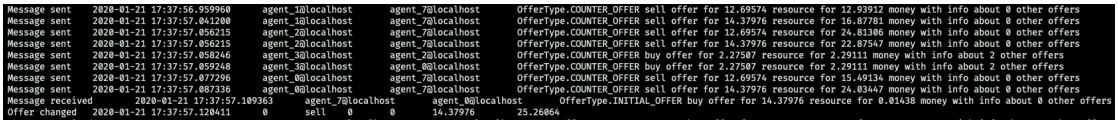
Rysunek 14: System poprawnie odnotowuje przywrócenie jednego z agentów.

Przywrócenie agenta spowodowało, że ponownie zaczął on brać udział w transakcjach handlowych, co można zauważyć na rysunku (15).



Rysunek 15: Przywrócony agent formułuje oferty handlowe.

Ponadto pozostałe agenty w systemie poprawnie reagują na jego oferty handlowe, co świadczy o poprawnym włączeniu uprzednio wyłączonego agenta do ich sieci połączeń (patrz rys. 16).



Rysunek 16: Pozostałe agenty poprawnie reagują na obecność przywróconego agenta.