SPACE GAMES

Competition and collaboration in multi-agent reinforcement learning for automated apartment floor plan generation

L.CHEN,
UNSW, Sydney, Australia
Levin.chen888@gmail.com

Abstract. Space Layout Planning (SLP) is a 60 years old problem with the ultimate solution of automation. Throughout the year, researchers experimented various ways to achieve the final solution, but those solutions do require minor manual adjustments. During this research, the author has discovered many researches that utilises the concept and method of reinforcement learning to encounter the problem of SLP and in this research, a new and young method for the AEC industry which also comes from the discipline of reinforcement learning, will be utilised to solve the problem of SLP. Multi-agent reinforcement learning (MARL) is a newly rising machine learning method to solve complex problems and is not often used in the AEC industry yet, but it has been widely developed in other disciplines, such as robotics, telecommunication and economics to address any complex problems. This project will try to fit MARL into SLP and try to achieve the ultimate solution through the usage of multiple software and plugins. The method involved five user inputs, the boundary outline for the floor plan, the position of the core block which consists of the elevator and fire exits, the corridor, the number of units and size of each unit. The inputs will then be taken and process within various software tools, such as Rhinoceros 6 and its visual scripting plugin Grasshopper, and TianShou python framework. The core objectives for this research is to generate a successful floor plan covering 100% of the given area from the boundary and maximise solar access for every unit generated with a public corridor access. This project will contribute to the grow scholarship of machine learning usage in the AEC industry and the system designed will contribute to automate floor plan generation, especially in mix-used residential apartments.

Keywords. Machine Learning: Multi-agent Reinforcement Learning; Space Layout Planning; Floor Plan Generation.

1. Introduction: (Research context and motivations)

The automated generation of floor plan layouts has been explored by architects and computer scientists alike to address Space Layout Planning (SLP) in various ways for almost 60 years and remains an elusive goal (Lobos & Donath, 2010). Current methods feature a range of drawbacks including the inability to automate for complete land coverage and generated options still requiring manual adjustment. Accepting that a human will always be integral to the process of SLP regardless of automation, there are opportunities to improve the current automated process of layout generation and few approaches to date have explored the value of multi-agent reinforcement learning (MARL) in this process. MARL is a branch of reinforcement learning which is one type of machine learning algorithm. More specifically, MARL involves multiple 'agents' programmed with restricted actions to work competitively and collaboratively and the goal of trying to achieve the highest reward in a defined environment. While MARL is a relatively recent approach for the AEC industry, it has been widely developed in a variety of areas, including robotics, distributed control, telecommunications and economics to address the complexity of the arising problems. (Busoniu & Babuska & De Schutter, 2010)

Accordingly, this research project investigates the role of MARL for generative design processes. Adopting an action research approach, this research project follows a cyclical process of planning, acting, reflecting and revising in collaboration with an industry partner HDR to develop a MARL integrated generative design workflow for SLP. More specifically, the workflow developed in this research uses Rhinoceros 6 with the Grasshopper visual scripting plugin to prepare the input data. The workflow further uses a Python library framework, Tianshou, for multi-agent reinforcement learning and application. The method involves five user inputs: the boundary of the floor plan, the core block including the position and size of elevator and fire exit staircases, number of units and type of each unit that needs to be planned, also the corridor paths that connects all regions together. The boundary lines will be divided into square grids creating cells that have one square metre per cell using Grasshopper plugin. Then through the Ladybug solar analysis, returning a solar value for each cell which will help the user to identify its core block position and corridor paths, exporting all data into a CSV file, passing it and the information of units to a built multi-agent model Python script for a solution.

The core objectives of the developed workflow are to cover all blank cells, to maximise the solar access area of each unit and to connect each unit to a corridor. This research will contribute to the growing scholarship on machine learning approaches for architecture in general and more specifically to the less explored techniques of MARL. Developing alternate automated space planning methods will contribute to enhancing the design process by generating the most optimal floor plans within the shortest time frame, providing extra choices for both designers and clients, clearly showing the possibility of the designed residential apartments. This system is designed to contribute to the space planning problem in mix-used residential apartments.

2. Research Aims

This research project hopes to achieve the ultimate solution of SLP with 3 main objectives:

- Efficiency
  - Full land coverage (land use) for a given boundary
  - Time used to generate one successful plan
- Optimisation
  - Maximise area has solar access during winter
  - Every unit should have access to the public corridor
- Realistic
  - Shape of units
  - How each unit intersect with each other

3. Research Question(s)

How can multi-agent reinforcement learning be more efficient in number of layouts generated and accurate in realistic aspect of plans when applied to unit layout inside of a mix-used residential building?

4. Methodology

Reinforcement learning as a branch of machine learning, it is somehow unique from other branches. Supervised and unsupervised learning are both dependent on the amount and quality of training data that they receive to produce an optimal solution, but reinforcement learning requires the minimum of data to construct an environment that suits the problem, and interact with certain constraints in the environment for an optimal solution through trial and error. From O'Brien, R. 1998, "put simply, action research is "learning by doing" - a group of people identify a problem, do something to resolve it, see how successful their efforts were, and if not satisfied, try again." To put it even more simpler, action research is research process through trial and error which is a fundamental method of problem-solving and the concept that is used in reinforcement learning to allow the machine or agent develop itself to output the best outcomes.

For my research, to explain my proposed process, I will use the AR cycle (adapted from Baskerville. 1999).

1. Diagnosing – The problem to be solved here is the efficiency of unit layout inside of a mix-used residential building in two perspectives. First, the efficiency in land use – to achieve 100% full land coverage. Second, the time used by the solution to generate a successful plan. But to generate a successful plan, two more objectives are required, optimised adjacency and unit plan shape, which means each unit will need natural lighting and access to corridor, also to be generated in a realistic shape.
2. Action Planning – Throughout my research, the space planning problem has been explored by many professionals through different methods. Out of many approaches, evolutionary solver and reinforcement learning interest me the most. However, evolutionary solver was explored frequently, reinforcement learning was decided to be the methodology used as the proposed solution. It is required to build an environment, agents that will learn and the criteria for scoring agent’s solution. The environment is built by dividing given boundary into square grids and allocate rewards to each cell for agents to occupy and earn the rewards for its score. Criteria will indicate how successful each agent is and fails the agent when certain requirements are not met, e.g. if an agent does not occupy any cell that is adjacent to the corridor which shows no access to egress, will fail the criteria and result in a failed generation.
3. Action Taking – By applying the multi-agent reinforcement learning, the prototype will hopefully start to develop itself to suit the criteria and generate successful plans.
4. Evaluating – To judge if the prototype worked successfully and adjust the criteria and rewards for the next iteration.
5. Specifying Learning – Understand which factors are important and crucial for generating successful plans with multi-agent reinforcement learning.

5. Background Research/Literature review

“The procedural generation of a city entails a number of ingredients, each with its specific procedures and generation techniques” Floor planning is an important if somewhat neglected ingredient. (Lopes et al. 2010, p.1) In this paper, the author reviews the research of multi-agent space plan generation. The research will be reviewed in two parts, multi-agent methods for floor planning, and reinforcement learning as a potential solution to the problem.

5.1. MULTI-AGENT METHODS

For modern floor planning, “we hypothesize that a system that can computationally generate vast numbers of design options, respect project constraints, and analyse for client goals, can assist the design team and client to make better decisions." (Das et al 2016, p.106) Autodesk sees "generative design technology delivering the ideal combination of innovation and productivity required to help companies address the challenges of this disruption" (Harvard Business Review & Autodesk, 2018). With the power of machine learning, Autodesk can generate multiple solutions simultaneously using generative design. Multi-agent based modelling that came from the discipline of generative design will be able to share the same characteristics of generate multiple solutions within the shortest frame of time.

An agent is an entity, also a distinctively higher scale software abstraction that defines a complex software unit in an efficient and convenient way (Abar & Theodoropoulos & Lemarinier & O'Hare, 2017). In the context of space planning, each agent will represent one module of the floor plan which can interact with other agents and environment. Modules can be different components in different floor plans, a module can be a unit when constructing a unit layout plan for an apartment floor, it can also be a bedroom when constructing a house floor plan. Modules are classified into two groups, soft and hard, soft module will be able to resize its dimension without changing the required area and by fitting multiple modules into one environment, a multi-agent system for space planning is formed. A multi-agent system will consist with numerous agents sharing the same environment while interacting with the environment, they can also interact with each other to communicate and exchange information for the shared environment. (Schneider & Fischer & Koenig, 2011) By manipulating the multi-agent model, it allows more generative methods to be applied in the environment for a better and more realistic result.

From (Veloso & Rhee & Krishnamurti, 2019) literature review, multi-agent space planning can be categorised into three types, "agents as moving spatial units", "agents that partition space" & "agents that occupy a space", each with a different interpretation of space. First type, agents as moving spatial units, is a very popular solution with packing algorithms and physics engine. Each individual agent represents a module which allows them to interact with each other to achieve the required spatial objectives. Second type involves agents defining their own territory within the space by splitting or partitioning.

The third type agents will allocate themselves through pre-defined grids, cellular automata is an example of the type 3 approach. "Cellular automata consist in their simplest form of a grid cell whose cells change their states depending on the states of their neighbouring cells." (Toffoli & Margolus, 1987) As explored by Toffoli & Margolus, a cellular automata system works with the same principles that a floor plan generator has. By dividing the given boundary or perimeter, it gives a much better or simpler way to consider the interaction that we need among the agents or modules. From (Slager et al. 2008 p.3), “cellular automata models are able to generate complex spatial structures based on relatively simple set of rules.” In type 3 approach, by implementing defined rules, agents will follow the transition rules, occupy certain cells and change their cell states to form their own border against other agents which will result in a floor plan. Multi-agent based modelling has been deeply explored in the field of floor plan generation with aspects of machine learning but reinforcement learning as a branch of machine learning was explored far less times than multi-agent methods.

5.2. REINFORCEMENT LEARNING

Reinforcement learning is an area of machine learning that deals with how to learn which actions to take in a given environment, in order to maximize a given long-term reward (Sutton & Barto 1998). The given environment of space planning will be the boundary perimeter and by dividing the boundary into grids, a cellular automata environment will be formed for agents to use reinforcement learning algorithm to develop themselves. The rewards are then set based on the cell positions and data, e.g. cell contains higher value for sunlight hours will have a higher reward for an apartment agent, and elevator agents will get better rewards if they occupy a cell with low hours of sunlight. As a branch of machine learning algorithm, reinforcement learning also requires training, but it does not need examples unlike supervised learning, it only interacts with the environment for an answer which allows dealing with uncertainty, making it capable to consider more realistic and non-isolated (with noise data) environment. (Ruiz-Montiel et al. 2013) By combining reinforcement learning and multi-agent based model, a new approach is created, multi-agent reinforcement learning (MARL).“MARL aims to provide an array of algorithms that enable multiple agents to learn the solution of difficult tasks, using limited or no prior knowledge.” (Busoniu & Babuska & De Schutter, 2010) It is a relatively new approach/method to evaluate the space planning problem, for a simple environment, it is often made with grids, which is the cellular automata approach to help and maximise the number of solutions for clients to make better decisions.

6. Case Study (Iteration 1)

6.1. USER INPUTS

The initial user input required for this project is the boundary polyline of the floor plan. It will be processed in Rhinoceros 6 and its visual script plugin

Grasshopper. A simple script is created on Grasshopper to divide and construct square grids within the given boundary that each square will be equivalent to an area of one square metre.

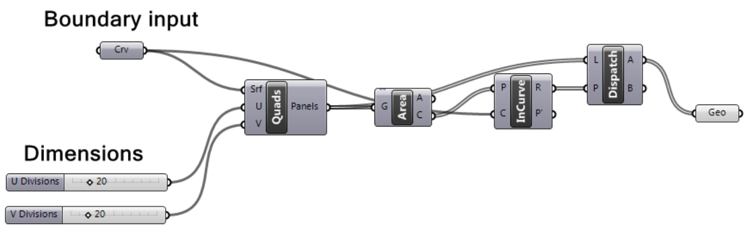


Figure 6.1.1. Script in Grasshopper that divides boundary outline into square grids.

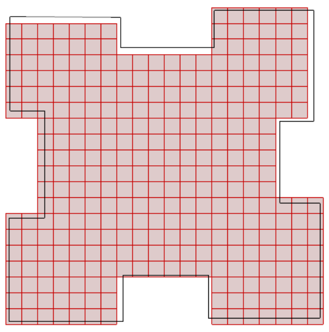


Figure 6.1.2. A floor plan divided into square grids.

For this project, a regular rectangle is used as an initial step to develop the MARL tool. The grids will be duplicated and elevated to construct a multi-level apartment floor plans, then analysed by Ladybug which allows analysis of standard weather data in Grasshopper.

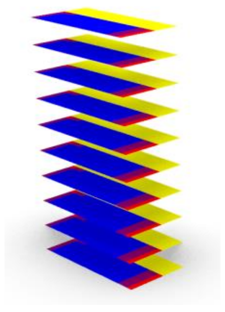


Figure 6.1.3. Perspective view of analysed multi-level floor plans.

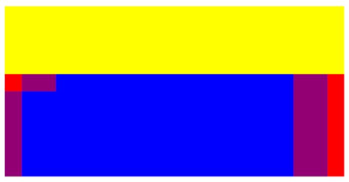


Figure 6.1.4. Top view of an analysed floor plan.

It indicates a value to each cell showing the hours of sunlight that they will receive during winter-time, which helps the user to define the position of their second input, the core block (involves elevator, fire staircase, public corridor).

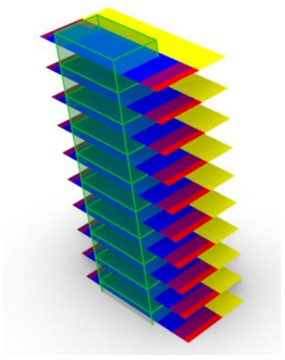
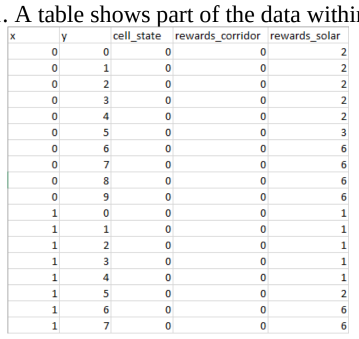


Figure 6.1.5. Perspective view of core block position.

The square grids will form a coordinate system for the data structure, while the solar hours of each square and core block position will be added into the data table as attributes to each square cell. The table will then be exported as a csv file to be used later during the method.

TABLE 6.1.1. A table shows part of the data within the csv file.



x	y	cell_state	rewards_corridor	rewards_solar
0	0	0	0	2
0	1	0	0	2
0	2	0	0	2
0	3	0	0	2
0	4	0	0	2
0	5	0	0	3
0	6	0	0	6
0	7	0	0	6
0	8	0	0	6
0	9	0	0	6
1	0	0	0	1
1	1	0	0	1
1	2	0	0	1
1	3	0	0	1
1	4	0	0	1
1	5	0	0	2
1	6	0	0	6
1	7	0	0	6

6.2. AGENT ACTIONS

The action defines how each agent will grow within the given floor plan boundary. In order to grow while allowing the agents to make a choice in their actions, four actions were planned initially to correspond the four direction that agents can grow. The action was designed to grow based on the numbers of its columns or rows, depends on the direction of growth. For example, when an agent receives a command to grow upwards, the agent will try to grow and occupy the number of its columns of cells upwards. However, the action will be cancelled if any of the cells that the agent tries to occupy has been occupied by other agents or is a core block cell.

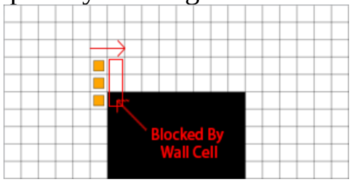


Figure 6.2.1. Showing the action is not viable because it was blocked.

The second version of actions was developed to allow interlocking among the agents' growth. The requirements for agents to grow has changed from requiring the cells that the agent tries to occupy to be 100% blank to over and including 50% as the ratio of blank to occupied.

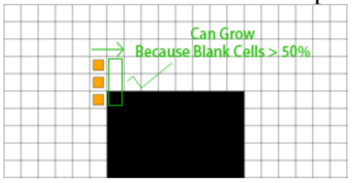


Figure 6.2.2. Showing the action is viable because of the new rules.

6.3. REWARD FUNCTION

The reward function is crucial factor in the reinforcement learning system. It indicates the direction or how the agents will learn within the defined environment. The reward function acts as the dog treat when training a dog to follow orders. In this iteration, the focus in the reward function is to achieve the first objective, full land coverage. The logic is written as shown below:

```
If End:
    If (Blank_Cell_Count == 0):
        Reward = 1
    Else:
        Reward = -1
Else:
    Reward = 0
```

Figure 6.3.1. Iteration 1 reward logic.

6.4. TIANSHOU ENVIRONMENT

TianShou is an open-source, light-weight python library that allows reinforcement learning neural networks training, especially MARL training.
The environment in this project consists of action functions, reward functions, a rendering function, a reset function, and a step function. It acts as a wrapper that packs all functions to allow interactions between the agents and the environment.

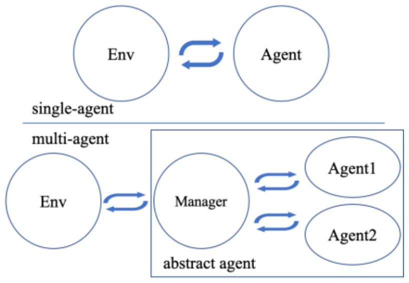
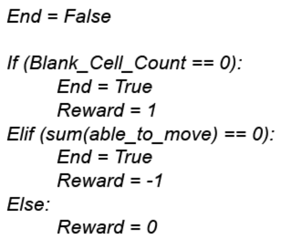


Figure 6.4.1. Relationship among each component in TianShou Framework.

Source: https://tianshou.readthedocs.io/en/master/tutorials/tictactoe.html

6.5. TRAINING AGENTS

Before the training session starts for the agents, there are few preparations to be done. Firstly, any environment requires an “end signal” to tell the neural network to stop this environment and calculate the rewards, allowing the neural network to study the behaviours. There are two “end signals” used in this iteration, the environment will be stopped when every single cell has been occupied or every agent cannot move/make any actions anymore/become locked. For any agent to become locked, it requires the agent to occupy more cells than its max value or it cannot grow in any of the four directions as it gets blocked by other agents. The two “end signals” will result in the final rewards of the training session.



```
End = False

If (Blank_Cell_Count == 0):
        End = True
        Reward = 1
Elif (sum(able_to_move) == 0):
        End = True
        Reward = -1
Else:
        Reward = 0
```

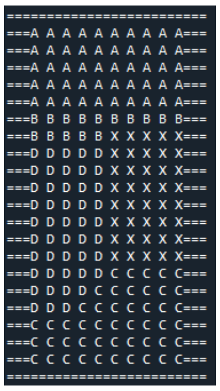
Figure 6.5.1. Combined logic of end signal and reward function.

7. Discussion (Iteration 1)

In this section, two types of policies will reveal their results, the random policy and deep Q-learning neural network (DQN).
Legend:
A, B, C, D = Unit Agent
X = Core block + Corridor
_ = Blank Cells

7.1. RANDOM POLICY OUTCOME

The training starts with utilising random policies for the agents, which means the agents will not be developing any logic for their actions and have random choices for any actions. This process is to test any abnormality or bugs within the environment code, which means this process is for debugging purposes. However, here shows some interesting results from the random policy.



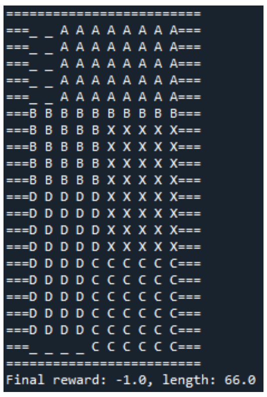
```
=========================
===A A A A A A A A A A===
===A A A A A A A A A A===
===A A A A A A A A A A===
===A A A A A A A A A A===
===A A A A A A A A A A===
===B B B B B B B B B B===
===B B B B B X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D C C C C C===
===D D D D C C C C C C===
===D D D C C C C C C C===
===C C C C C C C C C C===
===C C C C C C C C C C===
===C C C C C C C C C C===
=========================
```

Figure 7.2.1. A random policy result that achieved the objective.

In Figure 7.2.1, a perfect result for the current iteration is shown. This is a result should not be happening with random policy implemented agents but trained policy agents. The results proved the simplicity of the environment, too much restrictions are applied to the agents as there is one type of action, “growth”, and the agent is also restricted by its code as it will become locked for taking any actions when it occupies more cells than its max value.

7.2. DQN POLICY OUTCOME

DQN is a reinforcement learning algorithm that is based on a value called Q-value. The algorithm will generate a Q-value corresponds to each action, as a higher Q-value means that action has more chances to earn more rewards for the agent.



```
=========================
===_ _ A A A A A A A A===
===_ _ A A A A A A A A===
===_ _ A A A A A A A A===
===_ _ A A A A A A A A===
===_ _ A A A A A A A A===
===B B B B B B B B B B===
===B B B B B X X X X X===
===B B B B B X X X X X===
===B B B B B X X X X X===
===B B B B B X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D D X X X X X===
===D D D D C C C C C C===
===D D D D C C C C C C===
===D D D D C C C C C C===
===D D D D C C C C C C===
===D D D D C C C C C C===
===_ _ _ _ C C C C C C===
=========================
Final reward: -1.0, length: 66.0
```

Figure 7.2.1. A DQN policy result failed to achieve the objective.

From Figure 7.2.1, a result has been shown that the policy has failed. In comparison with the random policy result, DQN policy performed worse to random decisions made random policy. This reinforces the need of freedom for the agents in two perspectives, more types of action and less restriction to the agent.

7.3. CHANGES FOR ITERATION 2

- Add another type of action for the agents
- Removing the “locked” status when the agents occupies more cells than its max value and punish them in the reward function for exceeding the max value.

8. Case Study (Iteration 2)

8.1. AGENT ACTIONS

Another type of action is added to give more choices and freedom to the agent to account the problem mentioned in first iteration’s discussion. The action added is to “withdraw” the occupied cells. Same as the “growth” action, “withdraw” can also be called in four directions, while the requirements for an agent to “withdraw” is simpler. It requires a shape with at least two rows to do a vertical “withdraw” action (Top/Bottom), and at least two columns to do a horizontal “withdraw” action (Left/Right). The reason for these two requirements is to forbid the decision for an agent to eliminate itself from the environment.

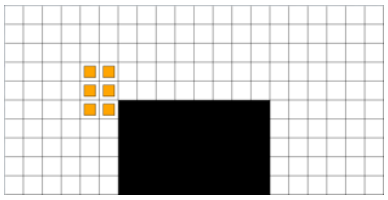


Figure 8.1.1. A 2×3 area allows withdraw vertically and horizontally.

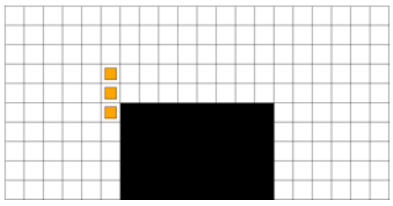


Figure 8.1.2. A 1×3 area allows only vertical withdraw.

8.2. REWARD FUNCTION

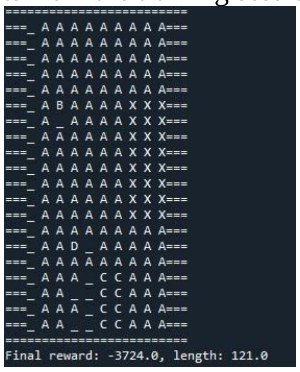
The reward function has developed to reward and punish based on multiple objectives simultaneously. In addition to the objective of full land coverage, solar access, corridor access and unit size are also in calculation for rewards. The added objectives can lead into two more aspects for consideration in the reward function, the number of window cells and the ratio between the number of window cells and corridor-access cells. The number of window cells is an important factor for solar rewards as it allows natural sunlight to pass through into the unit, hence it effects the solar rewards directly.
The logic is written as below:

```
for i in range(x):
    for j in range(y):
        if  Board[i][j] = agent_number:
            counter += 1
            cell_rewards += solar[i][j] + corridor[i][j]
            if (i == x) or (i == 0) or (j == y) or (j == 0):
                window_count += 1
            if corridor[i][j] != 0:
                corridor_count += 1

if counter < min:
    rewards = counter - min
elif counter > max:
    rewards = max - counter
else:
    rewards += cell_rewards

if (window_count == 0) or (corridor_count == 0):
    rewards = -1
if corridor_count > window_count:
    rewards = -1
```

Figure 8.2.1. Iteration 2 reward logic.

9. Discussion (Iteration 2)

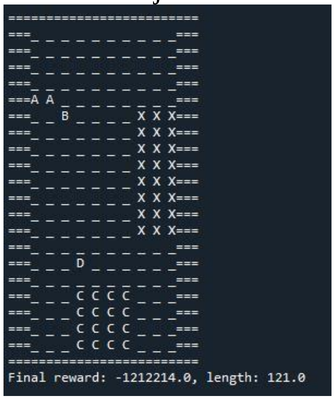
There are 3 different results from the training sessions.



```
========================
===_ A A A A A A A A A===
===_ A A A A A A A A A===
===_ A A A A A A A A A===
===_ A A A A A A A A A===
===_ A A A A A A A A A===
===_ A B A A A A X X X===
===_ A _ A A A A X X X===
===_ A A A A A A X X X===
===_ A A A A A A X X X===
===_ A A A A A A X X X===
===_ A A A A A A X X X===
===_ A A A A A A X X X===
===_ A A A A A A X X X===
===_ A A A A A A A A A===
===_ A A D _ A A A A A===
===_ A A A A A A A A A===
===_ A A A _ C C A A A===
===_ A A _ _ C C A A A===
===_ A A A _ C C A A A===
===_ A A _ _ C C A A A===
========================
Final reward: -3724.0, length: 121.0
```

Figure 9.1.1. Result from first training session

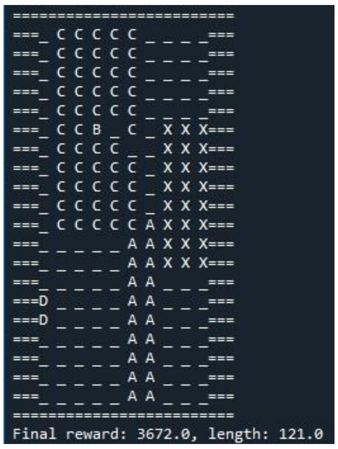
From Fig 9.1.1 shows a scene of agent "A" "bullies" all other three agents. It indicates a problem within the rewards function, as it did not give enough punishments to agent "A" for exceeding the number of cells occupied while trying to achieve the objective of no blank cells.



```
=========================
===_ _ _ _ _ _ _ _ _ _===
===_ _ _ _ _ _ _ _ _ _===
===_ _ _ _ _ _ _ _ _ _===
===_ _ _ _ _ _ _ _ _ _===
===A A _ _ _ _ _ _ _ _===
===_ _ B _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ X X X===
===_ _ _ _ _ _ _ _ _ _===
===_ _ _ D _ _ _ _ _ _===
===_ _ _ _ _ _ _ _ _ _===
===_ _ _ C C C C _ _ _===
===_ _ _ C C C C _ _ _===
===_ _ _ C C C C _ _ _===
===_ _ _ C C C C _ _ _===
=========================
Final reward: -1212214.0, length: 121.0
```

Figure 9.1.2. Result from second training session

Fig 9.1.2 shows a scene of a changed reward functions, all four agents are not growing, can be caused by the reward function over-punishes the agents during the training session.



```
=========================
===_ C C C C C _ _ _ _===
===_ C C C C C _ _ _ _===
===_ C C C C C _ _ _ _===
===_ C C C C C _ _ _ _===
===_ C C C C C _ _ _ _===
===_ C C B _ C _ X X X===
===_ C C C C _ _ X X X===
===_ C C C C C _ X X X===
===_ C C C C C _ X X X===
===_ C C C C C _ X X X===
===_ C C C C C A X X X===
===_ _ _ _ _ A A X X X===
===_ _ _ _ _ A A X X X===
===_ _ _ _ _ A A _ _ _===
===D _ _ _ _ A A _ _ _===
===D _ _ _ _ A A _ _ _===
===_ _ _ _ _ A A _ _ _===
===_ _ _ _ _ A A _ _ _===
===_ _ _ _ _ A A _ _ _===
===_ _ _ _ _ A A _ _ _===
=========================
Final reward: 3672.0, length: 121.0
```

Figure 9.1.3. Result from third training session

Fig 9.1.3 shows a much well-trained agent, "C", as it develops the idea of size and solar access to achieve the highest rewards, but still not perfect for an automated floor plan generation use.

Throughout the results, two major changes will be made in the following iteration development:

- Rewards Function
- MARL base library – TIANSHOU

As shown from the results, the current rewards function cannot train the agents perfectly as this project is focused on MARL. It did not satisfy to train all four agents within the same environment, but only one agent is trained instead. This leads into the second major change, the base library – TIANSHOU. TIANSHOU is a light-weighted and beginner- friendly MARL python library. However, from its developer's description, the feature of training multiple agents in conjunction has yet to be tested. The feature still has a possibility to be successful but in the current stage, it is not the perfect tool to continue this project. A replacement of the base library will be another reinforcement learning library, RLlib, which is a more developed open source library providing more advanced tools to train agents.

During this research, many researches that utilises reinforcement learning as the method has been found, but to use MARL as a solution for SLP has yet to be discovered on the internet publicly. This research can be the first step of utilising MARL to solve problems in the AEC industry which already has performed its abilities in robotics and economics. To extend its abilities into the AEC industry, from this project, it maybe limited by the performance of the library and the knowledge base of the author. However, the potential for MARL to shine is not only within the scale or an apartment floor plan. It can be also utilised in a master planning scale, such as town planning and urban planning.

10. Conclusion

Developing reinforcement learning methods for Space Layout Planning can generate a more optimised and efficient residential apartment floor plan. In this project, this has been explored with multi-agent reinforcement learning (MARL) which is based off a branch of machine learning, called reinforcement learning, which involves the agent with restricted actions to learn the way of how to achieve the highest rewards in a built digital environment. While MARL, has the feature of multiple agents to learn how to function collaboratively and competitively within the same virtual environment. From the results, the base library to use for this MARL method may not be the best choice in long term but a perfect choice for this 10-week project. The results may show unsatisfactory outcomes, but there is a huge potential for this method to be successful, not just in floor plan generation. The method can also be applied in town planning, urban planning, or any master planning projects. With the same concept of this project, the agents

will be placed in an environment to fight for their most desired areas, but many objectives will be alternated to suit the different projects.
This can be the very first step of using MARL in AEC discipline which could lead to more advanced technologies, such as digital twin, to simulate the city growth and plan ahead of time to overcome problems like over-populated areas. The MARL has many more uses in the AEC industry, this project has only shown a small percentile of its capability, it may even build a fully AI-driven virtual city, but it will take further development and training to evaluate a perfect outcome.

Acknowledgements

I would like to thank my industry partner, HDR (Leo, Jeremy, Stefano and Tina), for their valuable suggestions and guidance throughout this research project.
I would also like to thank the developer of Tianshou, Jiayi Weng, for his technical support and help for his understanding about TianShou.
Finally, I would like to thank my tutors, Nicole, Hank and Daniel, for their guidance and suggestions in this research project.

References

Schneider, Sven & Fischer, Jan-Ruben & Koenig, Reinhard. (2011). *Rethinking Automated Layout Design: Developing a Creative Evolutionary Design Method for the Layout Problems in Architecture and Urban Design.*

Busoniu, Lucian & Babuska, Robert & De Schutter, Bart. (2010). *Multi-agent Reinforcement Learning: An Overview*.

Veloso, Pedro & Rhee, Jinmo & Krishnamurti, Ramesh. (2019). *Multi-agent Space Planning: A Literature Review*.

Das, Subhajit & Day, Collin & Hauck, Anthony & Haymaker, John & Davis, Diana. (2016). *Space Plan Generator: Rapid Generation & Evaluation of Floor Plan Design Options to Inform Decision Making*

Lopes, Ricardo & Tutenel, Tim & Smelik, Ruben M. & de Kraker, Klaas Jan & Bidarra, Rafael. (2010). *A Constrained Growth Method For Procedural Floor Plan Generation*

Ruiz-Montiel, Manuela & Boned, Javier & Gavilanes, Juan & Jiménez, Eduardo & Mandow, Lawrence & Pérez De La Cruz, José-Luis (2013). *Design with shape grammars and reinforcement learning*

Sutton, Richard S. & Barto, Andrew G. (1998). *Reinforcement Learning: An Introduction*

Slager, Kymo & Vries, B. & Jessurun, A.K. (2008). *Methodology to generate landscape configurations foruse in multi-actor plan-making processes.*

Abar, Sameera & Theodoropoulos, Georgios & Lemarinier, Pierre & O'Hare, Gregory. (2017). *Agent Based Modelling and Simulation tools: A review of the state-of-art software. Computer Science Review.*

Toffoli, Tommaso & Margolus, Norman. (1987). *Cellular Automata Machines. Complex Systems.*

Harvard Business Review & Autodesk. (2018). *The Next Wave of Intelligent Design Automation.* E-book.