Centralized control for multi-agent RL in a complex Real-Time-Strategy game

Roger Creus Castanyer
Mila Québec
Université de Montréal
roger.creus-castanyer@mila.quebec

Abstract

Multi-agent Reinforcement learning (MARL) studies the behaviour of multiple learning agents that coexist in a shared environment. MARL is more challenging than single-agent RL because it involves more complex learning dynamics: the observations and rewards of each agent are functions of all other agents. In the context of MARL, Real-Time-Strategy (RTS) games represent very challenging environments where multiple players interact simultaneously and control many units of different natures all at once. In fact, RTS games are so challenging for the current RL methods, that just being able to tackle them with RL is interesting. This project provides the end-to-end experience of applying RL in the *Lux AI v2* Kaggle competition, where competitors design agents to control variable-sized fleets of units and tackle a multi-variable optimization, resource gathering, and allocation problem in a 1v1 scenario against other competitors. We use a centralized approach for training the RL agents, and report multiple design decisions along the process. We provide the source code of the project: `https://github.com/roger-creus/centralized-control-lux`

1 Introduction

Reinforcement learning (RL) has achieved great success in many complex domains, such as video games and robotics [Mnih et al., 2013]. However, most of the work in RL assumes single-agent settings, where the agent interacts with a static or predetermined environment. In recent years, there has been growing interest in multi-agent RL (MARL), where multiple agents interact with a shared environment and learn to coordinate or compete. Many of the most recent and impactful RL successes use MARL. For example, OpenAI Five [Berner et al., 2019], a team of five AI agents, defeated a team of professional human players in the complex video game Dota 2. Similarly, AlphaStar [Vinyals et al., 2019], a system developed by DeepMind, defeated professional human players in the StarCraft II game. AlphaGo [Silver et al., 2016], another DeepMind system, defeated the world champion human player in the board game Go. Recently, Meta AI published Cicero [FAIR], an AI agent that outperforms humans in the game of Diplomacy. These successes demonstrate the potential of MARL and the importance of developing effective strategies for coordination and cooperation among agents in complex environments.

Due to the computational requirements for training such complex systems, the majority of research institutions can't contribute to MARL with significant empirical evidence. However, there have been many recent advances in the design of affordable but complex environments which pose similar challenges to StarCraft and Dota II [Huang et al., 2021, Chen et al., 2023, Suarez et al., 2019]. Concretely in terms of coordination between units, size of the joint action space, non-stationary dynamics and non-Markovian rewards, and hence enable the research community in MARL to empirically study new techniques and algorithms. Several competitions have been hosted recently

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

in such affordable environments because it remains challenging to tackle these with RL due to the implicit complexity of MARL and RTS games. In this work, we participate in the *Lux AI v-2* Kaggle competition [1] and provide the end-to-end experience from defining the environment observation and action spaces to training RL agents via self-play. The *Lux* environment can be seen in Figure 1. Section A and the environment specifications[2] provide more details on Lux.

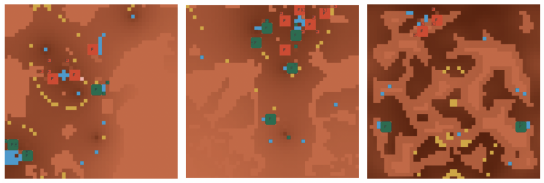

Figure 1: Three different 48x48 maps of the *Lux AI v2* environment. Player1 and Player2 compete for resources in pseudo-randomly generated maps. Yellow cells indicate *ore*, blue cells indicate *ice*, brown cells indicate *ruble*, 3x3 squares with the letter F are *factories*, and little squares with the letters L and H are *light* and *heavy* robots respectively.

The Lux environment poses several key challenges: *(i)* Lux provides entity observations which need to be pre-processed before feedxing them to an RL agent (e.g. see a sample observation here); *(ii)* at each turn, each player needs to issue primitive actions for variable-sized fleets of units; *(iii)* the units have different action spaces and specifications; *(iv)* resources are sparse and critical for survival; *(v)* there is a high variability of maps and terrains, the number of factories is also different at each game; and finally *(vi)* it's a non-stationary environment because it involves another agent. Arguably, these engineering, algorithmic and theoretical challenges conform to a very complicated environment for RL. In this work, we tackle each one of them and report our empirical experience.

2 Background

In the context of MARL, the Lux environment can be modelled as a Multi-Agent Markov Decision Process (MMDP). An MMDP [Boutilier, 1996, van der Pol et al., 2021] is a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ where $\mathcal{N}$ is a set of m agents, $\mathcal{S}$ is the state space, $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_m$ is the joint action space of the MMDP, $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_m \times \mathcal{S} \to [0, 1]$ is the transition function, and $\mathcal{R} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_m \to \mathcal{R}$ is the immediate reward function. This *decentralized* setup models the coordination between the m agents in the same team. $\mathcal{S}$ is the global state space, and the transition function models the probability of the future global states under the m-dimensional joint action distribution. Similarly, this framework defines a global reward distribution as a function of a single global state and the joint m actions, meaning that the m agents collaborate to maximize a single reward function. Moreover, in a 1vs1 environment like Lux, the opponent's units' actions are part of the state space $\mathcal{S}$, and if the opponent is a learning agent or evolves with time, then the transition distribution $\mathcal{T}$ is non-stationary.

Credit assignment in decentralized MARL is challenging [Chang et al., 2003]. For this reason, several works explore value-decomposition methods [Sunehag et al., 2017] to learn simpler agent-specific value functions or critics [Iqbal and Sha, 2019]. Other works leverage hand-crafted agent-specific reward distributions to independently train each agent [Ye et al., 2020]. However, such works assume a fixed number of agents and aren't feasible in massive environments like Lux where units are constantly destroyed and built. Finally, several works define a *centralized* set-up and use a pixel-to-pixel architecture that leverages multi-agent control in grid-like environments [Huang et al., 2021, Chen et al., 2023, Han et al., 2019]. In this context, the MMDP turns into a standard single-agent problem. However, the size of the action space, the cooperation between units, and the non-stationarity of the transition function remain challenging. In this work, we also use a centralized approach for training our RL agents, which we optimize with Proximal Policy Optimization (PPO) [Schulman et al., 2017]. PPO is an actor-critic policy-gradient-based algorithm, which compared to vanilla REINFORCE or A2C [Grondman et al., 2012], defines a surrogate objective which allows re-using the collected on-policy data for multiple corrected training updates. In Section 3, we describe the model architecture in detail and the PPO training process.

[1] `https://github.com/Lux-AI-Challenge/Lux-Design-S2`

[2] `https://www.lux-ai.org/specs-s2`

3 Methodology

This section defines the components that enable training an RL agent in the Lux environment. The main requirements are defining the state and action spaces, a dense reward distribution, a functional model architecture, and a training pipeline. While these requirements are present in all RL applications, their complexity scales as the environment complexity grows, making RL challenging in RTS games. The following describes our design decisions, which motivation is to turn Lux into a fully-observed MDP that allows training a single, and thus centralized, RL agent.

Observation space. We define the observations by extracting several feature maps containing independent information about the game. The feature maps are stacked together as objects of size $(H \times W \times C)$ where $H = 48$, $W = 48$ is the fixed map size and in our case $C = 17$. These provide information about the positions of the ally and enemy units, factories and resources on the map. Table 1 contains a detailed description of the observation space.

Action space. We define two action spaces, one for the factories (which can create robots and grow lichen) and one for the robots (which can move, dig and transfer resources). The factory's action space is multi-discrete with 2 components: the first one indicates the position of the factory on the map and the second allows 4 actions. The unit's action space is multi-discrete with 6 components: the first one indicates the unit's position on the map and the other 5 define the joint action (e.g. transfer - ice - south). Section A contains detailed descriptions of the action space. Furthermore, invalid action masks [Huang and Ontañón, 2020] are implemented to prevent the agent from learning from output actions that are not going to be executed in the game - the model outputs an action at each cell of the 48x48 map, but the agent shouldn't learn from actions that will not be issued to an actual unit. The action masks are used to effectively set the logits of illegal actions to large negative values so that their log probabilities are 0 and so are their gradients.

Reward distribution. We use reward shaping to define a hand-crafted reward distribution that encourages players to generate water. While players win by growing as much lichen in 1000 game steps, resource requirements are too critical in Lux to have RL agents learn to generate water only from game-winning rewards. To generate water, which is needed for survival and growing lichen, units have to navigate efficiently, dig ice, transfer it to factories and have it refined. Arguably, the latter is a too complex sequence of actions to be discovered from sparse or delayed rewards. A more detailed discussion of the reward distribution can be found in Section D.

Self-play. Prioritized Fictitious Self-play (PFSP) [Vinyals et al., 2019] is implemented to provide a stable training pipeline for the RL agents. Compared to population-based approaches [Jaderberg et al., 2017], in PFSP a single learner agent plays against a pool of previous checkpoints of itself. The pool is implemented as a queue of fixed size, so after every N training updates a new checkpoint is stored and the oldest one is removed. During training, the most recent checkpoint is sampled with a 50% chance and any other one randomly otherwise. Section E discusses self-play in more detail.

Architecture. An actor-critic pixel-to-pixel architecture [Han et al., 2019] is used, which is similar to *Gridnet* [Chen et al., 2023], and trained with PPO. The model architecture is shown in Figure 2. During training, the model computes the joint action probabilities of all valid robot and factory actions by summing their logits. The latter corresponds to multiplications in the space of probabilities which assumes independence between factory and robot actions. The critic is used to compute the returns and advantages, and the PPO updates consist of making the actor's predicted actions with higher estimated advantages more likely.

4 Experiments

In general, RL needs lots of data to work, and MARL is usually orders of magnitude more sample-inefficient [Garnelo et al., 2021, Berner et al., 2019, Vinyals et al., 2019]. Furthermore, the Lux environment is slow, achieving 60 environment steps per second running on a single A6000 GPU. We perform a hyperparameter sweep[3] running many jobs in parallel, each training the agent for 10M environment steps for a total of 1,296 hours of computing. After that, we take the best-performing configuration and run it using 8 RTX8000 GPUs in parallel for 220M environment steps during 124 hours. The report of the. More details can be found in Section F.

[3] `https://api.wandb.ai/links/rogercreus/49pdlt7a`

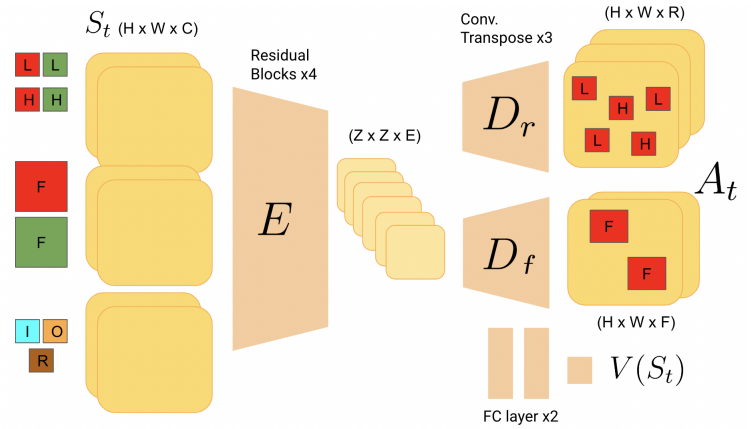

Figure 2: Our pixel-to-pixel architecture. We feed the model with a stack of C feature maps containing information about the units, factories and resources. We use a Residual Network [He et al., 2016] with squeeze-and-excitation layers [Hu et al., 2018] to encode the observations into low-dimensional representations. We feed the representations to three different heads: the critic which outputs a single scalar value representing the state value; the *robot actor* which outputs feature maps with as many dimensions as robot action components; and the *factories actor* which outputs as many actions as factory action components.

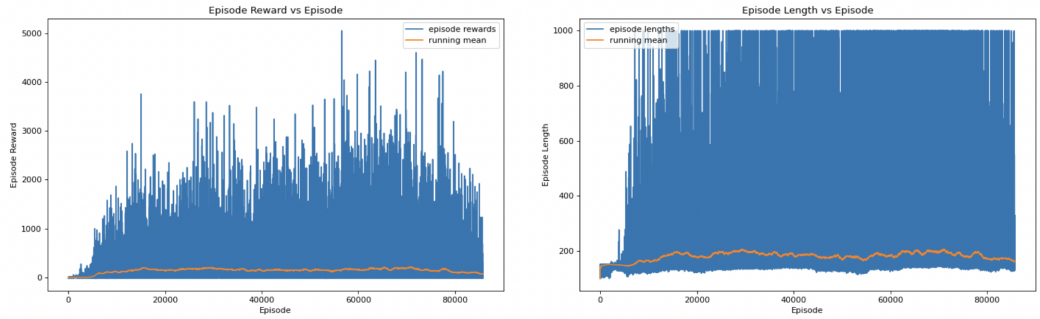

Figure 3: The figure shows the episode reward (left) and episode length (right) during training. The agent is trained for 220M environment interactions. Although the agent learns to gather resources and survive for the 1,000 steps of the game in many episodes, there is a high degree of variance. Since each episode has a significantly different map and a variable number of factories, it is hard for the agent to generalize.

5 Conclusions and Future work

This work defines a centralized training framework for MARL and shows how to train a pixel-to-pixel actor-critic architecture with PPO in a highly challenging environment for RL. We have uncovered several challenges for applying RL in RTS games, justified our design decisions and discussed possible improvements. We have also provided an open-source codebase which can be used as a starting point for similar future competitions [4] and shared the logs for all the experiments[5].

Several lines of work can improve the presented approach, some of which are: *(i)* better definitions of the observation and action spaces; *(ii)* using other architectures whose inductive biases are more appropriate for grid-like RTS games (e.g. avoiding the downscaling bottlenecks [Chen et al., 2023], U-Net [Ronneberger et al., 2015], LSTM [Hochreiter and Schmidhuber, 1997]); *(iii)* implementing population-based-training to shape the learning dynamics in favour of the learner agent [Garnelo et al., 2021]; and *(iv)* scaling up the architecture and enabling highly distributed training.

[4] `https://github.com/roger-creus/centralized-control-lux`
[5] `https://api.wandb.ai/links/rogercreus/49pdlt7a`

References

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *TARK*, volume 96, pages 195–210. Citeseer, 1996.

Yu-Han Chang, Tracey Ho, and Leslie Kaelbling. All learning is local: Multi-agent learning in global reward games. *Advances in neural information processing systems*, 16, 2003.

Hanmo Chen, Stone Tao, Jiaxin Chen, Weihan Shen, Xihui Li, Sikai Cheng, Xiaolong Zhu, and Xiu Li. Emergent collective intelligence from massive-agent cooperation and competition. *arXiv preprint arXiv:2301.01609*, 2023.

Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

Marta Garnelo, Wojciech Marian Czarnecki, Siqi Liu, Dhruva Tirumala, Junhyuk Oh, Gauthier Gidel, Hado van Hasselt, and David Balduzzi. Pick your battles: Interaction graphs as population-level objectives for strategic diversity. *arXiv preprint arXiv:2110.04041*, 2021.

Thore Graepel, Tom Minka, and R TrueSkill Herbrich. A bayesian skill rating system. *Advances in Neural Information Processing Systems*, 19:569–576, 2007.

Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

Lei Han, Peng Sun, Yali Du, Jiechao Xiong, Qing Wang, Xinghai Sun, Han Liu, and Tong Zhang. Grid-wise control for multi-agent reinforcement learning in video game ai. In *International Conference on Machine Learning*, pages 2576–2585. PMLR, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. Gym-μrts: toward affordable full game real-time strategy games research with deep reinforcement learning. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*, pages 2961–2970. PMLR, 2019.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Elise van der Pol, Herke van Hoof, Frans A Oliehoek, and Max Welling. Multi-agent mdp homomorphic networks. *arXiv preprint arXiv:2110.04495*, 2021.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, et al. Towards playing full moba games with deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:621–632, 2020.

Appendix A The Lux-v2 competition

In Lux, both AI agents and scripted bots compete for resources in 1vs1 games. The game has 3 phases: *(i)* the *starting phase*, in which both players bid for who will place a factory first; *(ii)* the *placement phase*, in which starting with the player who won the bid, each player iteratively places a factory at its chosen locations; and *(iii)* the *normal phase*, in which both players issue actions to all the factories and robots to gather resources and grow lichen. In this work, we have heuristically set the *bidding phase* to 0 for both players and the *placement phase* to a strategy that places the factories taking into account the locations of ice, ore, rubble and enemy factories. In this way, in our environment wrapper, each call to *env.reset()* already simulates the bidding and placement phases and returns the first state of the normal phase.

The games last for a maximum of 1,000 turns in which the two players submit actions simultaneously. If a player loses all their factories, the game ends. Otherwise, the player who has more lichen on the map wins.

Factories can either build light or heavy robots (consuming metal and power) or grow lichen (consuming water). Robots can move in 4 directions (consuming power), transfer resources (ice, ore, water, metal and power) in 4 directions, dig resources (ice, ore, rubble), and pick up resources from factories or self-destruct. Furthermore, robots act according to action queues, which allow scripted agents to submit entire action plans in a single step. Updating the action queue of a robot has an additional cost of power, encouraging efficient planning of sequences of actions. In this work, we assume single actions, paying the additional cost in power.

Appendix B Observation Space

Lux originally provides entity observations, which are dictionaries (i.e. JSON files) containing all the information of the current game state. For this reason, we parse this information into a format that can be fed into a deep RL model and describe it in the following:

Table 1: Description of the 17 feature maps that conform the observations fed to the agent. Negative values always indicate properties related to the opponent (e.g. a -200 in unit_ice means an enemy unit has 200 ice in it's cargo. Each of the feature maps is the same size of the map 48x48.

Name	Description	Range	Normalization
is_day	Whether the current turn is day or night	[0, 1]	-
rubble	Amount of rubble in each cell	[0, 100]	100
ore	Whether there is ore in each cell	[0, 1]	-
ice	Whether there is ice in each cell	[0, 1]	-
lichen	Amount of lichen in each cell	[0, 100]	100
is_resource	Whether there is ore or ice in each cell	[0, 1]	-
light_units	Whether there is a light robot in each cell	[-1, 0, 1]	-
heavy_units	Whether there is a heavy robot in each cell	[-1, 0, 1]	-
unit_ice	Amount of ice in the robot's cargo	[-3000, 3000]	3,000
unit_ore	Amount of ore in the robot's cargo	[-3000, 3000]	3,000
unit_power	Amount of power in the robot's battery	[-1000, 1000]	1,000
unit_on_factory	Whether each robot is on top of a factory	[-1, 0, 1]	-
factories	Whether there is a factory in each tile	[-1, 0, 1]	-
factory_ice	Amount of ice in each factory	$[-\infty, \infty]$	150
factory_ore	Amount of ore in each factory	$[-\infty, \infty]$	150
factory_water	Amount of water in each factory	$[-\infty, \infty]$	150
factory_metal	Amount of metal in each factory	$[-\infty, \infty]$	150

Appendix C Action Space

We define a joint action space for both robots and factories as a dictionary (*gym.spaces.Dict()*) in which both *action_space["factories"]* and *action_space["robots"]* are MultiDiscrete objects. A MultiDiscrete space is the multi-dimensional extension of the simple Discrete space [6]. In a MultiDiscrete action space, each component (i.e. dimension) is independent from each other and can have different sizes. The detailed descriptions for both MultiDiscrete action spaces is shown in Tables 2 and 3.

Table 2: Description of the robots' action space

Component	Description	Range
Source Unit	location of the selected unit to perform an action	$[0, h \times w - 1]$
Action Type	NOOP, move, transfer, pickup, dig, self-destruct	[0, 5]
Move param.	up, right, down, left	[0, 3]
Transfer param.	center, up, right, down, left	[0, 4]
Transfer amount	25%, 50%, 75%, 95%	[0, 3]
Transfer resource	ice, ore, power	[0, 2]

Table 2 shows the action space for robts. In this way, an action is obtained by sampling a value from each of the components. However, note how the first component, the *action type*, is the one that determines which action parameters are going to be used. For example, the action *move - right - left - 25% - ore* will execute *move - right* and the action *transfer - left - up - 50% - ice* will execute *transfer - up - 50% - ice*. The actions of *pickup*, *dig* and *self-destruct* don't need any other parameters as robots can only pick up a fixed amount of power, dig just means that the robot will dig whatever resource is in the tile where it's digging and self-destruct is self-explanatory.

[6] `https://www.gymlibrary.dev/api/spaces/#multidiscrete`

Table 3: Description of the factories' action space

Component	Description	Range
Source Unit	location of the selected unit to perform an action	$[0, h \times w - 1]$
Action Type	NOOP, build_light, build_heavy, grow_lichen	[0, 3]

Appendix D Reward distributions

RL agents must master several skills to have a chance to win in Lux, the most challenging one being efficiently generating water: each factory starts with 150 water and metal, and consumes 1 water at each turn. A factory is lost if it runs out of water, and can only generate water by sending robots to dig ice and transfer it to the factories to have the ice refined. If the RL agents don't master this sequence of actions, which requires a lot of efficient exploration, the games during training will last for 150 turns, when both players will lose all their factories. Solving this challenge has been the main stopper in our work, and for this reason, our initial dense reward only rewards robots for digging ice and factories for refining it. Ideally, after the RL agents master this strategy, they need to learn late-game dynamics, in which they have to spend water to grow lichen without running out of it. For this reason, we propose changing the reward distribution after lots of training updates to a sparse reward indicating the lichen amount at the last turn of the game. The latter can create a curriculum for RL agents to master basic skills and transfer them to late-game dynamics.

Appendix E Self-play

Self-play enables shaping the learning dynamics in favour of the learner agents. There is a whole subfield of MARL exploring different strategies for training populations of diverse agents that are helpful to train state-of-the-art agents to play complex video games. Population-based training [Garnelo et al., 2021] is about defining a metric to rank the true skill of agents within a population and a strategy to sample opponents for each agent in the pool at each game. We argue that population-based training has been a key feature of the recent most impactful works in MARL [Vinyals et al., 2019, Berner et al., 2019]. However, it is usually overlooked and is sometimes obscure, because there is a general lack of frameworks and sources to implement such techniques.

In this work, we have also run some experiments using the TrueSkill [Graepel et al., 2007] metric to rank the pool of agent's checkpoints. Instead of using the pool as a queue and keep removing the oldest checkpoints, using such a metric allows us to fairly evaluate whether the old checkpoints that were to be removed were actually less skilled than the others. However, we haven't used such a metric in our sweeping experiments because we haven't been able to scale the ranking function across the parallel jobs used due to our dependency on the external library [7].

Appendix F Experiments

Figure F shows the complete logs of the final PPO run. The architecture in Figure 2 was trained with PPO and run for 220M environment steps, which took 126 hours using 8 Nvidia A600 GPUs. Figure 3 indicates:

- **The value loss increases during training**. The spikes in the value loss are usually a good sign since it means that the agent observes transitions in which it is unable to estimate the value correctly at first. The latter indicates that the agent is exploring (possibly) high-value behaviours. However, at convergence, we would expect the value loss to get closer to 0.
- **The entropy increases at the beginning and decreases during training**. That is usually a good sign and means that the policy converges to a less stochastic behaviour.
- **The policy loss gets closer to 0 during training**. That is usually a good sign and is correlated with the decrease in entropy. It means that the policy changes less abruptly across updates, hence converging to a less stochastic policy.

[7] `https://trueskill.org/`

- **The KL divergence gets close to 0 but spikes abruptly**. The abrupt spikes are often undesired and indicate that the policy takes big learning steps, which can cause forgetting previously learnt behaviours. Ideally, we would like the KL to remain similar during training and close to 0.