Towards VEsNA, a Framework for Managing Virtual Environments via Natural Language Agents

Andrea Gatti and Viviana Mascardi
Department of Computer Science, Bioengineering, Robotics and Systems Engineering (DIBRIS)
University of Genova, Italy
s4496922@studenti.unige.it, viviana.mascardi@unige.it

Automating a factory where robots are involved is neither trivial nor cheap. Engineering the factory automation process in such a way that return of interest is maximized and risk for workers and equipment is minimized, is hence of paramount importance. Simulation can be a game changer in this scenario but requires advanced programming skills that domain experts and industrial designers might not have.

In this paper we present the preliminary design and implementation of a general-purpose framework for creating and exploiting Virtual Environments via Natural language Agents (VEsNA). VEsNA takes advantage of agent-based technologies and natural language processing to enhance the design of virtual environments. The natural language input provided to VEsNA is understood by a chatbot and passed to a cognitive intelligent agent that implements the logic behind displacing objects in the virtual environment. In the VEsNA vision, the intelligent agent will be able to reason on this displacement and on its compliance to legal and normative constraints. It will also be able to implement what-if analysis and case-based reasoning. Objects populating the virtual environment will include active objects and will populate a dynamic simulation whose outcomes will be interpreted by the cognitive agent; explanations and suggestions will be passed back to the user by the chatbot.

1 Introduction and Motivation

Factory automation is a safety-critical task that can help significantly in increasing production, but that does not come free of charge. Robots may be very expensive and their impact over the production pipeline is not always predictable. For this reason, many factory automation commercial software applications exist, including Computer Aided Design (CAD) tools and simulators[1]. Those applications are highly customized for factory automation. They are complete and efficient in that domain but require advanced programming skills to be used, that might prevent some categories of users from taking advantage of them – for example, small factories where industrial designers do not have sufficient computer science background, or where the price of those commercial software applications cannot be afforded. In those situations a cheap and "extremely-easy-to-setup" virtual twin of the factory along with the robots and the workers therein might bring many benefits, even if less effective and complete than the simulations generated by ad-hoc toolkits.

Consider, for example, the following scenario. Alice runs a small factory where various types of metallic components for industrial automation are assembled. She is a visionary businesswoman and in her vision safety deserves the first place. Alice got a funding for making her factory more efficient and productive by installing new robots. She wants to identify the safest configuration of robots in the

[1] See, for example, www.fastsuite.com/solutions-products/market-specific/factory-automation, store.indusuite.com/products/software/, www.createasoft.com/automation-simulation, https://new.siemens.com/global/en/products/automation/topic-areas/simulation-for-automation.html.

R.C. Cardoso, A. Ferrando, F. Papacchini,
M. Askarpour, L.A. Dennis (Eds.): Second
Workshop on Agents and Robots for reliable
Engineered Autonomy (AREA'22).
EPTCS 362, 2022, pp. 65–80, doi:10.4204/EPTCS.362.8

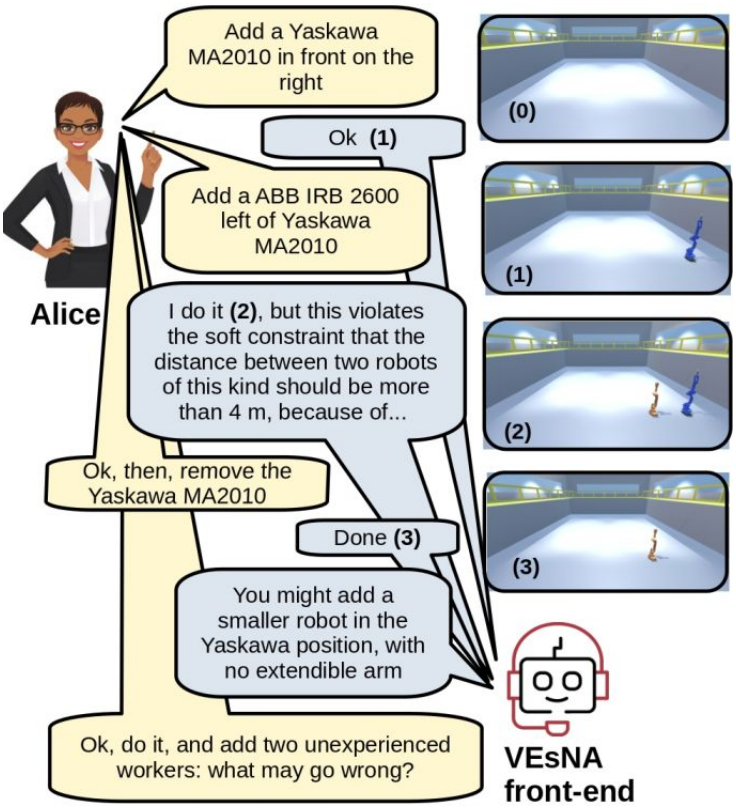


Figure 1: Example of visionary interaction between a user and VEsNA.

industrial building that she rented but her limited budget does not allow her to pay for the license of a commercial CAD or simulation tool. Alice is aware of specific issues raised by the robots she is going to buy and install, that may generate hard and soft constraints to meet, and of general legal requirements that may prevent her from making some choices. She also knows that robots may undergo malfunctioning, but that also human workers may not behave in the correct way. To ensure safety she would like to anticipate – or at least to understand – what might go wrong with different configurations of the robots. For this reason, she needs a (possibly unexpensive and open) tool that allows her to:

1 displace elements (different kinds of robots, furniture, ...) into a virtual environment via a user-friendly interface based on natural language interaction, in order to identify the best configuration of the building;

2 check that displaced objects meet the hard and soft safety and legal constraints related to their position and interactions;

3 provide a natural language explanation of why some constraints are not met by a given configuration, and suggest alternatives.

These three goals might be achieved by adding a natural language interface on top of any existing Computer Aided Design tool, but Alice needs something more sophisticated. Indeed, she would like to insert dynamic elements into the virtual environment, so that the result is not just a static rendering, but a running simulation. In particular, she wants to add virtual humans, and she wants to anticipate what may happen in case of unexpected or wrong maneuvers made by the human workers. The tool that Alice is looking for, should also allow her to

4 run simulations in the virtual environment, get statistics about them, and explain her – using a natural language interface – what may go wrong.

But this is still not enough. Once the best configuration will be devised, Alice wants to train her employees to move and act in the factory, before they start working in the real environment. The last feature of the tool Alice needs, is to

5 allow workers to *enter* the virtual and dynamic environment, interact with robots therein, learn what is safe and what is not, and get the most effective and realistic training with natural language explanations.

The tool Alice is looking for should

1 understand command issued in natural language; those commands might range from the simplest "Add a Yaskawa MA2010 in front on the right" to the more sophisticated "Add workers with some profile (experienced, unexperienced, reliable, etc...)"; the tool should provide a chatbot-like natural language interface;

2 be aware of rules (normative, physical, domain-dependent) and be able to verify whether a given configuration – that may include robots and workers – meets them; the tool should be able to reason on facts represented in a symbolic way ("a robot is placed in this position and an unexperienced worker is placed in that position"), on their logical consequences, and on the rules that may be broken by them;

3 be able to synthesise natural language explanations of its logical reasoning flow, besides reasoning and understanding natural language;

4 ensure a one-to-one correspondence between facts representing symbolic knowledge amenable for logical reasoning and objects placed in a virtual environment – equipped with a realistic graphical interface – where dynamic behaviors may be added: the tool should be suitable for running realistic simulations;

5 allow the virtual environment to be made available as a virtual reality for training purposes, with no extra effort.

To the best of our knowledge no such a tool exists: this paper moves the initial steps towards it by describing VEsNA, an implemented general-purpose framework for managing Virtual Environments via Natural language Agents[2]. At its current stage of development, VEsNA is far from offering the services that Alice needs. Nevertheless, the three technologies it builds upon have the *potential* to cope with all her needs, and some promising results have already been achieved in the natural language interaction, reasoning, explanation, simulation, training in the virtual reality dimension, although no integration among these capabilities has been provided yet. In fact, VEsNA exploits (i) **DialogFlow**[3] for building a chatbot-like interface, (ii) **JaCaMo** for integrating knowledge coming from the interaction with the user into a cognitive, rule-driven agent able to reason about this knowledge and to provide human-readable explanations; (iii) **Unity** for building the dynamic virtual environment, and letting human users immerse in it.

The goal that VEsNA aims at achieving in the future is to provide Alice and many other users with an integrated environment for managing virtual environments via natural language, cognitive agents able to

[2]VEsNA is freely available to the community from `https://github.com/driacats/VEsNA`.

[3]So far, DialogFlow is the only non open-source technology in VEsNA; we are considering to substitute it with an open-source equivalent application, Rasa, to make VEsNA completely open source.

support decision making thanks to their reasoning and explanatory capabilities: an example of interaction between Alice and VEsNA is depicted in Figure 1.

The problem that VEsNA solves in its current state is to integrate DialogFlow, JaCaMo and Unity into a single framework and to provide the means to *displace the elements relevant for finding the best configuration of the industrial building into a simulated environment via a user-friendly interface based on natural language interaction*.

The paper is organized in the following way. Section 2 introduces the three technologies VEsNA builds upon and motivates our confidence that filling the gap between the VEsNA-as-is and the VEsNA-to-be is technically feasible. Section 3 positions our contribution w.r.t. the state-of-the-art. Section 4 illustrates the envisioned VEsNA interaction and workflow and presents an example in the factory automation domain. Section 5 concludes the paper and discusses future developments.

2 Background

DialogFlow. DialogFlow [27] is a *lifelike conversational AI with state-of-the-art virtual agents* developed by Google. It allows users to create personal chatbots, namely **conversational agents** equipped with **intents**, **entities**, and **fulfillment**.

During a conversation between humans, a human speaker can utter different types of sentences, each one with a different intentional meaning. That meaning can be identified as the *intent* of that sentence. For example, if someone says *"Hello!"*, the hearer can infer that the (friendly, socially-oriented) intent is of greeting her. If someone says *"Go away!"*, the (unfriendly, command-like, action-oriented) intent is of having some concrete action performed by the hearer – namely, moving away. The speech act theory [4, 50] provides a theoretical and philosophical basis for this intention-driven communication model. In order to explain the map between sentences and intents to the chatbot agent, the agent's developer should provide examples of sentences that convey that intent, for each intent that is relevant for the application.

Real sentences are much more complex than *"Hello!"* and *"Go away!"*: they usually add some contextual information to the main intent. For example, in the *"My name is Bob"* sentence, the speaker's (friendly, socially-oriented) intent is to introduce himself, with the additional information about his name. The agent must be instructed that the *name* is something it should remember about persons, when they introduce themselves. This goal can be achieved by creating an *entity Person* with a field *name*, that we refer to as a "parameter". In the set of sentences associated with the intent, the string that identifies the name should be highlighted by the agent's developer, to let the agent learn how to retrieve the person's name inside sentences tagged with the "introduce himself" intent.

Once the sentence intent has been understood and the parameters have been retrieved, the agent must provide a meaningful and appropriate answer. By default, the answer is a fixed sentence whose only variable parts are those related with parameters identified in the input sentence. For example, if the user types *"Hello! My name is Bob"*, the agent can identify the name and answer something like *"Hi Bob! Nice to meet you!"*, but nothing more advanced. This is where *fulfillment* comes into play. The fulfillment is a sort of help from home: if the agent cannot answer messages for some specific intent, those messages are forwarded to an external, specialized source that is waiting. Fulfillment provides a field where the user can insert the URL address of the service to query. The service at that address will be consulted only for those intents that require it; in that case, DialogFlow will wait for the answer and will forward it to the user.

JaCaMo. JaCaMo [11, 10] *"is a framework for Multi-Agent Programming that combines three separate technologies, each of them being well-known on its own and developed for a number of years so they are fairly robust and fully-fledged"* [12].

The three technologies that JaCaMo integrates are: (1) **Jason** [14], for programming autonomous agents characterized by mentalistic notions like beliefs, goals, desires, intentions, and able to reason; (2) **CArtAgO** [49, 48], for programming environment artifacts; (3) **MOISE** [28], for programming multi-agent organisations. In our work, we will use only Jason and (in an indirect way) CArtAgO. By using JaCaMo we can build a multiagent system along with its environment. JaCaMo agents follow the Beliefs-Desires-Intentions (BDI) model [46, 26] and are implemented in Jason, that is a variant of the logic-based AgentSpeak(L) language [45]. The Jason elements that are more relevant for programming one individual, cognitive agent are:

- **Beliefs:** the set of facts the agent knows,
- **Goals:** the set of goals the agent wants to achieve,
- **Plans:** the set of pre-compiled, operational plans the agent can use to achieve its goals.

To make an example, a simple reactive agent that turns on the light either when someone enters the room, or when someone issues the "turn light on" voice command, or any other command with the same semantics, would need to know (1) either when someone enters the room or when the "turn light on" command has been issued, and (2) how to turn on the light. The first piece of knowledge is a *belief* (something the agent knows about the world either because it is informed of that fact by some other agent via communication, or because that fact about the environment is sensed via sensors or provided by artifacts, or because the agent itself generates that piece of knowledge) and the second one is a *plan* (a recipe to achieve some goal via a sequence of operational steps). Considering the first way to have light on, triggered from people entering the room, the scenario might involve a sensor that checks the presence of people in the room and triggers the addition or removal of belief from the agent's belief base depending on its measurement. Last, the agent has two *goals*:

1. turn on the light when the right condition is met,
2. turn off the light when no-one is there.

We assume that the agent's plans offer instructions on how to turn on the light, and how to turn it off. So, depending on its beliefs about the presence of people in the room (that in turn depend on the sensor's outcome, or on a command been issued), the agent decides to adopt one plan or the other, in order to achieve its goal.

The agent needs a way to communicate with the environment. Usually, CArtAgO artifacts are used for this purpose. In our example, we assume to have an artifact controlling the sensor and another one receiving information from a Dialogflow agent that listens for the user's commands. The bridge between DialogFlow and JaCaMo is built with **Dial4Jaca** [23, 21] that provides a set of CArtAgO artifacts that run throughout the execution of the JaCaMo agent. Dial4Jaca starts a listener and has the knowledge to receive and interpret messages from DialogFlow. These messages are not the ones the user writes on the chat, but a standardized and structured format that DialogFlow uses when it has to use fullfillments. Such messages contain the intent name and the entities' parameters identified (e.g., the object the user is talking about in the scene). When the messages are received by a JaCaMo agent, they are parsed and then added to the agent's beliefs, using Jason's syntax. Finally, the addition of such new belief inside the JaCaMo agent's belief base will cause a reaction driven by the most suitable plan among those in the agent's plan base. The use of Jason and JaCaMo paves the way to supporting those advanced

features mentioned in Section 1 namely sophisticated reasoning capabilities and goal-driven planning [55, 47, 40], exploitation of formal and semi-formal methods to implement monitoring and safety checks [13, 3, 24, 22], explainability, also in connection with DialogFlow thanks to Dial4JaCa [20, 25].

Unity. Unity [53] is a cross-platform game engine that allows developers to create scenes, and add objects to such scenes by dragging and dropping them from a palette to the scene. Objects inserted into a scene can be more or less realistic, and may enjoy or not physical properties, depending on what is needed in the application domain of choice. When an object is put inside a scene, a script written in C# can be attached to it. From that script, it is possible to instantiate other objects inside the scene, and modify (resp., destroy) those already placed there. Unity may allow users to have a controllable running simulation where elements have some degree of autonomy [9], and to turn that simulated environment into a virtual reality [29].

3 Related Work

The idea of integrating BDI agents into game engines like Unity is extremely recent and almost unexplored; surprisingly, the more general idea that existing game engines and simulation platforms are suited to act as platforms for building MASs - especially when learning capabilities are required - is still also poorly explored.

One of the first works dealing with the general vision dates back to 2014 [6] and presents a multi-agent system based on Unity 4 that allows simulating three-dimensional way-finding behavior of several hundreds of airport passengers on an average gaming personal computer. Although very preliminary, that work inspired successive research where – however – the focus was not on the use of a game engine as a general-purpose MAS platform, but rather on specific problems that a 3D realistic simulation raises such as signage visibility to improve pathfinding [39], and on ad-hoc simulations [56, 57, 31].

In [30] Juliani et al. move a step forward generalization and present a taxonomy of existing simulation platforms that enable the development of learning environments that are rich in visual, physical, task, and social complexity. In their paper, the authors argue that modern game engines are uniquely suited to act as general platforms and examine Unity and the open source Unity ML-Agents Toolkit[4] as case studies.

When moving to the integration of BDI-style agents into game engines, one of the first works to mention is the PRESTO project [17], spanning 2013-2016, whose outcomes were significantly constrained by the immaturity of game engines like Unity, besides the immaturity of platforms for BDI agents.

In the Master Thesis by N. Poli dating back 2018 [43], simple BDI agents were implemented using a lightweight Prolog engine, tuProlog [19], that overcame some limitations of UnityProlog[5], an existing Prolog intrpreter compatible with Unity3D. A roadmap to exploit game engines to model MAS that also discusses the results achieved in [43] has been published by S. Mariani and A. Omicini in 2016 [34].

Similarly to the work by N. Poli, the work by Marín-Lora et al. [35] describes a game engine to create games as multi-agent systems where the behaviour specification system is based on first-order logic and is hence closer to a declarative approach than a purely procedural programming language.

Simulations that exploit Unity as the engine and visualization tool and the BDI model as a reference for implementing individual agents have been developed in a few domains including large urban areas

[4] `https://github.com/Unity-Technologies/ml-agents`
[5] `https://github.com/ianhorswill/UnityProlog`.

[52], fire evacuation [41], first aid emergency [7], gas and oil industry [37], bushfires in Australia [54]. Those works are driven by an application to simulate and lack re-usability in other contexts.

The work closest to ours, at least in its final goal of creating a general-purpose extension of Unity that integrates BDI agents, is hence the one by A. Brännström and J. C. Nieves in [16]. There, the authors introduce UnityIIS, a lightweight framework for implementing intelligent interactive systems that integrate symbolic knowledge bases for reasoning, planning, and rational decision-making in interactions with humans. This is done by integrating Web Ontology Language (OWL)-based reasoning [5] and Answer Set Programming (ASP)-based planning software [32, 33] into Unity. Using the components of the UnityIIS framework, the authors developed an Augmented Reality chatbot following a BDI model: beliefs are the agent's internal knowledge of its environment, which is updated during the interaction by getting new observations; desires are goals that the agent aims to fulfill, which are updated during the interaction by reflecting upon new beliefs; intentions are what goals the agent has chosen to achieve, selected in a deliberation process and used for generating a plan. The belief of the agent is represented in an OWL ontology, as also suggested by other authors in the past [38, 36]. The UnityIIS framework enables belief revisions (OWL/ASP file updates) at run time by interweaving the agent's control loop with the Unity game loop. The Unity game loop does cyclic update iterations on a given frequency. During each frame, external inputs are processed, game status is updated, and graphics are redrawn.

Albeit sharing some similarities, there are also differences between VEsNA and UnityIIS. The main one is that UnityIIS does not integrate JaCaMo with Unity: the BDI model is used as a reference, but is not implemented using a standard agent programming language as Jason. Rather, UnityIIS relies on OWL and ASP as languages for modelling knowledge and declarative behaviour of cognitive agents. The second one is that the chatbot described in [16] is an example of application of UnityIIS, in the same way as factory automation is an example of application of VEsNA. In VEsNA, the chatbot is one of the three pillars of the framework, and not just an application. What we might borrow from the UnityIIS model, is the adoption of OWL to model knowledge and reason about it in an interoperable, portable way. What UnityIIS might borrow from VEsNA, is the closer integration of a standard framework for BDI agents into the system, JaCaMo, along with all its libraries and add-ons, rather than the adoption of more generic logic-based languages like ASP and OWL.

4 Envisioned VEsNA Interaction and Workflow

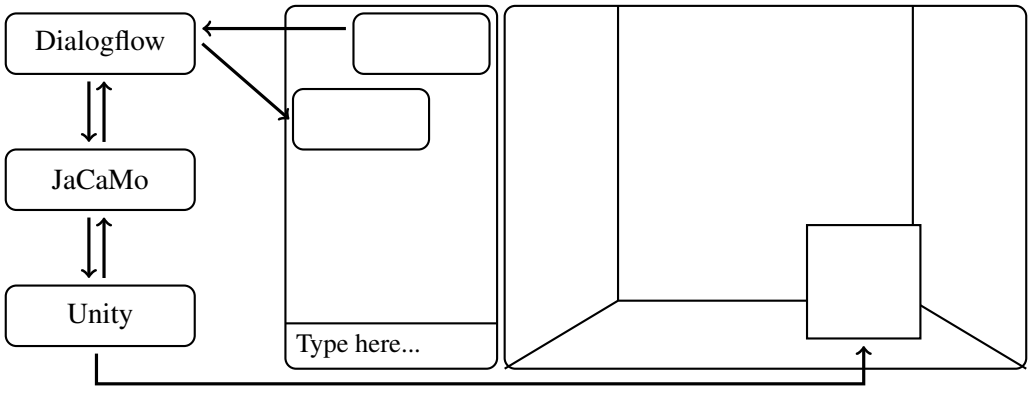


Figure 2: VEsNA architectural scheme.

In this section, we describe how a user may interact with VEsNA. Figure 2 reports a high-level architectural scheme of VEsNA. On the VEsNA screen, the user can find a chat and a pre-built scene made with Unity representing whatever he/she needs, based on the application domain; the scene has at least one empty floor. The user can describe how the scene is organized via the chat, using natural language[6] and so telling the VEsNA agent what to do. For example, the user could say[7] *"Add that object in the scene!"*. This would make the object appear inside the scene. However, the translation from the message to the graphics animation requires various intermediate steps.

First, the DialogFlow agent identifies the name of the object the user wants to add along with its position in the scene. This information is automatically extracted from the user's sentence through natural language processing. In more detail, the position can be described by the user in two different ways. The first is by using a **global positioning** on the entire space available. In the current implementation, the space is subdivided into 9 available positions. Hence, the user can say something like *"Ok agent, put that object in front on the right"* and the agent will understand what it is expected to do. While the global positioning may work well with small scenarios, it is likely to be too coarse grained with big ones; however, this may represent the "first" positioning of objects inside the scene. Indeed, other objects may be added later on by referring to relative positions (w.r.t. already present objects in the scene). The user can add new objects that **reference** those already in scene. In that case, it would be possible to say *"Ok, I told you that there is that object on right right. Well, behind it there is this other object"*. This is performed by giving a unique name to each object added inside the scene. Such name can be used any time it is needed to identify an object as reference. Note that, when a user adds an object, the answer contains the *reference name* for it.

Furthermore, while Dialogflow can identify every object name inserted by the user, an error will be generated if it is not one of the available items. To make the system work, the user has to insert the name of the object in Unity, in such a way that DialogFlow can recognize it. For the moment, this applies also to identifiers of already added objects: they must be those provided by Unity and communicated to the user. We are working to overcome this limitation by introducing scopes that will allow the user a more natural description of the scene, making prior knowledge available and supporting the ability to refer the last added objects without having to remember their identifiers.

Once the user's sentence has been processed by DialogFlow, the resulting intent is generated. In the case of the addition of a new object inside the scene, the intent *AddObject* is used and trained to identify *who* is the object to add and *where* to add it. It has Fulfillment flagged on, so that when a sentence is traced back to that intent, a request is propagated to the address where the JaCaMo agent is listening. It is relevant to remind here that only intents with the Fulfillment flagged on are propagated to the JaCaMo agent; all other intents are not propagated, which means are not in need of an agent to be handled (e.g., sentences not concerning the scene).

After the *AddObject* intent has been generated by DialogFlow and propagated to the VEsNA agent along with the *who* and *where* information, a request to Unity via http is sent by the agent.

Unity has an asynchronous listener waiting for requests and receives all the information sent by the agent as strings. For example, if the user said in the chat *"Put that object on the right"*, what arrives to Unity would only be `''that object, center, right''`. After that, Unity converts the received string into a vector that describes the object position inside the scene. In case of global positioning,

[6] So far we have run experiments with English only, but given that DialogFlow natively supports multiple languages, `https://cloud.google.com/dialogflow/es/docs/agents-multilingual`, letting the user talk in his/her own language should be also possible.

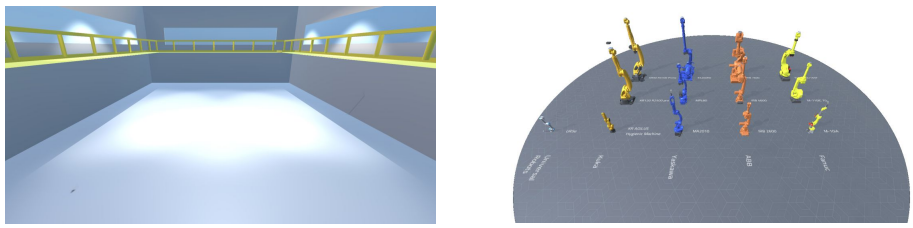
[7] So far, interactions are via a textual interface only; nonetheless, to plug a voice-to-text component into the VEsNA control flow should not raise any technical issues.

the resulting conversion is simple and is computed w.r.t. the size of the scene. However, for relative positioning, Unity has to look for the object used as a reference to compute the new position w.r.t the latter.

Once the position for the new object is found, a physical check is performed on the Unity side to be sure the selected place has not already been taken by another object (this is done by exploiting native Unity's colliders). If the position is already taken, an error message is sent back to the JaCaMo agent. Otherwise, the object is added and a "done" message is sent back to the JaCaMo agent containing the unique name of the object just added.

At this point, JaCaMo sends back a positive answer to DialogFlow, and a message is displayed to the user via the chat. Finally, the user may decide to stop or to start another iteration by adding a brand new object to the scene.

As a concrete example of use, let us consider the scenario introduced in Section 1. Let us suppose that VEsNA's user is the owner of a small-medium factory where extended-reach welding robots (robots that can move their only arm in almost all directions, but cannot wander through the factory, see Figure 3b) must be positioned in an optimal way to carry out some automation work. When VEsNA is first run, an empty model of the factory in Unity is available to the user, ready to be modified (Figure 3a). In this



(a) The factory (b) The robots

Figure 3: Model in Unity.

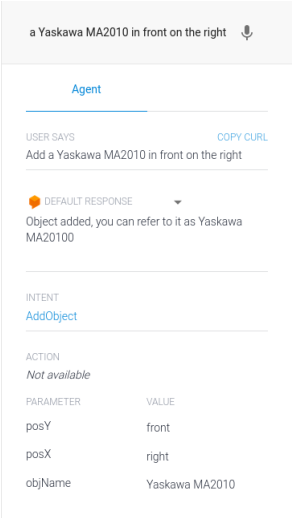
case, the user is interested in simulating in Unity the outcome of putting robots in different positions, to devise the optimal one. However, the user might not have the programming skills for working in Unity. This is when VEsNA comes into play to let the user position objects in the Unity scene by using natural language. Let us suppose that the user wants to position a Yaskawa MA2010 robot[8] in the front-right global position. The user may type *"Add a Yaskawa MA2010 in front on the right"* in the VEsNA chat. The sentence in the chat (Figure 4a) automatically creates the scene in Figure 4b. The machinery behind the result displayed in Figure 4b is the following. The Dialogflow agent brings the *"Add a Yaskawa MA2010 in front on the right"* back to the intent *AddObject*. An entity **Object** that has parameters

1. **Name**, the name of the object to be positioned, the robot in this case;
2. **PosX**, in global positioning is the horizontal position, in relative positioning contains the relative position (*left of*, *right of*, *behind*, *in front of*);
3. **PosY**, in global positioning is the forward position, in relative positioning contains the name of the object the user referes to.

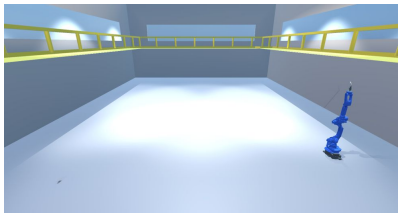
The VEsNA agent finds in the sentence the name *"Yaskawa MA2010"*, the posX *"right"* and the posY *"front"* and sends a request to the link provided in the Fulfillment (so where the JaCaMo agents awaits).

Dial4Jaca receives the information from Dialogflow and adds the following belief to the JaCaMo agent:

[8] `https://www.robots.com/robots/motoman-ma2010`.



(a) The Dialogflow chat



(b) The graphical result

Figure 4: Example of object addition with global coordinates.

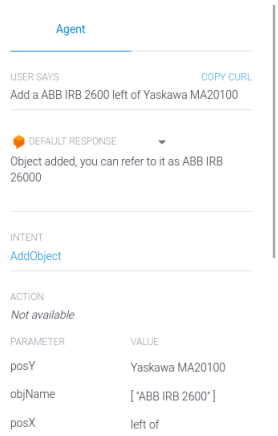
```
request(
        "undefined",
        "5b485464-f275-42ab-853e-59514b115359-cf898478",
        "AddObject",
        [
                param("posX","right"),
                param("posY","front"),
                param("objName","Yaskawa  MA2010")
        ],
        ...
```

The agent gets the object name and the position, then it calls the `addObject` function, implemented in a JaCaMo artifact. This function sends an HTTP request to the 8081 port on the same machine and waits for an answer (we remind that on port 8081 Unity is listening for messages). The HTTP request is assembled as follows

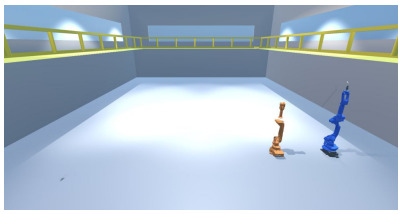
$$\texttt{http://localhost:8081/}\underbrace{\texttt{Yaskawa\%20MA2010}}_{\texttt{obj name}}\texttt{/}\underbrace{\texttt{right}}_{\texttt{posX}}\texttt{/}\underbrace{\texttt{front}}_{\texttt{posY}}$$

On the Unity side, the script attached to the factory has a listener on that port and receives this piece of information. It then converts the location from a couple of strings (in our example $(right, front)$) into a 3D vector of coordinates. When it has a position, it checks that it is not already occupied by something else and adds it to the scene.

Let us now assume that the user wants to continue the interaction with VEsNA. For instance, the user could ask to add a new robot to the left of the one inside the scene. Note that, since we now have already another robot inside the scene, we can add objects and position them by reference. In this case, the user could write on the chat something like *"Add a ABB IRB 2600 left of Yaskawa MA2010"*. The result we would obtain by such interaction is shown in Figure 5b:



(a) The DialogFlow chat



(b) The graphical result

Figure 5: Example of object addition with relative coordinates.

As last example, let us suppose that the user changes his/her mind and decides to remove the robot positioned at the beginning. The user could type *"Remove the Yaskawa MA20100"* and the result would be as shown in Figure 6, left, where the blue arm is no longer part of the scene.

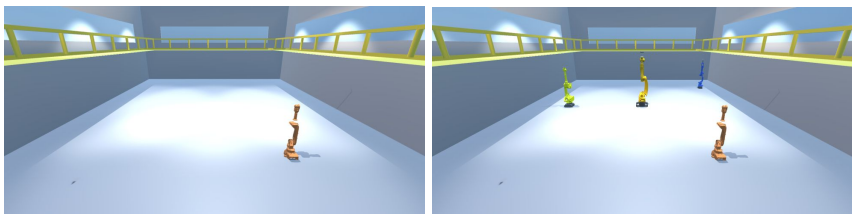


Figure 6: Example of object removal (left) and result of further interactions (right).

Finally, after a few iterations, all the robots would be positioned in the factory as the user desires and the result would look like the scene shown in Figure 6, right (this is just a possible example of final result).

5 Conclusions and Future Work

We presented VEsNA, a general-purpose framework to allow any users to populate simulations in Unity through natural language, heavily exploiting agent-based technologies that might help in reasoning on scenarios, verifying the feasibility of choices, providing explanations. We showed how starting from natural language sentences, VEsNA can automatically handle the creation, addition, and removal of objects into a simulated scene in Unity. Specifically, this is obtained in VEsNA through the combination of three different frameworks: DialogFlow, JaCaMo, and Unity. These frameworks support natural language analysis, the scene construction, and its graphic representation, respectively. To the best of our knowledge, VEsNA is the first framework which combines all these three aspects together.

For future developments, we are planning to explore additional uses for the objects in the simulated scene. In particular, instead of having static objects, we might have active objects as agents. This would

help making the simulation more realistic, engaging, and amenable for a more precise risk analysis. In the factory automation scenario, employees could also be simulated as agents in the factory to experiment the movements and the interactions with objects and agents therein.

Being general-purpose, VEsNA is not limited to the factory automation domain. Indeed, one of our original reasons for the development of VEsNA was to create a tool for interactive, agent-based theatrical story-telling, along the lines of [51, 18, 44, 42, 8]. In that domain the presence of cognitive, goal-driven characters in the Unity scene able to perform some actions on their own, provides a strong motivation for the integration of JaCaMo and Unity.

Finally, from a more practical perspective, we are also considering to explore additional frameworks to improve the VEsNA usability and accessibility. For instance, other options are available on the sentence analysis side, such as Rasa[9]. Rasa is a valid alternative to DialogFlow because open-source and can be run locally, differently from DialogFlow which instead requires to be executed in the cloud. At the current state, we opted for DialogFlow for its simplicity, and native support in Dial4JaCa. However, if used in scenarios where privacy might be an issue, the possibility to have all software running locally could be a necessary feature.

References

[1] *OWLAPI*. Available at `https://github.com/owlcs/owlapi#readme`.

[2] *Protégé*. Available at `https://protege.stanford.edu/`.

[3] Davide Ancona, Sophia Drossopoulou & Viviana Mascardi (2012): *Automatic Generation of Self-monitoring MASs from Multiparty Global Session Types in Jason*. In Matteo Baldoni, Louise A. Dennis, Viviana Mascardi & Wamberto Weber Vasconcelos, editors: *Declarative Agent Languages and Technologies X - 10th International Workshop, DALT 2012, Valencia, Spain, June 4, 2012, Revised Selected Papers*, *Lecture Notes in Computer Science* 7784, Springer, pp. 76–95, doi:10.1007/978-3-642-37890-4_5.

[4] John Langshaw Austin (1962): *How to Do Things with Words*. William James Lectures, Oxford University Press. Available at `http://scholar.google.de/scholar.bib?q=info:xI2JvixH8_QJ:scholar.google.com/&output=citation&hl=de&as_sdt=0,5&ct=citation&cd=1`.

[5] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter Patel-Schneijder & Lynn Andrea Stein (2004): *OWL Web Ontology Language Reference*. Recommendation, World Wide Web Consortium (W3C). See `http://www.w3.org/TR/owl-ref/`.

[6] Christian Becker-Asano, Felix Ruzzoli, Christoph Hölscher & Bernhard Nebel (2014): *A Multi-agent System based on Unity 4 for Virtual Perception and Wayfinding*. *Transportation Research Procedia* 2, pp. 452–455, doi:10.1016/j.trpro.2014.09.059. Available at `https://www.sciencedirect.com/science/article/pii/S2352146514000957`. The Conference on Pedestrian and Evacuation Dynamics 2014 (PED 2014), 22-24 October 2014, Delft, The Netherlands.

[7] Samira Benkhedda & Fatima Bendella (2019): *FASim: A 3D Serious Game for the First Aid Emergency*. *Simul. Gaming* 50(6), p. 690–710, doi:10.1177/1046878119865913.

[8] Leonid Berov (2017): *Character focused narrative models for computational storytelling*. In: *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[9] Alessandro Biagetti, Angelo Ferrando & Viviana Mascardi (2020): *The DigForSim Agent Based Simulator of People Movements in Crime Scenes*. In Yves Demazeau, Tom Holvoet, Juan M. Corchado & Stefania Costantini, editors: *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness. The PAAMS Collection - 18th International Conference, PAAMS 2020, L'Aquila, Italy, October 7-9,*

[9] `https://rasa.com/`

2020, Proceedings, *Lecture Notes in Computer Science* 12092, Springer, pp. 42–54, doi:10.1007/978-3-030-49778-1_4.

[10] Olivier Boissier, Rafael H Bordini, Jomi Hubner & Alessandro Ricci (2020): *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. MIT Press.

[11] Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci & Andrea Santi (2013): *Multi-agent oriented programming with JaCaMo*. *Science of Computer Programming* 78(6), pp. 747–761, doi:10.1016/j.scico.2011.10.004.

[12] Olivier Boissier, Rafael H. Bordini, Jomi H. Hübner, Alessandro Ricci & Andrea Santi: *JaCaMo Project*. Available at `http://jacamo.sourceforge.net/`.

[13] Rafael H. Bordini, Michael Fisher, Carmen Pardavila & Michael J. Wooldridge (2003): *Model checking AgentSpeak*. In: *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, ACM, pp. 409–416, doi:10.1145/860575.860641.

[14] Rafael H Bordini, Jomi Fred Hübner & Michael Wooldridge (2007): *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, doi:10.1002/9780470061848.

[15] Jack Brookes, Matthew Warburton, Mshari Alghadier, Mark Mon-Williams & Faisal Mushtaq (2020): *Studying human behavior with virtual reality: The Unity Experiment Framework*. *Behavior research methods* 52(2), pp. 455–463, doi:10.3758/s13428-019-01242-0.

[16] Andreas Brännström & Juan Carlos Nieves (2022): *A Framework for Developing Interactive Intelligent Systems in Unity*. In Jürgen Dix Amit Chopra & Rym Zalila-Wenkstern, editors: *Engineering Multi-Agent Systems (EMAS 2022)*, doi:10.1007/978-3-030-97457-2.

[17] Paolo Busetta, Paolo Calanca & Marco Robol (2016): *Applying BDI to serious games: The PRESTO experience*. Technical Report, University of Trento Trento, Italy.

[18] Rossana Damiano & Vincenzo Lombardo (2009): *Value-Driven Characters for Storytelling and Drama*. In Roberto Serra & Rita Cucchiara, editors: *AI*IA 2009: Emergent Perspectives in Artificial Intelligence*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 436–445, doi:10.1007/978-3-642-10291-2_44.

[19] Enrico Denti, Andrea Omicini & Alessandro Ricci (2005): *Multi-paradigm Java-Prolog integration in tuProlog*. *Sci. Comput. Program.* 57(2), pp. 217–250, doi:10.1016/j.scico.2005.02.001.

[20] Débora C. Engelmann, Lucca Dornelles Cezar, Alison R. Panisson & Rafael H. Bordini (2021): *A Conversational Agent to Support Hospital Bed Allocation*. In André Britto & Karina Valdivia Delgado, editors: *Intelligent Systems - 10th Brazilian Conference, BRACIS 2021, Virtual Event, November 29 - December 3, 2021, Proceedings, Part I*, *Lecture Notes in Computer Science* 13073, Springer, pp. 3–17, doi:10.1007/978-3-030-91702-9_1.

[21] Débora C. Engelmann, Juliana Damasio, Tabajara Krausburg, Olimar Teixeira Borges, Mateus da Silveira Colissi, Alison R. Panisson & Rafael H. Bordini (2021): *Dial4JaCa - A Communication Interface Between Multi-agent Systems and Chatbots*. In Frank Dignum, Juan Manuel Corchado & Fernando de la Prieta, editors: *Advances in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection - 19th International Conference, PAAMS 2021, Salamanca, Spain, October 6-8, 2021, Proceedings*, *Lecture Notes in Computer Science* 12946, Springer, pp. 77–88, doi:10.1007/978-3-030-85739-4_7.

[22] Débora C. Engelmann, Angelo Ferrando, Alison R. Panisson, Davide Ancona, Rafael H. Bordini & Viviana Mascardi (2022): *RV4JaCa - Runtime Verification for Multi-Agent Systems*. In: *Proceedings of AREA 2022, the Second Workshop on Agents and Robots for reliable Engineered Autonomy*.

[23] Débora Engelmann, Juliana Damasio Oliveira, Olimar Teixeira Borges, Tabajara Krausburg, Marivaldo Vivan, Alison R. Panisson & Rafael H. Bordini: *Dial4Jaca*. Available at `https://github.com/smart-pucrs/Dial4JaCa`.

[24] Angelo Ferrando, Louise A. Dennis, Davide Ancona, Michael Fisher & Viviana Mascardi (2018): *Recognising Assumption Violations in Autonomous Systems Verification*. In Elisabeth André, Sven Koenig, Mehdi Dastani & Gita Sukthankar, editors: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, pp. 1933–1935. Available at `http://dl.acm.org/citation.cfm?id=3238028`.

[25] Carlos Eduardo A. Ferreira, Alison R. Panisson, Débora C. Engelmann, Renata Vieira, Viviana Mascardi & Rafael H. Bordini (2022): *Explaining Semantic Reasoning using Argumentation*. In: *20th International Conference on Practical Applications of Agents and Multi-Agent Systems, PAAMS 2022, Proceedings*.

[26] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe & Michael Wooldridge (1998): *The Belief-Desire-Intention model of agency*. In: *International workshop on agent theories, architectures, and languages*, Springer, pp. 1–10, doi:10.1007/3-540-49057-4_1.

[27] Google: *DialogFlow*. Available at `https://cloud.google.com/dialogflow/`.

[28] Jomi F Hubner, Jaime S Sichman & Olivier Boissier (2007): *Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels*. *International Journal of Agent-Oriented Software Engineering* 1(3-4), pp. 370–395, doi:10.1504/IJAOSE.2007.016266.

[29] Jason Jerald, Peter Giokaris, Danny Woodall, Arno Hartbolt, Anish Chandak & Sébastien Kuntz (2014): *Developing virtual reality applications with Unity*. In: *2014 IEEE Virtual Reality, VR 2014, Minneapolis, MN, USA, March 29 - April 2, 2014*, IEEE Computer Society, pp. 1–3, doi:10.1109/VR.2014.6802117.

[30] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar & Danny Lange (2018): *Unity: A General Platform for Intelligent Agents*. *CoRR* abs/1809.02627. Available at `http://arxiv.org/abs/1809.02627`.

[31] Sara Karami & Mohammad Taleai (2022): *An innovative three-dimensional approach for visibility assessment of highway signs based on the simulation of traffic flow*. *Journal of Spatial Science* 67(2), pp. 203–218, doi:10.1080/14498596.2020.1787253.

[32] Vladimir Lifschitz (1999): *Action Languages, Answer Sets, and Planning*. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczynski & David Scott Warren, editors: *The Logic Programming Paradigm - A 25-Year Perspective*, Artificial Intelligence, Springer, pp. 357–373, doi:10.1007/978-3-642-60085-2_16.

[33] Vladimir Lifschitz (2019): *Answer Set Programming*. Springer, doi:10.1007/978-3-030-24658-7.

[34] Stefano Mariani & Andrea Omicini (2016): *Game Engines to Model MAS: A Research Roadmap*. In Corrado Santoro, Fabrizio Messina & Massimiliano De Benedetti, editors: *Proceedings of the 17th Workshop "From Objects to Agents" co-located with 18th European Agent Systems Summer School (EASSS 2016), Catania, Italy, July 29-30, 2016*, *CEUR Workshop Proceedings* 1664, CEUR-WS.org, pp. 106–111. Available at `http://ceur-ws.org/Vol-1664/w18.pdf`.

[35] Carlos Marín-Lora, Miguel Chover, José M. Sotoca & Luis A. García (2020): *A game engine to make games as multi-agent systems*. *Advances in Engineering Software* 140, p. 102732, doi:10.1016/j.advengsoft.2019.102732.

[36] Viviana Mascardi, Davide Ancona, Matteo Barbieri, Rafael H. Bordini & Alessandro Ricci (2014): *CooL-AgentSpeak: Endowing AgentSpeak-DL agents with plan exchange and ontology services*. *Web Intell. Agent Syst.* 12(1), pp. 83–107, doi:10.3233/WIA-140287.

[37] Otávio Arruda Matoso, Luis Lampert, Jomi Fred Hübner, Mateus Conceição, Sérgio P Bernardes, Cleber Jorge Amaral, Maicon R Zatelli & Marcelo L de Lima (2020): *Agent Programming for Industrial Applications: Some Advantages and Drawbacks*. *arXiv preprint arXiv:2006.05613*.

[38] Álvaro F. Moreira, Renata Vieira, Rafael H. Bordini & Jomi Fred Hübner (2005): *Agent-Oriented Programming with Underlying Ontological Reasoning*. In Matteo Baldoni, Ulle Endriss, Andrea Omicini & Paolo Torroni, editors: *Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*, *Lecture Notes in Computer Science* 3904, Springer, pp. 155–170, doi:10.1007/11691792_10.

[39] Ali Motamedi, Zhe Wang, Nobuyoshi Yabuki, Tomohiro Fukuda & Takashi Michikawa (2017): *Signage visibility analysis and optimization system using BIM-enabled virtual reality (VR) environments*. *Advanced Engineering Informatics* 32, pp. 248–262, doi:10.1016/j.aei.2017.03.005. Available at `https://www.sciencedirect.com/science/article/pii/S1474034616301203`.

[40] Vágner de Oliveira Gabriel, Alison R. Panisson, Rafael H. Bordini, Diana Francisca Adamatti & Cléo Zanella Billa (2020): *Reasoning in BDI agents using Toulmin's argumentation model*. *Theor. Comput. Sci.* 805, pp. 76–91, doi:10.1016/j.tcs.2019.10.026.

[41] Celine Haren Paschal, Cheah Wai Shiang, Sim Keng Wai & Muhammad Asyraf bin Khairuddin (2022): *Developing Fire Evacuation Simulation Through Emotion-based BDI Methodology*. *JOIV: International Journal on Informatics Visualization* 6(1), pp. 45–52.

[42] Federico Peinado, Marc Cavazza & David Pizzi (2008): *Revisiting Character-Based Affective Storytelling under a Narrative BDI Framework*. In Ulrike Spierling & Nicolas Szilas, editors: *Interactive Storytelling*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 83–88, doi:10.1007/978-3-540-89454-4_13.

[43] Nicola Poli (2018): *Game Engines and MAS: BDI & artifacts in Unity*.

[44] Stefan Rank, Steve Hoffmann, Hans-Georg Struck, Ulrike Spierling & Paolo Petta (2012): *Creativity in configuring affective agents for interactive storytelling*. In: *International conference on computational creativity*, p. 165.

[45] Anand S Rao (1996): *AgentSpeak (L): BDI agents speak out in a logical computable language*. In: *European workshop on modelling autonomous agents in a multi-agent world*, Springer, pp. 42–55, doi:10.1007/BFb0031845.

[46] Anand S Rao & Michael P Georgeff (1995): *BDI agents: from theory to practice*. In: *Proceedings of the First International Conference on Multiagent Systems, ICMAS*, 95, AAAI, pp. 312–319.

[47] Alessandro Ricci, Rafael Heitor Bordini, Jomi Fred Hübner & Rem W. Collier (2018): *AgentSpeak(ER): An Extension of AgentSpeak(L) improving Encapsulation and Reasoning about Goals*. In Elisabeth André, Sven Koenig, Mehdi Dastani & Gita Sukthankar, editors: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, pp. 2054–2056. Available at `http://dl.acm.org/citation.cfm?id=3238069`.

[48] Alessandro Ricci, Michele Piunti & Mirko Viroli (2011): *Environment programming in multi-agent systems: an artifact-based perspective*. *Autonomous Agents and Multi-Agent Systems* 23(2), pp. 158–192, doi:10.1007/s10458-010-9140-7.

[49] Alessandro Ricci, Mirko Viroli & Andrea Omicini (2006): *CArtAgO: A framework for prototyping artifact-based environments in MAS*. In: *International Workshop on Environments for Multi-Agent Systems*, Springer, pp. 67–86.

[50] John R. Searle (1979): *Expression and Meaning: Studies in the Theory of Speech Acts*. Cambridge University Press, doi:10.1017/CBO9780511609213.

[51] Ulrike Spierling, Dieter Grasbon, Norbert Braun & Ido Iurgel (2002): *Setting the scene: playing digital director in interactive storytelling and creation*. *Computers & Graphics* 26(1), pp. 31–44, doi:10.1016/S0097-8493(01)00176-5.

[52] Panich Sudkhot & Chattrakul Sombattheera (2018): *A Crowd Simulation in Large Space Urban*. In: *2018 International Conference on Information Technology (InCIT)*, pp. 1–8, doi:10.23919/INCIT.2018.8584878.

[53] Unity Technologies: *Unity*. Available at `https://unity.com/`.

[54] Sim Keng Wai, Cheah WaiShiang, Muhammad Asyraf bin Khairuddin, Yanti Rosmunie Binti Bujang, Rahmat Hidayat & Celine Haren Paschal (2021): *Autonomous Agents in 3D Crowd Simulation Through BDI Architecture*. *JOIV: International Journal on Informatics Visualization* 5(1), pp. 1–7, doi:10.30630/joiv.5.1.371.

[55] Michael J. Wooldridge (2000): *Reasoning about rational agents*. Intelligent robots and autonomous agents, MIT Press.

[56] Jiacheng Xie, Zhaojian Yang, Xuewen Wang, Quanren Zeng, Juanli Li & Bo Li (2018): *A Virtual Reality Collaborative Planning Simulator and Its Method for Three Machines in a Fully Mechanized Coal Mining Face*. Arabian Journal for Science and Engineering 43, doi:10.1007/s13369-018-3164-8.

[57] Berfin Yıldız & Gülen Çağdaş (2020): *Fuzzy logic in agent-based modeling of user movement in urban space: Definition and application to a case study of a square*. Building and Environment 169, p. 106597, doi:10.1016/j.buildenv.2019.106597.