Spatial Agent-based Architecture Design Simulation Systems

Anatolii Kotov[1], Rolf Starke[2], Ilija Vukorep[3]
[1,2,3]Brandenburg University of Technology Cottbus-Senftenberg
[1]anatolii.kotov@b-tu.de [2]starke@b-tu.de [3]ilija.vukorep@b-tu.de

This paper presents case studies and analysis of agent-based reinforcement learning (RL) systems towards practical applications for specific architecture/engineering tasks using Unity 3D-based simulation methods. Finding and implementing sufficient abstraction for architecture and engineering problems to be solved by agent-based systems requires broad architectural knowledge and the ability to break down complex problems. Modern artificial intelligence (AI) and machine learning (ML) systems based on artificial neural networks can solve complex problems in different domains such as computer vision, language processing, and predictive maintenance. The paper will give a theoretical overview, such as more theoretical abstractions like zero-sum games, and a comparison of presented games. The application section describes a possible categorization of practical usages. From more general applications to more narrowed ones, we explore current possibilities of RL application in the field of relatable problems. We use the Unity 3D engine as the basis of a robust simulation environment.

Keywords: *AI Aided Architecture, Reinforcement Learning, Agent Simulation.*

INTRODUCTION

Artificial Intelligence (AI) and Machine Learning (ML) methods are often more sophisticated in architecture and design than in standard tasks such as vision, natural language processing, and robotics. In many cases, the problem formulation is too vague or there are missing toolkits or datasets. The purpose of this paper is to present some theoretical and practical approaches to the application of reinforcement learning (RL) for architecture, design, and engineering. RL usually operates within an environment (game) that generates different situations for the agent to react to. Based on observations of this environment, an agent performs certain operations, acting in it and modifying it. To formulate design tasks in terms of RL is not trivial, so we need to dive into particular aspects of theory, methods and limitations. Therefore, we discuss problems and methods first in relation to general and design domain problems. Then we move on to the methods' application section and categorize practical simulated case studies made with the Unity game engine.

The black box problem

Artificial neural networks are good for solving complex problems in computer vision, language processing, and predictive maintenance. However, there is no possibility of interpreting how those results were achieved - a phenomenon called the black box problem, appearing in different areas of AI/ML application (Rudin and Radin, 2019). It describes a lack of understanding of why a system chooses one solution over another. If there are errors, a better understanding would be important, as it is difficult to tell from the neural network's

weights why these errors occur and how to avoid them. As a solution to this problem, the explainable AI (XAI) concept was introduced (Gunning and Aha, 2019). XAI systems are implementing different types of explainability ranging, depending on the target audience and their goals and domains (Heuillet, Couthouis, and Díaz-Rodríguez, 2020). The methods include but are not limited to graphs, text commentaries to actions and summation, generated visuals, or augmented vision inputs such as saliency maps and others.

Explainability is crucial in the building design domain as well, where processes must be reproducible and planning errors can be costly. As architecture and engineering tasks are by definition multimodal, it is difficult to formalize them without sacrificing some additional data or knowledge. Most likely, this is due to the nature of such tasks, as designing generally requires some level of invention, thinking outside the box. In other words, any attempt to implement an explanation by limiting constraints, problem domains, or goals can effectively narrow the solution. Therefore, we must strike a balance between the robustness of the system, its generalizability, and its explainability.

With RL, AI systems may eventually be able to solve architectural problems. RL can capture the essence of complex action spaces and general knowledge (Silver et al., 2021). The pursuit of explainable RL (XRL) may be more difficult in terms of pure mathematics since one has to explain not only some neural network prediction but also the policy, meaning why some action was taken in certain circumstances. However, we believe that the way we formulate the problem can counteract difficulties in understanding AI models. We asume that for architects and engineers, it is much more intuitive to formulate problems in a game-like manner. By formulating our end goals, we do not limit the ways that artificial intelligence can achieve them. In this way, we eliminate potential biases in datasets based on human-generated and -curated data. The downside, however, is computation time and problems with convergence. The RL system may take extensive time to find an optimal policy when the problem tends to be complex, or it may even be impossible if the game reward function is too sparse.

Zero-sum and non-zero-sum games in architecture

A lot of exemplary games used in some form with AI/ML are very often zero-sum games. This includes, but is not limited to chess, go, poker, bridge and others. The simplified concept is that players can win only by the loss of the other players. The total sum of losses and gains is equal to zero, therefore it's called zero-sum.

Due to the nature of some zero-sum games and their competitive subset it is conceptually easier and more natural to implement self-play games setups, where an AI system can play either against itself or other agent instances to improve. For RL, self-play is one of the key system components for such systems as AlphaGo (Silver *et al.*, 2016). Also, most classic games have very definitive game endings, e.g., in chess either way one of the players will prevail. This is leading to incremental learning improvements in the case of RL applied systems, utilizing clusters of different players and evolving the best playing agent.

Non-zero-sum games on the other hand are implying that the total sum of gains and losses is larger than zero. These games differ from zero-sum games in that there is no single optimal strategy and no predictable outcome. It also sometimes fosters cooperative behavior between agents in addition to competitive elements.

Formulation of architecture problems in a game-like form is required if one would like to apply RL methods to them. However, even before diving into RL or implementation specifics, we could do a preliminary analysis of such games formulation problems. First of all, a significant amount of the applications tending to capture the essence of real-world architecture or engineering problems will be classified as non-zero-sum games. There is no winning or losing side in the default generalized formulation of optimal floor plan layout problems.

Furthermore, the actual computation of game gains and losses could be non-linear, making the decision on the game type quite complex. Therefore, we propose an approach where we treat *all* architecture problems by default as non-zero-sum problems. Methods required to solve non-zero-sum games could potentially solve zero-sum games as well, while vice-versa operation will be more complex.

Game design task completion and beyond

How do we define a game end or a task completion in general? It is the open-ended nature of design that represents the indeterminacy of finding the ideal solution to any free-given problem. The definition of game completion poses a very practical problem when it comes to defining our games and goals in simulations. However, there is potentially no way to determine if a certain problem is solvable at all due to Gödel's Incompleteness Theorems (Raatikainen, 2013) and Tuning completeness in general, and halting problem ('Halting problem', 2022) in particular. Therefore, we must limit the definitions of our games' goals. As a rule of thumb, we can use the following heuristic: if we can solve such a problem, AI can solve it too. This thinking pattern can definitely use survivorship bias ('Survivorship bias', 2022) - due to the human mind's complexity and power, we constantly underestimate the real complexity of tasks we're solving. For humans, the problems (or games in the case of this paper) seem simple to solve. However, AI algorithms may find this challenging.

Similarities between RL and other optimizations algorithms

Optimization algorithms are widely used in production in design, architecture, engineering (Vukorep and Kotov, 2021). There are various methodologies and techniques in ML, but a lot of them share conceptually similar ideas. In particular, we would like to mention the concept of the *fitness function* in Genetic Algorithms (GA) and swarm intelligence. GA is intended to evaluate each entity based on some target parameters. As an example in architecture, it is often used to optimize building shape & form vs energy or lighting properties. There's a clear similarity between the fitness function and *reward function* from RL. However, the actual use of this concept is drastically different. GA is computing all the fitness landscape every use-time from scratch, finding the Pareto front and eventually producing the required results. In contrast to this, RL agents are not trained every time from scratch, and this is a significant strategic advantage over GA. One could imagine having one-time use airplanes vs multi-time use as an analogy to this difference. In general and on average, No Free Lunch Theorems (Wolpert and Macready, 1997) state that there is no advantage, but in our particular application domain, it's certainly better not to start every single time from scratch. Trained RL agents are free from this problem. Additionally, because they do not require active training, deployment is easier, since the target machine does not need to be as powerful as the computers used to train ML models.

RL in existing research

Apart from some usual RL applications such as robot controls, there are other applications, which we could also consider for design areas. For instance, the state-of-the-art chip design and engineering (Mirhoseini *et al.*, 2021). This is particularly relevant for us in the context of research because chip design is conceptually very close to any layout and/or structural design. Visually close-up pictures of hardware chip circuits are even comparable to city structures and their spatial and connectivity problems. Another interesting solution is using RL to implement optimal fire evacuation paths (Tian and Jiang, 2018), or finding an optimal policy for shape grammars (Teboul *et al.*, 2011). This last paper is particularly interesting since it combines different concepts such as shape grammar and RL.

RL in architecture applications

In the paper, *An Academy of Spatial Agents report* (Veloso & Krishnamurti, 2020) the

authors successfully attempt the use of a double-deep Q-network (DDQN) combined with a dynamic convolutional neural network (DCNN) with multi-agent deep reinforcement learning (MADRL) in order to solve multi-goal problems (shape, area, adjacency, etc.). An off-policy DDQN algorithm is implemented, in contrast to ours on-policy Proximal Policy Optimization (PPO). Furthermore, the advantage of or our Unity-based approach is that it does not require Rhino/Grasshopper to simulate the environment and completion. Their vision incorporates a lot of conceptual understanding by default. Such as computation of adjacency, connectivity, space, and so on. Using such shortcuts makes the overall problem much easier to converge for an AI agent and makes perfect sense for practical applications. However, to create more general AI solutions, we consider using less pre-computed knowledge information for agent feed input, thus forcing it to understand those problems on a deeper level.

In *Reinforcement Learning for Sequential Assembly of SL-Blocks - Self-interlocking combinatorial design based on Machine Learning* (Wibranek, 2021) the author presented a method to design complex patterns with assembled blocks using the dry-joint assembly method. In the study, RL was used to find optimal joint combinations and structures. The paper shows that RL with PPO algorithm performs better than GA and naive greedy algorithms. In the given research RL was trained from zero every time to find a form. So, it was merely downscaled to some optimization algorithm, like GA used in the comparison.

Architecture Gridworld (Kotov & Vukorep, 2021) proposed a way to model and solve abstract architecture problems in minimal forms (e.g., division of complex skill sets into parts and training them separately to gain complex knowledge) using gridworld representation. Although it was presented in the paper, it presents a challenge due to the definition of those problems in RL, and in particular, due to the use of sparse reward functions and can be mitigated via curriculum learning.

METHODOLOGY

The game categories and case studies are a novel development for this investigation. The games are not only conceptualized as single case studies, but as testing environment for different RL algorithms. Benchmarking different algorithms and comparing their performance on a standardized dataset like MNIST is a common practice in supervised learning. The current research in the field of architectural RL lacks standardized benchmark environments, so we have attempted to transfer this process to the architecture field. Case studies are results of teaching seminars and are demonstrating the plausibility of categories.

In contrast to Architecture Gridworld (Kotov & Vukorep, 2021), the games in this work are set up within a three-dimensional environment. Other difference is that we use dense and semi-dense reward functions instead of just only sparse. For some problems and applications, it may help in the scope of speed of convergence and be generally more practical and intuitive, due to more direct connection between reward function and resulting actions of an agent.

Game categories

For the selection of the games, the following aspects were taken into consideration. For the game to be solvable, an architectural problem must be made quantifiable in the form of a reward function. While performance criteria like stability or energy consumption are easier to work with, qualitative criteria like aesthetics are more difficult to break down, therefore we ignored such subjective tasks.
With the selected three game categories and studies, we wanted to address main application types both in terms of architecture and RL. The games developed for this investigation prove the applicability of RL in different scales: urban, architectural cubature, and building services, acknowledging that this list can be expanded. From the RL point those games are composed differently using different reward functions and distinctive action-space types. These games are specifically set

Table 1
Case studies

Simulation game	Adjacent application area	Reward function	Environment
Urban plan matching	Urban planning / Architecture	Semi-sparse	Discrete
Infrastructure networks	Engineering / Interior design	Sparse	Discrete
Block physical playground	Engineering	Dense	Continuous

up to produce spatial solutions, require volumetric reasoning and/or understanding concepts such as connectivity.

Limitations

The development of a stable simulation for each study case requires significant resources, therefore we were limited in a total amount of resulting games. In fact, creation of a full simulation is equal to gathering dataset. However, we believe that resulting case studies cover an acceptable range of variability and complexity to prove that RL can help in some aspects of architectural problem solving.

Testing

The testing of the environments was conducted with ML-Agents for the Unity 3d engine. However, other software configurations are conceivable via standardized RL interface. Our testing framework included tests for stability, variability of combinations and convergence of AI models using mentioned case studies.

RESULTS

This chapter presents three different game categories with several case studies that use different game approaches, such as autonomous infrastructure planning, optimal zoning solutions, and physical-based structural predictions. The concept is based on the understanding that an architectural design task can be decomposed into smaller sub-problems. Different application categories feature different problems, and the formulation of the problem for each of them is unique.

Case studies

The case studies proposed in this paper were selected within this line of thought and prove to be solvable, abstract, game-like simulations. They were intentionally selected from very different scales and architectural design stages. Urban plan matching represents one simplified spatial planning part, block physical playground represents structural and conceptual design, and Infrastructure networks represent construction (Table 1). It is understood that each game by itself, with the exception of the Infrastructure networks, provides ideas rather than applicable solutions for architectural practitioners at their current stage. This is because each game comes with a set of limitations, which can be explained by the basic reward functions. Urban plan matching functions as a simplified tool for city consolidation and does not take into account the loss of green space, increased traffic and air quality degradation, animal habitation, parks and impervious surfaces, building typology, and urban heat islands, as these phenomena are not included in the reward function. Further research needs to show whether these phenomena can also be formulated as solvable game-like simulations and used to refine the agent through meta-learning (Yu et al., 2021) to achieve more generalizing solutions, similar to multi-objective optimization.

Urban plan matching

Concept & goals

This game is a proof-of-concept augmentation of the city fabric. Existing cities often feature gaps between the buildings and other city entities. The agent will attempt to augment those lots according to their surroundings while keeping the inner spaces free.

Rules of the simulation

The cityscape is randomly generated with different spatial configurations, thus giving the agent enough diversity to generalize different combinations and concepts. The agent is naturally learning the concept of building lines and open spaces in the city. This game offers a dense reward system. Agent gets positive feedback for positioning buildings inside some height-range of the surrounding buildings (Figure 1) and filling the gaps in city fabric (Figure 2).

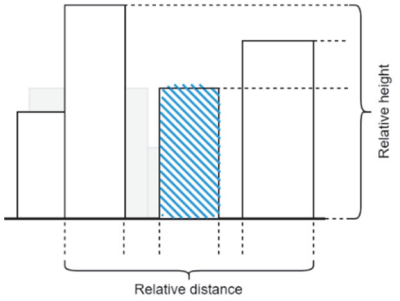


Figure 1
Analysis Attention Diagram

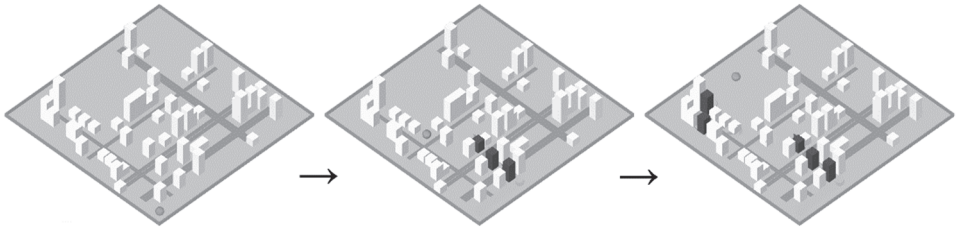
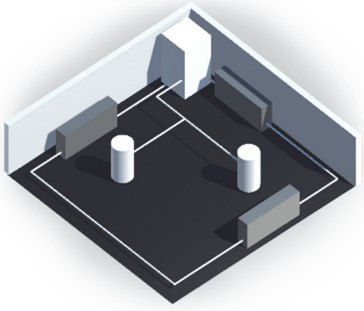


Figure 2
Solution progress

Infrastructure networks

Concept & goals

The problem of finding an optimal solution for infrastructure systems is a very practical one that requires a lot of attention, but can sometimes become routine. Solving it would enable many architects and engineers to automate these processes. This game's objective is a simplified network infrastructure consisting of closed and non-closed loops. It mimics heating and electric systems respectively (Figure 3).



Rules of the simulation

Agents can place blocks freely, connecting target appliances with respective network types. After the placement is done, the evaluation starts. There is a sparse reward function in this game, which makes it difficult for the policy to converge to optimality. The game simulation size is playing a significant role in the rise of complexity. For the sake of simplification, we used only planar configurations. The solution is evaluated in the end of episode, based on performance and connectivity of targets.

Figure 3
Infrastructure networks simulation

Block physical playground

Concept & goals

With this simulation we're testing AI capabilities to combine different blocks into physically stable and stapled structure. Since blocks have different spatial, rotational, and physical configurations/qualities, the task has some complexity. It formulates a first step towards designing more sophisticated forms by stappling irregular elements.

Rules of the simulation

Using a random initialization, we're able to achieve high variation of generated configurations. The Unity physics engine allows us to test results structures for physical stability. The game features two modes for the agent. *Editing mode*: disabling physics for the free placing of the block. *Evaluation mode:* enabling physics for the evaluation of stability and height, when all object reach their assumed stable positions (Figure 4). This is semi-dense reward function, because feedback for an agent is divided in active rewards on physical interaction and post-hoc analysis of overall stability and height ratings.

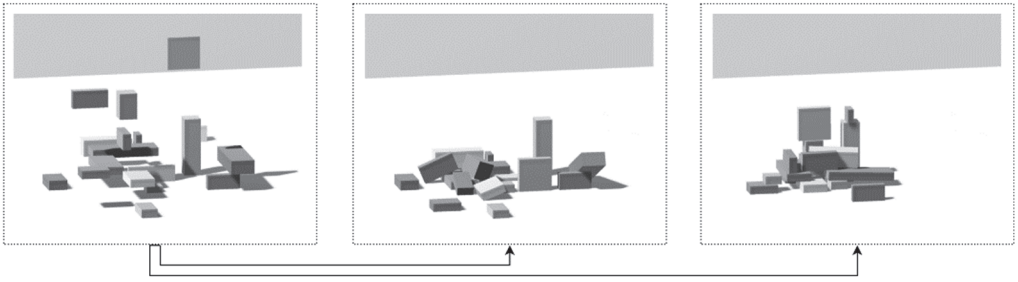


Figure 4
Block Physical Playground Simulation

a. Editing Mode: physics disabled
b. Evaluation Mode: physics enabled; solution not stable; low height
c. Evaluation Mode: physics enabled; solution stable; medium height

DISCUSSION AND CONCLUSIONS

It is likely that future architects will use pre-trained models rather than training their own, and this research lays the foundation for the broader application of RL in architecture. The paper presents a method for decomposing some architectural tasks as well as a 3D testbed for evaluating various ML approaches to solving these problems. While the case studies need to be developed further, they demonstrate the breadth of possibilities on different scales.

The limitation regarding the application of *Block physical playground* is its current high level of abstraction. However, the game proves that its agents are capable of using Unity's built-in Nvidia PhysX engine to account for stability when vertically assembling blocks. Further research could lead to a full-fledged masonry agent or dome builder. Potentially not only automating traditional methods but also developing truly novel assembly methods.

Infrastructure networks are considered to be an overt opportunity for architecture. Further research needs to compare agent-based piping with other pathfinding algorithms, and integration with BIM models and conventional software used in the AEC industry would impose further limitations.

The Unity game engine proved to be an excellent environment for using RL in combination with 3D geometry because of its high computational performance, the ML-Agents toolkit's state-of-the-art implementation of ML libraries through Anaconda, and cross-platform compatibility. There is, however, a disadvantage to Unity when compared to common CAD software: the lack of transformation operations for 3D geometry.

The code for the games and ML configuration files and models is available in our public GitHub repository (link referenced).

REFERENCES

Kotov, A. and Vukorep, I. (2021) 'Gridworld Architecture Testbed', in Stojakovic, V and Tepavcevic, B (eds.), Towards a new, configurable architecture - Proceedings of the 39th eCAADe Conference - Volume 1, University of Novi Sad, Novi Sad, Serbia,

8-10 September 2021, pp. 37-44. CUMINCAD. Available at: http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_252 (Accessed: 21 March 2022).

Rudin, C. and Radin, J. (2019) 'Why Are We Using Black Box Models in AI When We Don't Need To? A Lesson From an Explainable AI Competition', *Harvard Data Science Review*, 1(2). doi:10.1162/99608f92.5a8a3a3d.

Silver, D. *et al.* (2021) 'Reward is enough', *Artificial Intelligence*, 299, p. 103535. doi:10.1016/j.artint.2021.103535.

Vukorep, I. and Kotov, A. (2021) 'Machine learning in architecture: An overview of existing tools', in *The Routledge Companion to Artificial Intelligence in Architecture*. Routledge.

Gunning, D. and Aha, D. (2019) 'DARPA's Explainable Artificial Intelligence (XAI) Program', *AI Magazine*, 40(2), pp. 44–58. doi:10.1609/aimag.v40i2.2850.

Arrieta, A.B. *et al.* (2019) 'Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI', *arXiv:1910.10045 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1910.10045 .

Heuillet, A., Couthouis, F. and Díaz-Rodríguez, N. (2020) 'Explainability in Deep Reinforcement Learning', arXiv:2008.06693 [cs] [Preprint]. Available at: http://arxiv.org/abs/2008.06693 (Accessed: 15 February 2022).

Silver, D. *et al.* (2016) 'Mastering the game of Go with deep neural networks and tree search', *Nature*, 529, pp. 484–489. doi:10.1038/nature16961.

Raatikainen, P. (2013) 'Gödel's Incompleteness Theorems'. Available at: https://plato.stanford.edu/archives/spr2022/entries/goedel-incompleteness/ (Accessed: 21 March 2022).

'Survivorship bias' (2022) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Survivorship_bias&oldid=1075611402 (Accessed: 30 March 2022).

Wolpert, D.H. and Macready, W.G. (1997) 'No free lunch theorems for optimization', *IEEE Transactions on Evolutionary Computation*, 1(1), pp. 67–82. doi:10.1109/4235.585893.

Mirhoseini, A. *et al.* (2021) 'A graph placement methodology for fast chip design', *Nature*, 594(7862), pp. 207–212. doi:10.1038/s41586-021-03544-w.

Tian, K. and Jiang, S. (2018) 'Reinforcement learning for safe evacuation time of fire in Hong Kong-Zhuhai-Macau immersed tube tunnel', *Systems Science & Control Engineering*, 6(2), pp. 45–56. doi:10.1080/21642583.2018.1509746.

Teboul, O. *et al.* (2011) 'Shape grammar parsing via Reinforcement Learning', in *CVPR 2011*. *CVPR 2011*, pp. 2273–2280. doi:10.1109/CVPR.2011.5995319.

Veloso, P. and Krishnamurti, R. (2020) 'An Academy of Spatial Agents', p. 11.

Wibranek, B. (2021) 'Reinforcement Learning for Sequential Assembly of SL-Blocks - Self-interlocking combinatorial design based on Machine Learning', in *Stojakovic, V and Tepavcevic, B (eds.), Towards a new, configurable architecture - Proceedings of the 39th eCAADe Conference - Volume 1, University of Novi Sad, Novi Sad, Serbia, 8-10 September 2021, pp. 27-36*. CUMINCAD. Available at: http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_247 (Accessed: 23 March 2022).

Yu, T. *et al.* (2021) 'Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning', *arXiv:1910.10897 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1910.10897 (Accessed: 31 January 2022).

'Halting problem' (2022) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Halting_problem&oldid=1079812058 (Accessed: 31 March 2022).

Repository (2022) *anatolii-kotov/arch_design_simulation*. Available at: https://github.com/anatolii-kotov/arch_design_simulation (Accessed: 1 April 2022).