An Efficient Open World Benchmark for Multi-Agent Reinforcement Learning

Eric Ye, Natasha Jaques
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
`ericy4@uw.edu,nj@cs.washington.edu`

Abstract

Many challenges remain before AI agents can be deployed in real-world environments. However, one virtue of such environments is that they are inherently multi-agent, and contain experienced expert agents (like humans) who demonstrate useful behaviors. Such behaviors can help an AI agent generalize and apply to new use-cases and scenarios. While this type of social learning ability could improve generalization and human-AI interaction, it is currently difficult to study due to the lack of open-ended multi-agent environments. In this work, we present an environment in which multiple self-interested agents can pursue complex, independent goals. We have developed the first multi-agent version of the Craftax benchmark. Built upon the `Craftax-Classic` environment in JAX, this extension supports efficient multi-agent training on accelerators [1]. Our experiments reveal that using a 4-agent LSTM model on an Nvidia T4 GPU can complete 100 million steps in approximately one hour. This environment will enable research into improving the social learning capabilities of AI agents in open-ended multi-agent settings, potentially enabling better generalization and faster learning through observing other agents.

1 Introduction

The real world is inherently multi-agent. For AI systems to be useful to people for a variety of applications, from autonomous cars to household robots, they will need to be able to effectively navigate human multi-agent environments. In recent years, the field of multi-agent reinforcement learning (MARL) has garnered significant attention in applications such as games, robotics, and autonomous driving [24]. In such situations, agents are not only required to perform individual tasks but must also collaborate and adapt to dynamic, often unpredictable conditions.

However, multi-agent environments do not merely present additional challenges such as coordinating with other agents; they also present new opportunities. Acquiring information from other intelligent agents in the environment through *social learning* can actually address fundamental challenges with modern AI algorithms, like sample complexity and generalization. It is well-known that acquiring skills by learning from expert agents can massively improve sample complexity vs. randomly exploring to attempt to independently discover a complex solution to a sparse reward problem [20]. Further, a recent line of work [21, 8, 2] has begun to demonstrate that social learning can actually address the fundamental problem faced by modern AI algorithms: generalization outside the training data. Consider for example an

[1]Source code available at `https://github.com/ericyuxuanye/Multi-Agent-Craftax`

Preprint. Under review.

autonomous car that encounters a novel situation on deployment that it never experienced during training: an underpass it is approaching is flooded. Rather than simply relying on the training strategy and driving directly into the water, it could instead take cues from what other cars are doing in order adapt safely to this unexpected scenario [17]. By *learning how to learn from other agents*, agents can use social learning to adapt to fundamentally new tasks at test time [21].

Social learning is especially effective in realistic multi-agent environments that contain many self-interested agents who are pursuing their own goals. This effect was convincingly demonstrated in a study of social learning in a multi-agent multi-armed bandit tournament [22]. In this tournament, agents could either independently explore to find the best strategy, or take an action to copy the strategy of another agent. The authors found that there was a linear relationship between the amount of copying performed by agents and their success in the tournament, with the most successful strategies performing copy on over 90% of turns [22]. The reason copying other agents strategies was so effective is that each agent was attempting to win the tournament, and thus maximize its own reward. Thus, at each timestep they were playing the most rewarding strategy they know how to play. Human environments have precisely this quality; most humans are not acting randomly or adversarially, but instead sensibly going about their business the best way they know how. AI agents deployed to these environments should be able to leverage this information to improve their performance on everyday tasks of interest.

Therefore, we propose to create a multi-agent environment with these properties, where multiple agents can independently pursue highly rewarding behavior. Our goal is for each agent to be able to pursue different final goals, but that the goals will share overlapping subtasks. This will enable testing whether agents can use skilled social learning to acquire related skills from other agents, even if the ultimate goal of those agents is not precisely the same. Further, the environment should test complex skills like exploration, long-term planning, and the ability to coordinate with others.

To this end, we present `Multi-agent Craftax`, the first multi-agent version of Craftax [19]. Our environment has the advantage of fast performance, enabling a wide variety of novel open-world experiments. Just like `Craftax-Classic`, we consider long-horizon, complex tasks requiring the completion of multiple subtasks. Agents must gather resources, such as wood, and craft tools, such as pickaxes, to unlock more advanced tasks like mining. These tasks necessitate an understanding of both the immediate environment and the long-term dependencies between actions. For example, to mine ore, an agent must first gather wood to craft a pickaxe, highlighting the importance of task-sequencing, resource management, and long-term planning. In addition, we can also test agents' abilities to work together. For example, one agent can create a crafting table that another agent uses later to craft tools.

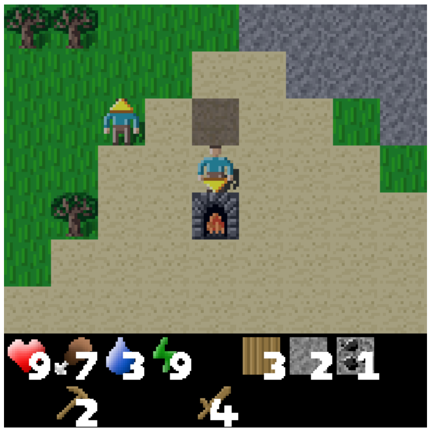

Figure 1: Sample pixel representation of a single agent during the game.

We present experiments benchmarking the state-of-the-art multi-agent learning algorithm, independent Proximal Policy Optimization (IPPO) [6, 23], in our environment. We compare the performance of multi-agent IPPO to single-agent PPO, and find that they both yield similar performance, with low sample efficiency and low probability of achieving the most complex tasks. This suggests that state-of-the-art multi-agent learning algorithms are incapable of using social learning to benefit from the presence of other intelligent learning agents in their environment. This highlights a crucial gap in current methods and presents an open research topic in the field of MARL.

The observed performance bottlenecks point to the need for more sophisticated strategies, such as social learning, where agents could learn from the discoveries and skills of other agents, or adapt their behavior based on shared knowledge and experience. Such approaches could lead to more effective collaboration, resource management, and task prioritization in complex, dynamic environments. Our environment, with its task dependencies and resource constraints, offers an ideal test bed for exploring social learning and other advanced coordination techniques in multi-agent systems. By investigating these methods, future research could develop more robust solutions for multi-agent cooperation, addressing the limitations seen with current RL approaches.

2 Background

2.1 Crafter

Crafter [11] is an open world survival game, partially based on Minecraft, that evaluates a wide range of general abilities within a single environment. Figure 1 shows as example of the type of visual input used in the game. The Crafter game is designed with the following criteria in mind:

- **Challenging.** Crafter tests capabilities lacking from state-of-the-art AI systems, including generalization (through procedurally generating novel environments), exploration (through a deep technology tree) and long-term reasoning and credit assignment (through repeated subtasks and sparse reward).
- **Meaningful and consistent evaluation.** Crafter measures the range of achievements that can be unlocked in each episode. The achievements are picked such that they represent meaningful milestones in behavior. For example, ‘Eat Cow’ represents a milestone in gathering food while ‘Make Wood Pickaxe’ represents a milestone being able to collect new resources.
- **Performance.** Crafter offers fast iteration speed due to evaluating multiple different agent abilities in a single environment, which vastly reduces the computational requirement of benchmarking multiple agent abilities compared to using a benchmark suite that runs a single agent across multiple different environments.

One limitation of Crafter is it being written in pure Python. While simpler to modify and extend, it introduces a bottleneck as the environment must be simulated on the CPU. As a result, the benchmark is limited to only 1 million environment interactions, which places a stronger emphasis on sample efficiency compared to other RL benchmarks.

2.2 RL Environments in JAX

JAX [3] is a Python library that facilitates high-performance, accelerator-optimized array computation and automatic differentiation. By leveraging Accelerated Linear Algebra (XLA), JAX allows Python code to be seamlessly compiled and executed on hardware accelerators such as GPUs and TPUs. This capability is particularly beneficial in reinforcement learning (RL) environments, where performance can be significantly improved by offloading computationally intensive tasks to specialized hardware.

A key advantage of using JAX in RL environments is its support for function transformations, such as `jit`, `vmap`, and `pmap`, which enable the compilation and parallelization of code across multiple devices. For instance, `vmap` allows the creation of vectorized environments by applying operations over batches of data, which is highly efficient for training multiple agents in parallel. This is especially relevant for multi-agent RL, where agents operate in parallel and interact with complex, dynamic environments. Furthermore, `scan` is often employed to perform entire rollouts of an environment in a highly optimized, loop-unrolling fashion, minimizing overhead and maximizing throughput.

Another crucial benefit of JAX is the ability to write both the environment logic and the neural network model in the same framework, which avoids the performance penalties associated with data transfer between the environment and the model. Libraries built on JAX, such as Flax [12] or Equinox [14], offer additional flexibility for building and training neural networks without incurring unnecessary computational overhead.

By utilizing JAX in multi-agent RL environments, we can efficiently scale simulations, optimize resource use, and significantly accelerate both training and inference, providing a powerful platform for research in open-world, task-dependent environments.

2.3 Craftax

Craftax [19] is a recently developed RL benchmark that is designed to strike a balance between computational efficiency and task complexity. For instance, environments like Crafter [11], NetHack [15], and Minecraft [10] are highly complex but slow, requiring substantial computational resources, while others like Minigrid [4] and Procgen [5] are fast but lacks the mechanics that make the previous environments so interesting.

`Craftax-Classic` [19], a reimplementation of Crafter using JAX, offers significant computational speed improvements, running up to 250 times faster than the Python-native Crafter environment. This efficiency allows RL algorithms like Proximal Policy Optimization (PPO) to perform one billion environment interactions in under an hour on a single GPU while achieving near-optimal performance. The benchmark is designed to simulate open-ended tasks, demanding deep exploration, long-term planning, memory, and adaptation to novel scenarios.

This makes Craftax an ideal test bed for exploring advanced RL algorithms, particularly in resource-constrained settings, without sacrificing task complexity. Despite its advancements, current RL methods like global and episodic exploration struggle to make significant progress on this benchmark, demonstrating the need for novel approaches.

In this project, we extended the `Craftax-Classic` environment as it had less complexity, making it easier to modify.

2.4 Partially Observable Markov Games

The environment is modeled as a partially observable markov game $M = (S, A, T, R, \Omega, O, \gamma)$ such that S, A, and Ω define the environment's joint state, action, and observation space respectively [9]. In each environment interaction, each agent i selects action $a_i \in A_i$, which yields the joint action $(a_1, \dots, a_n)$ of all agents. The environment subsequently transitions to a new state according to the transition function $T : S \times A \to S$, and produces the corresponding observations consistent with $O : S \times A \to \Omega$. Finally, the reward is calculated according to the reward function $R : S \times A \to \mathbb{R}^n$. Each agent's policy is represented as $\pi_i : \Omega_i \to A_i$, and the goal of each agent's policy is to maximize its own expected future discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{ti}\right]$ where $\gamma \in [0, 1]$ is the discount factor and r_{ti} is the reward at time t for agent i. The set of all agents' policies $\pi = (\pi_1, \dots, \pi_n)$ is referred to as the joint policy [9].

3 Related Works

There have been many attempts at creating minecraft-like or minecraft-based environments such as Minecraft Building Assistance Game [16] and VillagerBench [7]. However, we believe that this is the first multi-agent minecraft-like environment with the logic entirely written in JAX.

4 Environment Mechanics

This section describes the game mechanics of the environment, which is largely derived from `Craftax-Classic`.

4.1 Game Objective

The core game is equivalent to `Craftax-Classic`, which in turn has almost exactly the same characteristics as Crafter, except for being rewritten and some performance optimizations that should not affect normal gameplay. Agents need to complete as many achievements

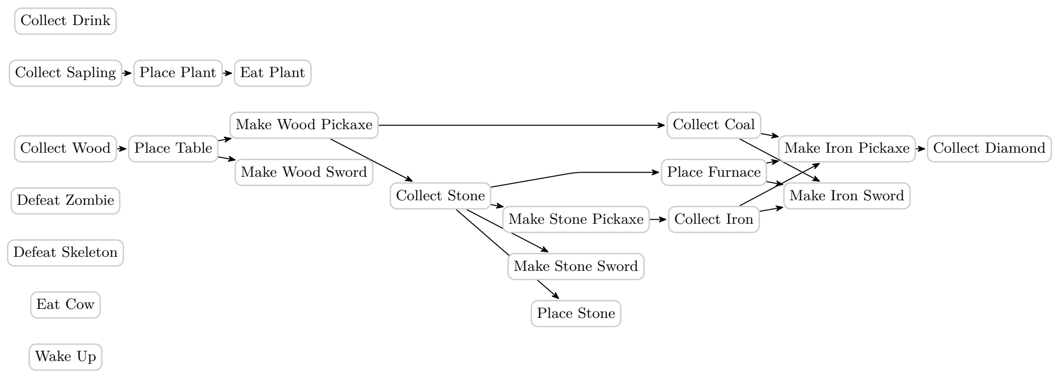

Figure 2: Tech Tree of the 22 achievements that can be achieved each episode. The arrows indicate the order that achievements need to be completed. In order to complete a task, all previous tasks with arrows to the current task must be completed. Note that some tasks need to repeated multiple times, such as requiring two wood in order to place a table. In the multi-agent environment, all previous tasks do not necessarily need to be finished by the same agent in order for an agent to finish the current task. For example, an agent can make a wood pickaxe without achieving "Place Table" if another agent has already placed a table.

as possible within a single episode. However, in order to have time to complete their achievements, players will also need to try to survive by keeping their health above zero. This is achieved by keeping their intrinsics (food, water, energy) positive and avoiding attacks from zombies and skeletons.

The 22 Achievements are structured in a way that balances depth and breadth. Some achievements, such as creating iron tools, require many subtasks such as building a furnace, collecting coal, and collecting iron, but many other achievements, such as `COLLECT DRINK` and `EAT COW`, can be completed without completing any other tasks. Consult Figure 2 for more details.

Additionally, achievements are structured in a way such that they represent significant milestones in the abilities of the agents. Therefore, achievements are an appropriate way to objectively measure agent performance. An agent that can on average obtain more achievements per episode than another agent is objectively better in its exploration and long-term planning abilities.

4.2 Observation

Just like Craftax, the game has both Symbolic and Pixel environments. The Pixel environment directly shows the game in terms of RGB pixels as a human would if they play it, while the Symbolic environment expresses the game state in a more compact way that is easier to process for machines. The simulations ran in this environment all used the Symbolic environment as it requires a smaller neural network, less training, and faster iteration, and we are interested in studying the social learning capabilities of RL agents rather than the computer vision task of extracting information from pixels.

The observation is returned in an array with shape $(n, 1346)$, where n is the number of players in the environment. Each agent's observation contains the locations of blocks, mobs, and other players within 4 blocks horizontally and 43 blocks vertically of the player. If needed, the observations can be modified relatively easily by modifying the rendering function. For more information, refer to appendix A.

Returning an array of observations also makes it possible to implement other multi-agent algorithms which allow agents to share observations as a way of enabling social learning through post-processing of the observations.

4.3 Actions

This environment implements a discrete action space, which can be used with many modern deep RL algorithms. At each time step, each player picks one of 17 actions, which includes movement actions, the `DO` action for interacting with objects, crafting actions, or `NOOP`. More specific information is in appendix B.

4.4 Rewards

The reward calculation is largely unchanged from `Craftax-Classic` [19], except that the rewards are calculated for each agent individually and returned as an array with the same length as the number of agents. Each agent would get a reward of +1 if an achievement was achieved during an environment step. Additionally, to encourage agents to preserve their health, a reward of 0.1 times the change in the player's health is added to the reward.

4.5 Player Constraints

The only constraints placed on the multi-agent environment is that only one player is allowed to perform a `DO` or block placement action on the same block. If, for example, two players try to perform `DO` on the same tree at the same time step, one player will be chosen at random to complete the action, while the action of the other players will be changed to a `NOOP`. This prevents players from exploiting the game through acquiring duplicate resources from the same block, and more closely model real-world dynamics.

For computing which actions conflict and the number of conflicts for each action in JAX, a naïve $\mathcal{O}(n^2)$ search was implemented that iterates through every agent pair, which runs fast as long as the number of agents in the environment is reasonable.

4.6 Mob Logic

`Craftax-Classic` [19] optimizes the performance by not computing updates for mobs that are unseen by players. The main mobs in `Craftax-Classic` (zombies, skeletons, and cows) are spawned near the agent and despawned when the agent moves too far away. However, this logic needs to be modified in the case of multiple agents. In the multi-agent case, the nearest player would be considered. This means that any tile that is within `mob_despawn_distance` from the closest player can have a mob spawned (if all mob spawning conditions are met), and any mob that is farther than `mob_despawn_distance` from the closest player will be despawned. The tiles are calculated by computing a $(n, 64, 64)$ array of distances, and folding it along the first axis by taking the minimum.

A similar calculation was made for computing the direction that the mobs would take. In Craftax, the direction that the mobs would move in depended on the mob type and distance from the player. For example, a skeleton would move towards the player if it is too far from the player, move away from the player if it is too close, or move in a random direction otherwise. In the multi-agent case, the movement of the skeleton is simply based on the nearest player instead.

4.7 Handling of Dead Players

During the course of a single episode, some agents will inevitably die before other agents. This means that special logic was required to handle 'dead' agents. In this implementation, the environment will continue to run until all agents have died, either through depletion of player health or running into lava. During the course of each episode, all agents will receive their respective observations whether dead or alive, but each player will have a 'dead' bit set to 0 or 1 in their observation arrays. The environment would modify the actions of dead players into `NOOP`s. Additionally, dead players do not gain or lose health, and are no longer visible to other players. Therefore, models should be able to learn that no further achievements are possible once dead.

One potential issue with this current implementation is that agents that perform better will likely last longer, causing it to have more opportunities to train compared to agents that die

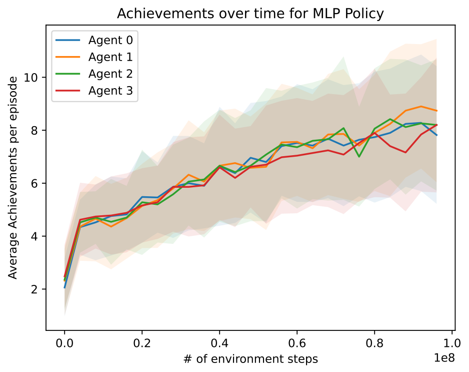

Figure 3: PPO Achievements during the course of training for 100 million environment interactions with MLP policy. The error area represents standard deviation.

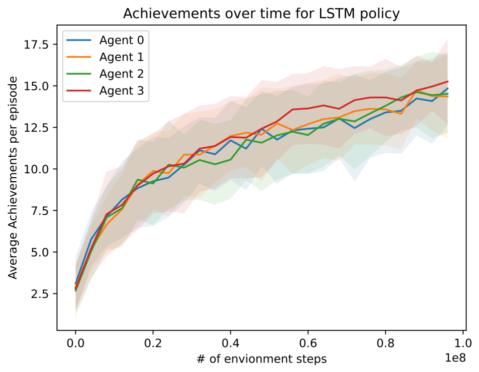

Figure 4: PPO Achievements during the course of training for 100 million environment interactions with LSTM policy. The error area represents standard deviation.

earlier. This could allow those agents to perform even better relative to other agents, causing the training curve to diverge. However, this behavior was not observed when training for only 100 million environment interactions.

5 Experimental Setup

All experiments were run with 4 agents with 100 million steps on a Google Colab instance with a single Nvidia T4 GPU, which is accessible for free by any researcher. Being written in JAX, training with 4 agents took approximately 1 hour, making it relatively easy to anyone to replicate. We used Independent PPO [6, 23], a state-of-the-art MARL algorithm which trains all agents independently with PPO with no additional information sharing mechanisms. Instead of only evaluating one agent each step, we evaluated all of the agents, and independently applied PPO updates after each environment batch. The PPO algorithm is based on CleanRL [13], *Recurrent PPO in JAX* [1], and PureJaxRL [18]. The two variants tested were a version with only linear activation layers and an LSTM version.

Later, we ran another LSTM model for the same number of steps but with only a single agent. We compared the single agent performance with the average of the multi-agent version.

All experiments were conducted using the Symbolic environment instead of the Pixel environment to focus more efficiently on testing social learning capabilities rather than improving computer visual skills.

6 Experimental Results

Figures 3 and 4 were generated by using the mean and standard deviation of the achievements in one episode over 50 random starting environments as all agents had a similar number of achievements per episode throughout training.

Using snapshots of the models during training, we ran each snapshot over a random distribution of environments and averaged the achievements. Comparing Figures 3 and 4, the LSTM performed better than the model with only linear and activation layers due to the environment being only partially observed. While replaying the episodes, we noticed that the non-LSTM version would occasionally get stuck if the agent's position was mostly surrounded by objects, while the LSTM version would try a variety of unique techniques in order to get unstuck.

Comparing the single and multi-agent performance in Figure 5, we notice that the version with a single agent is performed slightly better on average than the scenario with four agents. This indicates that agents are not benefitting from being in an environment with other

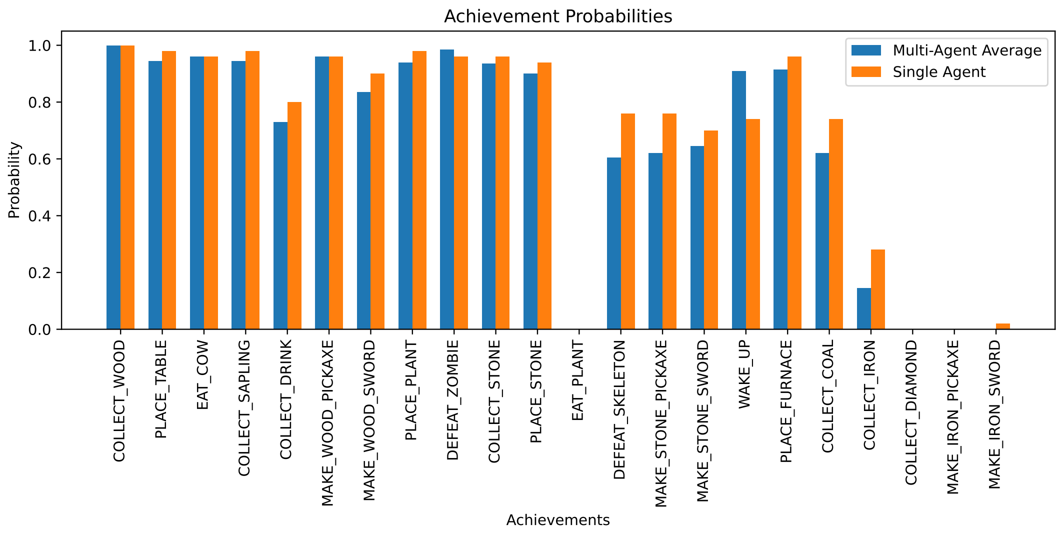

Figure 5: LSTM achievement probabilities after training with four agents versus a single agent, using a sample of 50 episodes. The multi-agent average takes the average of all four agents.

learning agents, even though those other agents are simultaneously discovering behaviors and skills that could be useful to learn from. Therefore, our results indicate that state-of-the-art MARL algorithms are not able to effectively leverage social learning from other self-interested agents. Therefore we propose that there is significant work to be done to improve these abilities, and argue that Multi-agent Craftax is a useful tool in facilitating this research.

7 Discussion

7.1 Future Work

While this environment is challenging for agents to perform well in a small number of environment interactions, the authors of Craftax have shown that after training for a billion environment interactions on the `Craftax-Classic` environment, they were able to have a model that solved around 90% of all possible achievements [19]. This represents extremely poor sample efficiency. We believe that if agents could leverage social learning to aquire skills from other agents and build on them, this sample complexity could be significantly reduced.

7.2 Conclusion

We present the first multi-agent adaptation of Craftax that supports multiple agents in an open world environment. This environment enables testing and improving the social intelligence of AI agents in a complex environment that presents significant exploration, long-term planning, and reasoning challenges. Preliminary testing conducted with state-of-the-art MARL algorithms shows no significant improvement over the single agent case, indicating that agents are not currently able to benefit from social learning to improve performance, in spite of having access to more information about how to successfully obtain achievements. We hope that this benchmark will spur research into the development of improved social learning abilities for AI agents.

References

[1] Recurrent ppo in jax. `https://github.com/subho406/Recurrent-PPO-Jax`, 2023.

[2] A. Bhoopchand, B. Brownfield, A. Collister, A. Dal Lago, A. Edwards, R. Everett, A. Fréchette, Y. G. Oliveira, E. Hughes, K. W. Mathewson, et al. Learning few-shot imitation as cultural transmission. *Nature Communications*, 14(1):7536, 2023.

[3] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[4] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid miniworld: Modular customizable reinforcement learning environments for goal-oriented tasks, 2023.

[5] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020.

[6] C. S. De Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. Torr, M. Sun, and S. Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[7] Y. Dong, X. Zhu, Z. Pan, L. Zhu, and Y. Yang. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft, 2024.

[8] A. Filos, C. Lyle, Y. Gal, S. Levine, N. Jaques, and G. Farquhar. Psiphi-learning: Reinforcement learning with demonstrations using successor features and inverse temporal difference learning. In *International Conference on Machine Learning*, pages 3305–3317. PMLR, 2021.

[9] N. Grupen, N. Jaques, B. Kim, and S. Omidshafiei. Concept-based understanding of emergent multi-agent behavior. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[10] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations, 2019.

[11] D. Hafner. Benchmarking the spectrum of agent capabilities, 2021.

[12] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee. Flax: A neural network library and ecosystem for JAX, 2024.

[13] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[14] P. Kidger and C. Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[15] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment, 2020.

[16] C. Laidlaw, E. Bronstein, T. Guo, D. Feng, L. Berglund, J. Svegliato, S. Russell, and A. Dragan. Scalably solving assistance games. In *ICML 2024 Workshop on Models of Human Feedback for AI Alignment*, 2024.

[17] N. C. Landolfi and A. D. Dragan. Social cohesion in autonomous driving. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8118–8125. IEEE, 2018.

[18] C. Lu, J. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, and J. Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.

[19] M. Matthews, M. Beukman, B. Ellis, M. Samvelyan, M. Jackson, S. Coward, and J. Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning, 2024.

[20] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[21] K. K. Ndousse, D. Eck, S. Levine, and N. Jaques. Emergent social learning via multi-agent reinforcement learning. In *International conference on machine learning*, pages 7991–8004. PMLR, 2021.

[22] L. Rendell, R. Boyd, D. Cownden, M. Enquist, K. Eriksson, M. W. Feldman, L. Fogarty, S. Ghirlanda, T. Lillicrap, and K. N. Laland. Why copy others? insights from the social learning strategies tournament. *Science*, 328(5975):208–213, 2010.

[23] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

[24] K. Zhang, Z. Yang, and T. Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2021.

A Agent Observations

The following are the observations in the Symbolic Environment.

- 21 7×9 grids, one-hot encoded, which show surrounding objects or mobs around the player, who is always in the center of the grid. Each grid indicates different objects or mobs. A 1 in a position means that there is a specific mob or object, and 0 means there is no mob or object of a specific type at that location. Observed mobs include zombies, skeletons, cows, arrows, and other players.
- An entry for each inventory of the player, 0.0 meaning that the player does not have a specific item in the inventory, and 0.9 meaning that the player has 9 items of that type in their inventory
- An entry for each of the player intrinsic (health, food, drink, energy). 0.0 means that the player does not have any of that intrinsic, and 0.9 means that the player has the max of that intrinsic.
- One hot encoding for each of the 4 possible player directions
- Light level of the environment
- Whether the player is sleeping
- Whether the player is alive

B Agent Actions

At each time step, each player can take the following actions:

- `NOOP`: Don't perform any action
- `LEFT`, `RIGHT`, `UP`, `DOWN`: Move actions. This is equivalent to a `NOOP` if there is a mob, tree, stone, plant, or water blocking the way.
- `DO`: The `DO` action performs a different action depending on the block that the player is facing.
  - Grass: The player will try to collect saplings.
  - Stone, coal, iron, or diamond block: the player will try to mine that block.
  - Tree: The player will mine for wood.
  - Water: The player will drink and replenish the drink intrinsic.
  - Cow: The player will eat the cow and replenish the hunger intrinsic.
  - Plant: If ripe, the player will eat fruits on the plant and replenish the hunger intrinsic.

    - Zombie or Skeleton: The player will try to attack.
    - Other blocks will be equivalent to a `NOOP`.
- `SLEEP`: If the player's energy level is less than 9, begin sleeping.
- `PLACE STONE`, `PLACE TABLE`, `PLACE FURNACE`, `PLACE PLANT`: Placement actions that will place the specified item if the player has the required inventory.
- `MAKE WOOD PICKAXE`, `MAKE STONE PICKAXE`, `MAKE IRON PICKAXE`, `MAKE WOOD SWORD`, `MAKE STONE SWORD`, `MAKE IRON SWORD`: Player will create the specified items given that the conditions are met.

C Training Parameters

The MLP policy had two hidden layers with size 64 for both the actor and critic networks. The LSTM policy had a shared network with hidden layers of size 64 and 32, connected to an LSTM block with 32 features. This is connected to separate actor and critic networks each with a hidden layer of size 64.

For both instances, we used the same parameters as the CleanRL [13] defaults, but with 400 parallel environments, 8 minibatches, 500 steps per batch, and no learning rate annealing.