SIMULATING HOSPITAL ACQUIRED INFECTION TRANSMISSION WITH THE AGENTS ASSEMBLY ECOSYSTEM

Przemysław Hołda
Kajetan Rachwał
Warsaw University of Technology
00-662, Warsaw, Poland
Email: przemyslaw.holda.stud@pw.edu.pl
Email: kajetan.rachwal.stud@pw.edu.pl

Wiktoria Głowniak
Marcin Rojek
Medical University of Silesia
40-514, Katowice, Poland
Email: s82790@365.sum.edu.pl
Email: s83107@365.sum.edu.pl

KEYWORDS

Agent-based simulations; Hospital acquired infections; Infection transmission modeling

ABSTRACT

While agent-based models and simulations materialize in multiple areas, the existing simulation-focused agent platforms require in-depth programming knowledge, or are overly simplistic. In this context, the Agents Assembly (AASM) domain-specific language and platform have been recently proposed. The AASM ecosystem is highly scalable, as the software stack allows its straightforward deployment on multiple networked computers (physical or virtual machines). The domain-specific language has been designed to capture key concepts, needed to run agent-based simulations, while hiding their technical aspects. It can be thus stipulated that the AASM may provide a mid-way point between agent researchers and domain specialists. In this context, the aim of this contribution is twofold. First, to outline the reasoning behind, and details of, improvements introduced to the AASM since the original release. Second, show how the AASM can facilitate cooperation with researchers with a medical background. Here, the simulation models of the behavior of the *Clostridium difficile* bacteria in a hospital environment have been jointly conceptualized and experimentally explored. The developed model was focused on capturing features that reduce the spread of the bacteria. While the proposed scenarios are relatively simple, they illustrate the ease with which the AASM can be used to capture real-life phenomena.

INTRODUCTION

In recent years, there has been a growing interest in agent-based simulations in a variety of fields [Madsen and Pilditch, 2018] [Gatti and Desiderio, 2015] [Castro et al., 2021]. Some agent-based simulation tools, like HASHKAT [Raudeliūnienė et al., 2018] or AgentSheets [AgentCubes, 2020], enable users without programming knowledge to utilize Multi-Agent Systems (MAS) when designing their simulations. However, such platforms have limited domain of applicability, e.g., HASHKAT can only be used to simulate "Twitter-like networks", while AgentSheets was designed to be a tool for teaching children programming in an interactive way. Other tools are more robust, e.g., Mason [Luke et al., 2005], Swarm [Iba, 2013], and Repast [North et al., 2013], with the latter focused on facilitating the creation of high-performance agent-based models. However, their use requires technical know-how, which complicates work on model development, when multidisciplinary teams are to work together. Specifically, domain experts may have problems "understanding the code", while agent modelers may not be able to easily model/represent domain concepts.

Here, worth mentioning is NetLogo [Tisue and Wilensky, 2004], which tries to find a compromise between the two extremes. It provides users with a mature tool for running complex simulations, which includes live visualization, changing parameters during runtime, and a simple language for defining the model. However, in NetLogo, agents are represented as "turtles", the environment as "patches" (squares in 2D or cubes in 3D), and relations between agents as "links" (agents representing connections with properties) which can limit the users' ability to model the environment in other ways. This is directly related to the ideas originating from the Logo programming language and the roots of the NetLogo in teaching programming to school kids. NetLogo has been successfully used for deploying complex models in various fields [Gatti and Desiderio, 2015] [Isha et al., 2021]. However, developing models in NetLogo requires the use of concepts that bring a somewhat myopic perspective on the capabilities of agent-based simulations.

To overcome the mentioned issues, the Agents Assembly has been proposed [Hołda et al., 2022]. The developed ecosystem comprises a Domain-Specific Language (DSL) – Agents Assembly (AASM) – coupled with the modular runtime architecture. The AASM has been designed to allow users without advanced programming skills to develop multi-agent simulations with desired behavior and structure. This has been accomplished by expressing high-level MAS abstractions (e.g., agents, behaviors) as language keywords. As a result, the implementation details are hidden behind simple instructions, which are generated through a GUI.

The AASM code is translated into a realization running on the Smart Python Agent Development Environment (SPADE) [Palanca et al., 2020]. The choice of SPADE was based on the fact that, at the time of AASM development, SPADE was the most robust, general-purpose agent platform, which was regularly updated (see, for instance, [Pal et al., 2020]). After translation to SPADE, the obtained code can be distributed through a Docker Swarm stack. As a result, simulations can be run on multiple networked machines (physical or virtual). Since the system runs within multiple Dockerized instances of SPADE, available resources are automatically managed and can be easily scaled (by adding Docker

Communications of the ECMS, Volume 37, Issue 1,
Proceedings, ©ECMS Enrico Vicario, Romeo Bandinelli,
Virginia Fani, Michele Mastroianni (Editors) 2023
ISBN: 978-3-937436-80-7/978-3-937436-79-1 (CD) ISSN 2522-2414

instances). This also means that achieving scalability does not require advanced knowledge of computer networks.

The original Agents Assembly has been described in [Hołda et al., 2022] and tested as a part of a thesis project [Hołda and Rachwał, 2022]. Further testing has been done by students who utilized the system for class projects at the Warsaw University of Technology. From these experiences, feedback has been gathered to identify the system's shortcomings. A description of the gathered feedback and how it was used to develop the second release of the AASM is one of the goals of this contribution.

As noted above, one of the overarching goals of the development of AASM was to deliver a tool that would be "understandable enough" to non-specialists to support the joint development of agent models and simulations. To establish if AASM can deliver the right level of conceptual granularity, collaboration with a team interested in modeling medical phenomena (MPT) was established. As a result, an attempt to model the spread of *C. difficile* bacteria has been made. Here the project loop consisted of MPT describing processes happening in the hospital, the agent team (AT) formulating them in terms of AASM constructs, and presenting them to the MPT. The MPT reflected on the presented code and suggested improvements that were realized by the AT. In this way, the AASM became the "place" where the two teams collaborated, while the developed was used to functionalize the developed simulation.

The programming environment constitutes an innovative element in the conducted research, whereas the agent-based approach to simulating *C. difficile* is relatively well-known [Stephenson et al., 2020] [Barker et al., 2020]. While the final simulation is relatively simple, at this stage, the goal was not to develop a realistic simulation. Instead, the aim of the work was to initially validate the claim that the second release of AASM is closer to developing a collaborative agent simulation development tool. As will be discussed, the perspectives are very positive, indeed. Moreover, lessons learned and future directions for AASM improvements will be summarized.

AGENTS ASSEMBLY ECOSYSTEM

The Agents Assembly ecosystem enables running simulations defined using AASM. It consists of a complete solution providing the features for the end user, such as tools for managing simulations, visualization of the simulation state, or tools for data analysis. The large-scale agent simulations can be run in the system based on a distributed microservice architecture that can be set up to run on multiple networked machines. The ability to deploy the solution on a cluster and the scalable behavior is achieved mainly due to Docker and Docker Swarm usage. Utility scripts for deployment and scaling are also provided, making the setup straightforward.

The AASM language provides the user with the ability to describe multi-agent systems in simple terms. Specifically, it features three top-level instruction environments – *Agent*, *Graph*, and *Message*. Each instruction environment is an enclosing scope in which the user can specify implementation details for the simulation elements. In particular, the *Agent* environment uses *Parameters* to describe the internal state of an Agent and to initialize it. Next, *Behaviours* consist of *Actions*, which allow granularization of the description of the simulation logic. Finally, *Actions* are described using *Instructions*, such as control flow modifiers, arithmetics, list operations, and more. The complete language definition can be accessed at `agents-assembly.com`.

The system has been initially tested by students of the Warsaw University of Technology for the development of a traffic simulation involving a taxi service. Throughout their work, they have provided the authors of the system with feedback, which has been used to improve the systems functioning.

The primary point of feedback was that the scope mechanism of AASM is too restrictive. In the original release, the language did not provide any way to define parameters that could be utilized in the definitions of all agents. In order to address this, selected meta-programming features have been added to the new release. These are `const` and `makro` definitions, which provide functionality similar to the `#define` directive in the C programming language. They can be used to define constants and code snippets, respectively, which can be used in all environment scopes.

The second addition to the language based on the feedback from the simulation was the extension of the graph-generating capabilities. In addition to the initially available parameterized random graphs, two more methods have been introduced. The user is now able to define graphs of particular structures using the `matrix` generation algorithm or utilize the Barabási-Albert model [Barabási and Albert, 1999]. In the first case, the user explicitly defines every connection between agents in the simulation. In the language, this is achieved by utilizing an adjacency matrix representation of the graph. However, that can be a difficult concept for a non-technical user. In order to account for that, a special UI has been developed to allow users to graphically define the structure as seen in Figure 1. The user can add and remove nodes of a selected type by clicking and connecting them with edges by selecting a node and dragging a connection to another one.

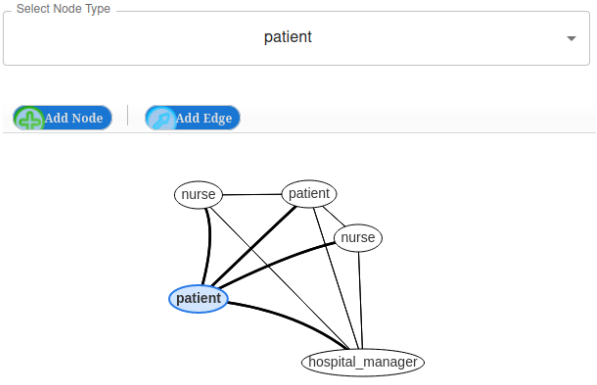

Fig. 1: Example of Graph Structure Definition using Generating GUI

In order to further simplify the use of these high-level abstractions, in the current release of AASM, a dedicated code generator has been developed. It utilizes Blockly [Pasternak et al., 2017] – a library designed for enabling block-based visual programming. Specifically, required blocks have been implemented to accommodate language instructions. An

example of an agent *Action*, represented using the code generating tool, can be seen in Figure 2. The code generated from that block definition has been depicted in Listing 1. As can be seen, while the AASM is somewhat reminiscent of assembly code, the generating interface is relatively easily readable. Moreover, since users can interact with the code by dragging and dropping dedicated blocks, defining simulations should be relatively straightforward.

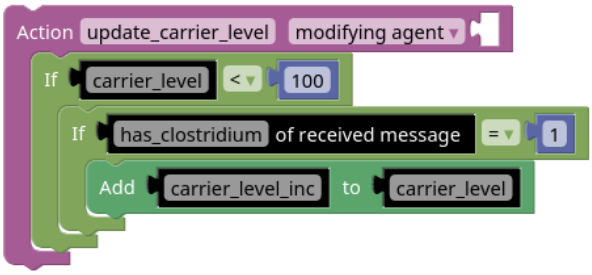

Fig. 2: Action Implemented using the Code Generating Interface.

```
ACTION update_carrier_level, modify_self
  ILT carrier_level, 100
    IEQ RCV.has_clostridium, 1
      ADD carrier_level, carrier_level_inc
    EBLOCK
  EBLOCK
EACTION
```

Listing 1: AASM Code Generated from Blocks Represented in Figure 2.

In comparison to the code generator presented in the previous release, the Blockly-based one offers two significant advantages. The first one is that it provides a way to visually represent the scope – a feature unavailable in the previous text-based generator, requested in feedback. That can vastly help users without programming experience understand this inherently abstract concept. The second advantage is its ability to generate multi-instruction snippets using a single block. An example of that can be seen by comparing the block in Figure 2 with Listing 1. In AASM, every conditional statement (`ILT`, `IEQ` instructions in the figure) has to have a corresponding `EBLOCK` to signal the end of the conditional block. In the generator, however, this implementation detail is hidden from the user, as `EBLOCK` statements are implicitly added when using a conditional block. In general, there is no limit to the amount of AASM instructions that can be generated using a single GUI block, as AASM has been designed with ease of generation as one of the primary goals. Other examples of implemented compound instruction blocks include declaring a variable and setting it to a parameterized random value (2 AASM instructions) or adding an element to the list only if it does not exist in it already (3 AASM instructions). The generator can be easily extended to include more compound instruction blocks.

Previously [Hołda et al., 2022], the ecosystem was set up to run on 15 physical computers with a total of 120 agent containers (each agent container can run multiple agents). However, while the distributed architecture can be successfully utilized in a computing cluster environment, it can be too resource intensive for a single machine – it consists of more than 20 microservice applications (that can be scaled). This observation results from the feedback gathered from the group of Warsaw University of Technology students. In this context, an alternative lightweight solution that gives users the essential features to run and test simulations on a single device has been prepared.

To verify the proposed solution, the authors ran experiments that measured the lightweight environment's improvements in performance on a single machine – the most common setting during development. The authors prepared a simulation with a single type of agent to run the experiments. Each agent connects with some other agents. Later, agents send messages containing numerical values to each other. The experiments were run on a single machine with a 16-core CPU and 32 GB of RAM. The performance of the distributed environment and the lightweight environment was compared. Initially, the RAM usage of empty (without agents) environments was measured. Then, the number of agents was scaled incrementally from 100 to 10000. The results are presented in Figure 3 – the dashed line represents the distributed architecture, and the continuous line the new, lightweight solution. One can observe that with the growth of the number of agents, the amount of used RAM increased monotonically. However, the initial cost of starting the empty environment is much lower in the case of the lightweight environment (570 MiB) compared to the distributed environment (5269 MiB) – the difference is almost 10-fold. Also, the distributed environment includes more services varying in resource usage over time. Hence, its line in Figure 3 is more erratic than the one representing the lightweight environment's performance.

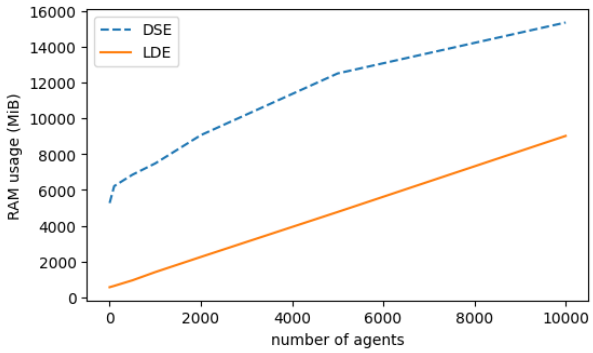

Fig. 3: RAM Usage Comparison

MEDICAL SIMULATION

As noted, the developed ecosystem should facilitate cooperation between experts with domain knowledge and agent system specialists. In this case, experts should focus on accurately describing what is to be simulated and verifying the results. On the other hand, agent systems specialists should be responsible for setting up the system and helping the experts translate their concepts into the "world of agents". Moreover, the Agents Assembly should be the "place" where the two sides can work jointly and cooperate. To establish if the current version of the developed ecosystem is a step

in the right direction, a team from the Medical University of Silesia was invited to jointly develop a simulation of *C. difficile* bacteria spreading in a hospital. The authors will now report how the simulation was developed and what lessons have been learned.

Clostridium difficile bacteria

What follows is the summary of key facts concerning the medical phenomenon that has been modeled. The *Clostridium difficile* is a Gram-positive anaerobic bacteria capable of producing spores. Its spores are widely distributed in the natural environment and can be found in soil and open water reservoirs, among others [Janezic et al., 2016]. Infection most commonly occurs through the ingestion of spores excreted by infected individuals in feces. This route of infection, combined with resistance to alcohol-based disinfectants and requiring hygienic care for symptomatic patients, poses fundamental difficulties in infection control [Mehlich et al., 2015]. *C. difficile* hospital-acquired infections are a growing problem in healthcare, particularly in conjunction with antibiotic therapy. In particular, therapy with clindamycin, cephalosporins, penicillin, and fluoroquinolones has been identified as a risk factor [Kelly, 2012].

In 90% of cases, the infection appears within the first two months of antibiotic therapy and is associated with an increase in the share of anaerobic bacteria in the intestinal microbiota. [Martirosian et al., 2018]. It is characterized by intestinal colonization, with high virulence. [Mehlich et al., 2015]. The main symptoms include diarrhea and toxic colonic distension, while no other cause is easily identified. Despite this, some patients or personnel remain asymptomatic despite the infection [Badurek et al., 2016]. It has been established that in healthy adult populations, *C. difficile* colonizes the gastrointestinal tract in approximately 3% of adult individuals and even up to 80% of healthy infants but does not cause any disease symptoms. However, no data suggest that routine screening of patients and staff for latent infection is justified [Martirosian et al., 2018].

Relatively few simulations based on agent-based programming have been developed so far, and the ones published so far have focused primarily on the interaction of patients and staff with the environment and attempts to develop the most effective model of aseptic behavior [Barker et al., 2017].

Difficulties arising from attempts to stimulate the hospital environment in silico deserve attention. Variables impossible to measure in a simple way, such as the absolute probability of an event, are problematic in designing simulations – an example here may be the problem of calculating the infection in the vicinity of two patients, one of whom is an asymptomatic carrier, or the efficiency of cleaning the room. In such a situation, it becomes necessary to simplify or extrapolate such parameters. In order to maintain the values of the work despite such simplifications, the authors usually present the results of research in the form of a function depending on the problematic parameters [Bintz et al., 2016].

Another way to run a simulation with incomplete data is to interpret its results relative to an alternative scenario. This type of approach significantly reduces the number of simulations necessary to design and analyze but gives only approximate results depending on the assumptions made. This is the approach we used in our experiment.

Simulation development process

The initial stages of planning and problem analysis have been conducted jointly by the MPT and AT members. The MPT has been introduced to the high-level AASM abstractions – Messages, Agents, and their Behaviours. Members of the AT have been provided with insights regarding the functioning of a hospital unit and *C. difficile* infections.

After the initial knowledge exchange, the MPT has designed an outline of a MAS, consisting of Agents that have to be present in the simulation, actions they should perform, and a communication schema to be employed by the Agents. The AT has developed an initial implementation of the simulation in AASM, providing a foundation for joint development. From that point onward, the development proceeded in the above-outlined loop consisting of joint development sessions and evaluation of intermediate results. The MPT considered particular implementations of *Actions* and provided feedback, suggesting needed improvements. Some ideas were written by the AT after consultation, and some were implemented by the MPT. The latter has been enabled by the code-generating user interface. The process has proven sufficiently intuitive so that even without prior programming knowledge, multiple solutions implemented directly by the MPT have been utilized in final simulations.

Details of developed agent model

The simulation consisted of four types of agents – *nurses*, *patients*, *hospital manager*, and *day manager*. The day manager's purpose was to manage the in-simulation time, i.e., informing all other agents about the beginning of new days. The hospital manager kept track of all patients currently hospitalized, as well as which of them were isolated. Based on that data, it assigned patient rotations to the nurses.

The nurses were responsible for "testing" the patients. The testing was performed in the rotation order provided by the hospital manager. Each day a nurse was provided with two unique rotations. Every rotation had a length of at most eight patients (sometimes, the occupancy of the hospital was insufficient to fill all rotations). During the testing, the nurse can become a carrier by contact with a patient infected with *C. difficile*. To model this, each nurse has a parameter reflecting their "carrier level", which increases upon repeated contact with *C. difficile* (via multiple patients) during the day. After the test, a nurse has a 39% to wash hands, which decreases the carrier level by a random percentage between 40% and 60%. The percentages were based on the available literature [Lambe et al., 2019]. If the test results are positive (*C. difficile* detected), the nurse sends the patient to isolation and moves to the next patient. Each nurse performs two rotations daily without washing hands in between, and at the end of the day, they reset their carrier level.

Two variations of setting the rotation order have been implemented and compared. In the first one, nurses were assigned eight patients in a random order to visit. In the second one, the nurses were assigned eight random patients divided into isolated ones and not. They visited the isolated patients at the end of the rotation to reduce the potential spread of *C.*

difficile.

The patients represent the general population. Half of them begin the simulation in the hospital, where they are vulnerable to *C. difficile* infections. Each patient has a 50% chance of beginning the simulation already infected. In such a case, they begin the simulation already at a certain point in the infection – a random number of days drawn from a uniform distribution between 0 and 7. If the random number of days exceeds the hidden period (2 days), the patient starts in isolation. While hospitalized, each day, there is a 20% chance of getting better and leaving the hospital. These percentages have been chosen experimentally to achieve a sufficient occupancy rate of the hospital for the simulation to yield meaningful results.

During each day, the patients not in isolation interact with each other. This is the first vector of infection of *C. difficile* modeled – upon interaction with an infected patient, there is a chance of acquiring the bacteria. To model this, each patient has a parameter called "personal hygiene" randomly initialized for each patient with a value drawn from a uniform distribution between 0 and 100. Upon contact with an infected patient, a test is performed by drawing another number from the same distribution – $transmission\_chance$. Patient's hygiene is then scaled by 0.2 and subtracted from $transmission\_chance$. If that number is greater than a $threshold$ (parameter set to 95), the patient becomes infected. The other vector of infection is the tests done by the nurse agents. In this case, the $transmission\_chance$, patient's hygiene, and the nurse's carrier level according to the following equation: $C = transmission\_chance - 0.2 * hygiene + carrier\_level$. Then if C is greater than the $threshold$, the patient becomes infected. The above calculation is only carried out if the nurse's carrier level is greater than 0. Isolated patients do not move around, which means they only interact with nurses. However, before the patient can be moved to isolation, they have to develop symptoms. *C. difficile* has a hidden period during which patients are asymptomatic but still can spread the virus [Badurek et al., 2016]. For modeling purposes, it has been assumed that only symptomatic patients can test positive. In addition, 2% of the population is genetically asymptomatic [Ulatowska et al., 2017] – these patients will never be moved to isolation. The duration of *C. difficile* infection varies between 7 and 14 days, and after the infection passes, the patient leaves the hospital. Each day a random number has been drawn from a normal distribution with a mean of 10 and standard deviation of 1 and clipped to the desired range. That number has been compared to the patient's infection duration, and if the duration has been greater, the patient was released from the hospital.

Experiments

The experiments were performed on 12 nurse agents and 60 patient agents, one hospital manager, and one day manager. The expected result was that imposing the restriction on the rotation order would decrease the number of patients with *C. difficile*. The simulations were performed five times utilizing different random seeds, which allowed to average their results and meaningfully compare the two settings.

Experimental results

After the design and development phases were over, the AT preprocessed the data retrieved from the simulations. Thanks to the MPT's knowledge of implementation details, they were able to assist in data processing, suggesting statistics of particular interest. The final analysis of the results has been conducted jointly. What follows is the discussion of results obtained from running the simulations, with different ordering of visiting patients by nurses in the hospital. In subsequent figures, the dashed line represents the setting where the order of patients in the rotation is random (S_{rand}), while the continuous line represents the simulation results when the nurse visits isolated patients at the end of the rotation (S_{isol}). In the following considerations, the μ symbol represents the mean, and the σ represents the standard deviation.

First, the results regarding the population size with *C. difficile* are displayed (Figure 4). In the S_{rand} configuration, the average number of people with the bacteria on a given day is $\mu_{rand} = 25.54$ ($\sigma_{rand} = 2.06$). The S_{isol} configuration yields the average result of $\mu_{isol} = 15.87$ ($\sigma_{isol} = 3.05$). The trend is visible in the figure – the S_{isol} configuration outcomes in lower bacteria spread inside the hospital. The maximum values were reached after the initialization of the simulations in both cases – $\max_{rand} = 30.80$ and $\max_{isol} = 24.80$, respectively.

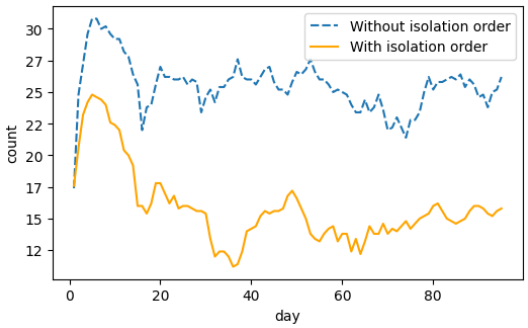

Fig. 4: Population with *C. difficile*

As only some people in the simulation are inside the hospital, the size of the hospitalized population (not necessarily with the bacteria) in both settings has been considered (Figure 5). The trends representing the occupancy of the hospital presented in the accompanying figure are comparable. The configurations average in $\mu_{rand} = 47.56$ ($\sigma_{rand} = 2.11$) and $\mu_{isol} = 46.20$ ($\sigma_{isol} = 1.62$).

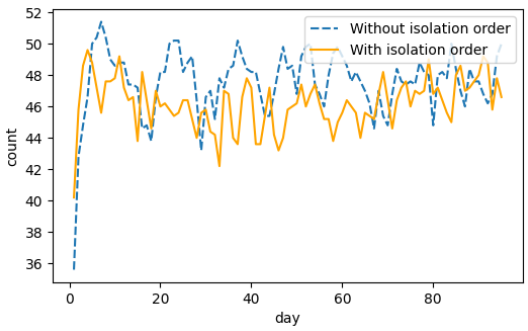

Fig. 5: Population in Hospital

Next, the metric describing the ratio of the population si-

ze with *C. difficile* to the hospitalized population size was examined (Figure 6). The S_{rand} case has the average ratio of $\mu_{rand} = 0.53$ ($\sigma_{rand} = 0.03$), and the S_{isol} case averages in $\mu_{isol} = 0.34$ ($\sigma_{isol} = 0.06$). The obtained results are a consequence of the previous two measurements (because of the definition of the analyzed ratio). Thus, one can observe the similarities in the population size with *C. difficile* and this metric (as the number of hospitalized people remain similar in both configurations).

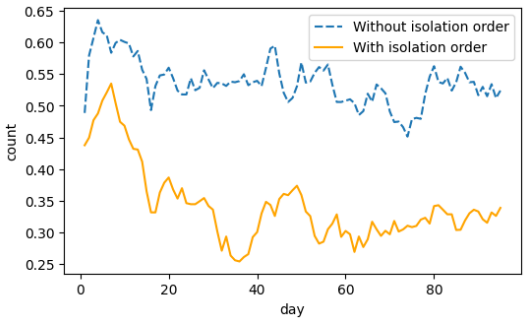

Fig. 6: Population with *C. difficile* to Population in Hospital Ratio

Consequently, the number of isolated hospital patients was investigated (Figure 7). This number oscillates within specific ranges. In the case of the S_{isol}, the oscillations attenuate faster than in the case of the S_{rand} setting. Keeping the rotation order (by nurses) of first visiting non-isolated patients resulted in lower occupancy of the isolation rooms, mainly due to the low spread of the bacteria inside the hospital. The average count of isolated patients in the S_{rand} case was $\mu_{rand} = 16.88$ ($\sigma_{rand} = 2.67$), and in the S_{isol} case, $\mu_{isol} = 9.03$ ($\sigma_{isol} = 2.80$).

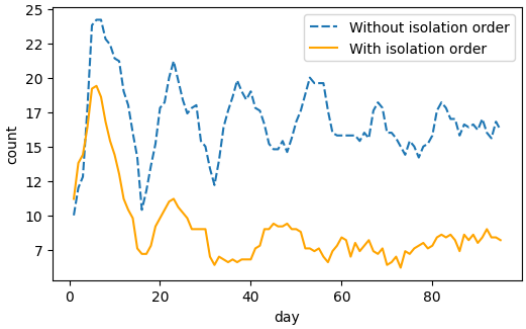

Fig. 7: Isolated Patients

Finally, the average carrier level of all nurses was compared (Figure 8). For the S_{rand} configuration the average carrier level was $\mu_{rand} = 28.57$ ($\sigma_{rand} = 2.71$), for the S_{isol} configuration – $\mu_{isol} = 10.05$ ($\sigma_{isol} = 2.87$). In the figure, one can observe that the S_{isol} design resulted in lower values, which is a consequence of the decision that a nurse can increase their carrier level after having contact with a patient with *C. difficile*.

Discussion

The results obtained from the simulation confirm the hypotheses – namely, that utilization of sanitary procedures – isolation of the infected significantly lowers the spread

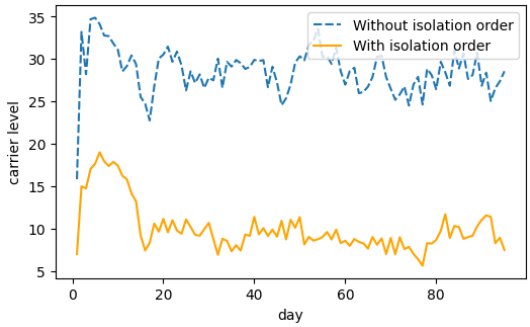

Fig. 8: Average Carrier Level of All Nurses

of bacteria. It is to be noted that the hospital environment modeled in the presented simulation was characterized by significant simplifications, and the results obtained in this context cannot be used to analyze the results in a medical context. That being stated, all expected outcomes have been realized through the simulation procedure. The initial heavy increases of patients infected with *C. difficile* (Figures 4 and 6) correspond to the initial outbreak, where the infected are spreading the bacteria without displaying the symptoms. These amounts begin decreasing in both scenarios after around ten days, which is the expected duration of the *C. difficile* infection. A significant reduction in infections of *C. difficile* has been observed in the S_{isol}, not only in the absolute numbers but also as a proportion of the infected population compared to current hospital occupancy. An interesting pattern can be observed in Figure 7, with the number of isolated patients fluctuating in a seemingly periodic manner (with some variation). The period of these fluctuations corresponds to the duration of *C. difficile*. During the simulation, agents get infected at different moments, which leads to them developing symptoms at different moments. Over time this trend stabilizes, which can be seen in Figure 7 as lowering the fluctuation's amplitude. It can be concluded that the results confirm the recognized rules of asepsis.

CONCLUDING REMARKS

The results obtained from the simulations show that the AASM ecosystem can be used to define simulations in the medical field. Nonetheless, it should be noted that the medical results are of limited generalizability as only stripped-down scenarios were considered for the experiments. The researched topic should be further explored using more complex models. However, while the simplicity of the proposed simulation limits the applicability of the achieved results in the real-world medical context, it was a necessary prerequisite to test how usable the proposed tools are by users without a technical background. In this context, the results of teamwork are very promising, with the members of the MPT being able to actively participate in the model's implementation – allowing for the direct application of their expertise. The usage of Blockly-based code-generating GUI showed that utilizing visual programming techniques can be invaluable in terms of cooperation between technical and non-technical team members. Therefore, the smooth collaboration confirms that the tested ecosystem has the potential to facilitate cooperation with researchers with a medical background and increase their involvement in the development

of simulations.

Future work

The AASM ecosystem has proven itself as a valid tool for making MAS-based simulations more accessible to researchers in other fields. Throughout this project, feedback has been gathered from the members of MPT in terms of any difficulties they have encountered when using the system. It has been pointed out that while certain concepts can be easily expressed and modeled, the currently available abstractions pose certain difficulties – for example, the usage of typed variables can be slightly confusing. Thus a need has been identified for the development of more complex compound instruction blocks with more powerful capabilities. This feedback will be implemented in the form of a more robust block library and released in the near future.

The results achieved in this simulation have also shown the potential of the AASM ecosystem to be utilized in more complex models. Plans are being made for designing these models in the medical and other domains. The planned models will be extended in terms of the complexity of agent behaviors and, more importantly – the scale of simulations. The valuable experience gathered during the collaboration between the two teams will be used in future projects.

ACKNOWLEDGMENT

This work is financially supported by a grant from the Ministry of Science and Higher Education in Poland (grant no. POWR.03.03.00-00-P019/18). The authors also thank professor Maria Ganzha from Warsaw University of Technology and professor Marcin Paprzycki from the Systems Research Institute of the Polish Academy of Sciences for valuable insights and guidance.

REFERENCES

[AgentCubes, 2020] AgentCubes (2020). Main page — agentcubes,. [Online; accessed 19-February-2023].

[Badurek et al., 2016] Badurek, S., Muszytowski, M., Stróżecki, P., and Manitius, J. (2016). Clostridium difficile-associated disease in patients with chronic kidney disease. In *Renal Disease and Transplantation Forum*, volume 9, pages 141–148.

[Barabási and Albert, 1999] Barabási, A.-L. and Albert, R. (1999). Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512.

[Barker et al., 2017] Barker, A., Alagoz, O., and Safdar, N. (2017). Interventions to reduce the incidence of hospital-onset clostridium difficile infection: An agent-based modeling approach to evaluate clinical effectiveness in adult acute care hospitals. *Clinical infectious diseases : an official publication of the Infectious Diseases Society of America*, 66.

[Barker et al., 2020] Barker, A. K., Scaria, E., Alagoz, O., Sethi, A. K., and Safdar, N. (2020). Reducing c. difficile in children: An agent-based modeling approach to evaluate intervention effectiveness. *Infection Control & Hospital Epidemiology*, 41(5):522–530.

[Bintz et al., 2016] Bintz, J., Lenhart, S., and Lanzas, C. (2016). Antimicrobial stewardship and environmental decontamination for the control of clostridium difficile transmission in healthcare settings. *Bulletin of mathematical biology*, 79.

[Castro et al., 2021] Castro, B. M., de Abreu de Melo, Y., Fernanda dos Santos, N., Luiz da Costa Barcellos, A., Choren, R., and Salles, R. M. (2021). Multi-agent simulation model for the evaluation of COVID-19 transmission. *Computers in Biology and Medicine*, 136.

[Gatti and Desiderio, 2015] Gatti, D. and Desiderio, S. (2015). Monetary policy experiments in an agent-based model with financial frictions. *Journal of Economic Interaction and Coordination*, 10(2):265–286.

[Hołda et al., 2022] Hołda, P., Rachwał, K., Sawicki, J., Ganzha, M., and Paprzycki, M. (2022). Agents assembly: Domain specific language for agent simulations. In Dignum, F., Mathieu, P., Corchado, J. M., and De La Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection*, pages 487–492, Cham. Springer International Publishing.

[Hołda and Rachwał, 2022] Hołda, P. and Rachwał, K. (2022). Agent based information flow simulation.

[Iba, 2013] Iba, H. (2013). *Agent-Based Modeling and Simulation with Swarm*.

[Isha et al., 2021] Isha, A., D' Silva, T. C., P M V, S., Chandra, R., and Vijay, V. K. (2021). Stabilization of anaerobic digestion of kitchen wastes using protein-rich additives: Study of process performance, kinetic modelling and energy balance. *Bioresource Technology*, 337:125331.

[Janezic et al., 2016] Janezic, S., Potocnik, M., Zidaric, V., and Rupnik, M. (2016). Highly divergent clostridium difficile strains isolated from the environment. *PLOS ONE*, 11(11):1–12.

[Kelly, 2012] Kelly, C. (2012). Can we identify patients at high risk of recurrent clostridium difficile infection? *Clinical Microbiology and Infection*, 18:21–27.

[Lambe et al., 2019] Lambe, K., Lydon, S., Madden, C., Vellinga, A., Hehir, A., Walsh, M. E., and O'Connor, P. (2019). Hand hygiene compliance in the icu: A systematic review. *Critical Care Medicine*.

[Luke et al., 2005] Luke, S., Cioffi, C., Panait, L., Sullivan, K., and Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81:517–527.

[Madsen and Pilditch, 2018] Madsen, J. K. and Pilditch, T. D. (2018). A method for evaluating cognitively informed micro-targeted campaign strategies: An agent-based model proof of principle. *PLOS ONE*, 13(4):1–14.

[Martirosian et al., 2018] Martirosian, G., Hryniewicz, W., Ozorowski, T., Pawlik, K., and Deptuła, A. (2018). Zakażenia clostridioides (clostridium) difficile: epidemiologia, diagnostyka, terapia, profilaktyka. [Online; accessed 26-February-2023].

[Mehlich et al., 2015] Mehlich, A., Górska, S., Gamian, A., and Myc, A. (2015). Wybrane aspekty zakażeń clostridium difficile. *Advances in Hygiene & Experimental Medicine/Postepy Higieny i Medycyny Doswiadczalnej*, 69.

[North et al., 2013] North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1.

[Pal et al., 2020] Pal, C., Leon, F., Paprzycki, M., and Ganzha, M. (2020). A review of platforms for the development of agent systems. *CoRR*, abs/2007.08961.

[Palanca et al., 2020] Palanca, J., Terrasa, A., Julian, V., and Carrascosa, C. (2020). Spade 3: Supporting the new generation of multi-agent systems. *IEEE Access*, 8:182537–182549.

[Pasternak et al., 2017] Pasternak, E., Fenichel, R., and Marshall, A. N. (2017). Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pages 21–24.

[Raudeliūnienė et al., 2018] Raudeliūnienė, J., Davidavičienė, V., Tvaronavičienė, M., and Jonuška, L. (2018). Evaluation of advertising campaigns on social media networks. *Sustainability*, 10(4).

[Stephenson et al., 2020] Stephenson, B., Lanzas, C., Lenhart, S., Ponce, E., Bintz, J., Dubberke, E. R., and Day, J. (2020). Comparing intervention strategies for reducing clostridioides difficile transmission in acute healthcare settings: an agent-based modeling study. *BMC Infectious Diseases*, 20(1):799.

[Tisue and Wilensky, 2004] Tisue, S. and Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. pages 16–21.

[Ulatowska et al., 2017] Ulatowska, A., Plagens-Rotman, K., Piskorz-Szymendera, M., Włodarczyk, E., Smolarek, N., Miechowicz, I., Jóźwiak, A., and Bączyk, G. (2017). Clostridium difficile – still a problem among the xxi century in case of geriatic patient. *Journal of Medical Science*, 86(4):272–278.

PRZEMYSŁAW HOŁDA is a MSc student in Computer Science at Warsaw University at the Faculty of Mathematics and Information Science. He works at the Systems Research Institute Polish Academy of Sciences. His research interests include distributed systems and artificial intelligence. His email address is przemyslaw.holda.stud@pw.edu.pl

KAJETAN RACHWAŁ is a MSc student in Computer Science at Warsaw University at the Faculty of Mathematics and Information Science. He works at the Systems Research Institute Polish Academy of Sciences. His research interests include distributed systems and artificial intelligence. His email address is kajetan.rachwal.stud@pw.edu.pl

WIKTORIA GŁOWNIAK is a medical student at Medical University of Silesia. Her research interests include Ophthalmology. Her email address is s82790@365.sum.edu.pl

MARCIN ROJEK is a medical student at Medical University of Silesia. He works at Research and Development Center INVESTEKO S.A. His research interests include artificial intelligence in medicine. His email address is s83107@365.sum.edu.pl