Dokumentacja

Mateusz Szczepański

Symulacja zarażeń w różnych środowiskach

Wstęp

W poniższym pliku znajduje się dokumentacja dotycząca tytułowego projektu. Projekt polega na stworzeniu symulacji epidemii społeczeństwa ludzi, pod różnymi rządami, wskutek różnych wirusów i posiadając różne szczepionki. W każdej numerowanej sekcji zostanie omówiony szczegółowo każdy wykorzystany moduł (wykorzystane zostały również biblioteki random oraz matplotlib). Do projektu dołączony jest również diagram UML, którego poszczególne elementy zostaną omówione na koniec pliku. W każdym module innym od *Events* in UnitTests znajduje się klasa nazywająca się tak samo jak moduł.

Z uwagi na ogromną złożoność problemu zarażeń ludzi symulacja ta jest jedynie bardzo prostym schematem, w jakim wirus mógłby się rozprzestrzeniać. Wiele schematów symulacji zostało zaprojektowanych na podstawie liczb i zjawisk losowych, a wykorzystane stałe zostały dobrane na podstawie statystyk światowych i zdrowego rozsądku. Zarażanie się wirusem odbywać będzie się tylko poprzez spotkanie dwóch osób ze sobą, czyli drogą kropelkową. Zakładamy w naszym modelu również, że wszystkie osoby są od siebie niezależne i nikt z nikim nie mieszka. Możliwym jest stworzenie własnej autorskiej populacji, wirusa, szczepionki, ustalenie długości trwania epidemii i testowania zachowania naszych populacji w obliczu fantazji użytkownika. Projekt będzie na bieżąco rozwijany, a wszelkie sugestie prosi się o wysyłanie na mail: mateusz.szczepanski.kontakt@gmail.com

1. Person

W pierwszym module znajduje się klasa Person, która odpowiada za tworzenie ludzi w symulacji. Są oni podstawowym nośnikiem wirusa. Posiada ona następujące atrybuty:

- *id: int (unikatowe id każdej osoby)*
- *likes_gov: bool (stwierdzenie czy osoba jest zwolennikiem obecnego rządu)*
- *job: str (aktualnie wykonywany zawód)*
- *hand_washing_level: int 1:5 (poziom mycia rąk)*
- *infected: bool (informacja czy osoba jest zarażona)*
- *resist: bool (informacja czy osoba jest zaszczepiona)*
- *dead: bool (informacja czy osoba jest martwa)*

W przyszłości planowane jest stworzenie więcej atrybutów takich jak *hand_washing_level* co pozwoli stworzyć bardziej szczegółową i skomplikowaną symulację. Wysoki poziom mycia rąk daje mniejsze prawdopodobieństwo zarażenia się wirusem. Atrybut *job* wpływa na wiele aspektów, przykładowo, brak medyków w populacji sprawia, że nie jest możliwe aplikowanie szczepionek, a tylko osoby będące niewolnikami mogą umrzeć na koncercie Elvisa.

W klasie Person znajdują się również następujące metody:

```
+ is_infected()
  (zwraca True, gdy osoba jest zarażona oraz False w przeciwnym razie)

+ is_dead()
  (zwraca True, gdy osoba jest martwa oraz False w przeciwnym razie)
```

+ does_like_gov()
  (zwraca True, gdy osoba jest zwolennikiem rządu oraz False w przeciwnym razie)

- get_infection(vac: Vaccine)
  (jeżeli dana jako argument szczepionka nie jest wystarczająco silna to osoba się zaraża, jeżeli osoba nie była zaszczepiona to traktujemy to jako szczepionka o efektywności 0%)

- get_vac(vac: Vaccine)
  (osoba dostaje szczepionkę, która może ją częściowo uodpornić albo zabić)

- meet(other_person: Person, virus: Virus)
  (powoduje spotkanie się dwóch osób, jeżeli jedna z nich jest zakażona to istnieje szansa zakażenia drugiej osoby, istotne są tutaj czynniki siły wirusa)

- die()
  (powoduje zmianę stanu dead danej osoby, osoba taka nie będzie mogła dalej zarażać ani z nikim się spotykać)

2. Vaccine

Klasa Vaccine odpowiada za szczepionkę, która będzie mogła pomóc zakończyć epidemię. Posiada ona następujące atrybuty.

- *name: str (nazwa szczepionki)*
- *price: int (cena jednostkowa szczepionki za jaką rząd może ją kupić, liczy się wartość absolutna wprowadzonej ceny)*
- *virus: Virus (wirus, na który dana szczepionka może dawać odporność)*
- *done: bool (informacja czy szczepionka już jest przygotowana)*
- *progress: int (progres tworzenia szczepionki)*
- *mortality: int 0:100 (opcjonalna wartość szczepionki, która może powodać śmierć osoby zaszczepionej)*
- *efficiency: int 0:100 (opcjonalna wartość skuteczności szczepionki w %)*

Klasa Vaccine posiada tylko jedną metodę `-make_progress(new_progress: int)`, która odpowiada za inkrementacje procesu tworzenia szczepionki.

3. Virus

Następną klasą jest klasa Virus odpowiedzialna za wirusa, który zaraża i zabija zarażonych. Posiada ona następujące atrybuty:

- *name: str (nazwa wirusa)*
- *mortality: int 0:10 (śmiertelność wirusa)*
- *contagiousness: int 1:10 (dotkliwość wirusa)*
- *strength: int 1:3 (siła wirusa tj. odporność na szczepionkę)*

Wirus posiada następujące metody:

- change_mort(new_mort: int)
  (metoda pozwala ustalić nową śmiertelność)

- change_cont(new_cont: int)
  (metoda pozwala ustalić nową dotkliwość)

- change_strg(new_strg: int)
  (metoda pozwala ustalić nową siłę wirusa)

- mutate(new_mort: int, new_cont: int, new_strg: int)
  (metoda umożliwiająca dynamiczną mutację wirusa)

4. Government

Następująca klasa odpowiada za reprezentacje abstrakcyjnego rządu w stworzonej populacji. Rząd decyduje o poziomie restrykcji oraz kupuje szczepionki. Jego atrybuty są następujące:

- *name: str (nazwa rządu)*
- *level_of_restrictions: int 1:10 (poziom restrykcji rządu)*
- *budget: int (dzienny budżet, za który rząd może kupić szczepionkę)*
- *know_about_virus: bool [kav](informacja czy rząd wykrył już wirusa)*
- *vaccine_is_ready: bool [vir](informacja czy szczepionka już jest dostępna dla społeczeństwa)*

Government posiada następujące metody:

```
+ change_restrictions(new_level: int)
  (zmienia poziom restrykcji)

- get_know_about_virus()
  (metoda zmieniająca status rządu na temat wiedzy o istnieniu wirusa)
```

5. Day

Klasa Day odpowiada za reprezentowanie poszczególnego abstrakcyjnego dnia (można utożsamiać z dowolnym okresem czasu) i jest zarazem zbiornikiem danych dla danej populacji. Posiada on następujące atrybuty:

- *day: int (liczbowy idendyfikator każdego dnia)*
- *healthy: list (lista przechowująca wszystkie zdrowe osoby)*
- *infected: list (lista przechowująca wszystkie zakażone osoby)*
- *dead: list (lista przechowująca wszystkie martwe osoby)*

Day posiada następujące metody:

```
+ get_healthy()
  (zwraca listę osób zdrowych)

+ get_infected()
  (zwraca listę osób zakażonych)

+ get_dead()
  (zwraca listę osób martwych)

+ get_numbers()
  (zwraca krotkę liczb, które odpowiadają odpowiednio za ilość osób zdrowych, zakażonych, martwych)

- set_lists(Population: Population)
  (ustala 3 pierwsze atrybuty klasy w zależności od danej argumentem populacji)

- null_lists()
  (metoda czyszcząca listy w pierwszych 3 atrybutach, działanie odwrotne do set_lists())
```

Day posłuży również w symulacji do zapisywania odpowiednich danych oraz tworzenia z nich wykresów.

6. Population

Główna klasa tworząca całą populację i społeczność. Zawiera w sobie listę ludzi oraz rząd nimi przewodzący. Posiada również dodatkowe parametry, które zosstaną rozbudowane w przyszłości. Atrybuty tej klasy to:

- *name: str (nazwa społeczeństwa)*
- *people: list (lista przechowująca obiekty typu Person)*
- *government: Government (rząd w danej populacji)*
- *attitude_to_gov: float 1:10 (poziom nastawienia do rządu wyliczany na podstawie ilości zwolenników)*
- *economy_level: float 1:10 (poziom ekonomiczny społeczeństwa, atrybut z planem rozbudowania go w przyszłości)*

Istotnym aspektem metod klasy Population jest możliwość stworzenia populacji z gotowym społeczeństwem bez wcześniejszego tworzenia każdej osoby 'od ręki'. Poniżej znajdują się metod naszej klasy:

+ `set_population(num_of_people: int, gov: Government, att_to_gov: float, pot_eco_lev: float, hand_min: int, jobs: tuple, distribution: tuple)`
  `(funkcja tworząca populację o wielkości num_of_people, rządzie gov, wyliczanym podejściu do rządu i poziomie ekonomicznym, dodatkowym argumentem jest hand_min, która ustala minimalny poziom mycia rąk w społeczeństwie). Dodatkowo możliwym jest ustawienie jakie zawody będą pełnili ludzie w naszym społeczeństwie za pomocą jobs. Argument distribution odpowiada za rozkład danych zawodów wśród ludzi. Domyślnie mamy 4 zawody bezdomny, lekarz, słodziej, niewolnik z rozkładem jednostajnym.)`

+ `set_infected(num_of_infections: int)`
  `(ręczne zakażenie danej ilości ludzi w społeczeństwie)`

- `get_alive()`
  `(zwraca listę ludzi, którzy nie są martwi)`

- `change_attitude(new_attitude: float)`
  `(zmiana podejścia do rządu)`

- `change_economy(new_level: float)`
  `(zmiana aktualnego poziomu ekonomicznego)`

- `list_of_vaccinted()`
  `(zwraca listę osób zaszczepionych)`

- `list_of_not_vaccinted()`
  `(zwraca listę osób niezaszczepionych)`

- `cure_people(num_of_cured: int, vac: Vaccine)`
  `(aplikuje danej ilości osób podaną szczepionkę)`

7. *Events*

Moduł *Events* zawiera w sobie interfejs *Events* oraz jego różne implementacje. Wydarzenia odpowiadają za ustalenie odpowiednich parametrów danego dnia. Na tę chwilę implementacje interfejsu *Events* są niezmienialne. Każda z nich posiada atrybut *name: str* odpowiadający za jego nazwę i taką samą metodę `-set_parameters_of_the_day(pop: population)`, która jak już wspomnieliśmy ustala parametry takich danych jak ilość spotkań wśród społeczeństwa, dodatkowy budżet na szczepionki etc. Domyślnym trybem dnia jest NormalDay, lecz w przypadku, gdy rząd dowie się o wirusie mogą wydarzyć się następujące rzeczy.

NormalDay odpowiada za normalny przebieg dnia.

ElvisConcert powoduje zwiększoną ilość spotkań ludzi, którzy udali się na koncert Elvisa. Do tego umiera 5 osób o zawodzie 'slave'.

RobotsKillPeopleOutside jest działaniem rządu rozkazującym abstracyjnym robotom zabicie wszystkich osób, które wyjdą na ulice. Ilość spotkań tego dnia jest znacząco ograniczona.

KillMedics sprawia, że z powodu wirusa umiera więcej lekarzy.

NationalVaccination daje rządowi większy budżet na zakup szczepionek, lecz jest obarczony większą ilością spotkań ludzi w kolejce po szczepionkę.

8. Simulation

Główny moduł odpowiedzialny za tworzenie symulacji. Symulacja jako atrybuty przyjmuje następujące obiekty:

- *populations: list (lista populacji, a których przeprowadzimy symulację epidemii)*
- *days: list (lista dni, przez które przeprowadzimy epidemię)*
- *events: dict (słownik, który jako klucze posiadać będzie elementy Events, a jako wartości przyjmowane będę krotki reprezentujące 'przedział' obustronnie domknięty)*[1]

Dla takich atrybutów dane są następujące metody:

```
_ set_zero_day(Population: Population)
  (statyczna metoda tworząca dzień 0 dla danej populacji na podstawie jej zawartości)

_ save_data(diary: str)
  (statyczna metoda zapisująca przebieg epidemii, dla każdej populacji w pliku 'symulacja.txt'
  w folderze z modułami)

+ plot_infections()
  (metoda tworząca wykres przebiegu zakażeń, wyzdrowień i śmierci)

+ simulation_start(virus: Virus, vac: Vaccine)
  (główna metoda odpowiedzialna za symulację, jej działanie zostanie opisane w dodatkowej podsekcji)
```

8.1 simulation_start

Metoda ta tworzy dziennik, który później za pomocą `save_data()` zapisuje przebieg epidemii w pliku `symulacja.txt`. Następnie w pętli dla każdej populacji ustalamy dzień zerowy i dzienny budżet rządu na szczepionki. Następnie dla każdego dnia z listy wykonujemy następujące czynności:

1. Czyścimy listy Day, w którym aktualnie jesteśmy.
2. Ustalamy dla rządu budżet NA TEN DZIEŃ.
3. Tworzymy listę ludzi żywych.
4. W przypadku, gdy rząd nie wie o wirusie ustalamy, aby dzisiejszym wydarzeniem był NormalDay, w przeciwnym razie możliwe są tez inne wydarzenia.
5. Zapisujemy wszystkie parametry, które daje nam dane wydarzenie.
6. Ustalamy ilość spotkań, dodajemy dodatkowy budżet na szczepionki i kupuje je.
7. Następnie poszczególnych zdarzeń dzieją się różne rzeczy opisane w rozdziale 7. Ludzie się spotykają, zabiją na koncercie Elvisa etc.
8. Wirus zabija zależnie od swojej śmiertelności.
9. Ustalamy liczbę medyków, gdyż ich brak skutkuje niemożnością aplikowania szczepionek.
10. Później sprawdzamy czy rząd już wie o wirusie, jeśli nie to możliwe, że powinien. Jeśli już wie to zależnie czy mamy już szczepionkę czy nie dajemy ją danej liczbie osób lub pracujemy nad szczepionką.
11. Na koniec istnieje niewielka szansa, że wirus zmutuje.
12. Cały proces powtarza się dla każdego dnia z listy days oraz dla każdej populacji z listy Populations.
13. Wszystkie dane zostaną zapisane w pliku `symulacja.txt`, a z każdą kolejną populacją zobaczymy wykresy obrazujące zakażenia, wyzdrowienia i śmierci.

[1] Przedział taki odpowiada za procentowe możliwe wystąpienie wydarzenia w przypadku, gdy rząd już wie o wirusie.

9. UnitTest

Moduł, w którym znajdują się testy jednostkowe.

Presentation

Moduł pomocniczy, w którym znajdują się gotowe dane do rozpoczęcia symulacji, które można śmiało modyfikować i cieszyć się epidemią w naszych populacjach.

Diagram UML

- Person wykorzystuje Vaccine i Virus w swoich metodach, stąd mamy zależność.
- Vaccine posiada Virus jako jeden z atrybutów, stąd asocjacja. Szczepionka może działać tylko na jednego wirusa.
- Symulacja wykorzystuje konkretnego wirusa i konkretną szczepionkę stąd zależność.
- Populacja składa się z ludzi, a oni są od niej uzależnieni. Wiele osób może przypadać do jednej populacji, lecz jedna osoba może być tylko w jednej populacji.
- Zachodzi asocjacja pomiędzy populacją i rządem, gdyż rząd jest jednym z atrybutów populacji.
- Day oraz *Events* wykorzystują w działaniu Population stąd zależność.
- Symulacja sklada się z populacji, dni oraz wydarzeń, lecz nie jest odpowiedzialna za ich tworzenie lub usuwanie stąd agregacja częściowa. Symulacja może mieć 5 wydarzeń, dowolną większą niż 1 ilość populacji i dowolną ilość dni.
- Jak już wspomnielismy w sekcji 7 *Events* to interfejs, a NormalDay, ... , KillMedics to jego implementacje.

Ciekawostki

- Jeśli wirus losowo bardzo często mutuje to znacząco zmienia to przebieg epidemii.
- Symulacja lubi zabić wiele osób praktycznie jedynie za pomocą robotów.
- Implementacja Day nie jest optymalna. Przechowuje w swoich listach całe obiekty Person zamiast samych identyfikatorów.
- Interfejs *Events* i wszystkie jego implementacje mogłyby zostać napisane jako rozbudowane funkcje, lecz daje to większą przejrzystość i nadaje znaczenia wydarzeniom. Pomóc to może również w przypadku rozbudowy symulacji w przyszłości.
- *W rozwoju*