BabyDungeon: Multi-Agent Collaboration to Solve Escape-Room Style Tasks

Mollie Bakal **Barrett Lattimer** **Adam Li**

Abstract

A long-term goal of AI is the ability for agents to collaborate amongst each other to achieve a common goal. Such an ability allows for emergent behavior that can complete tasks far more advanced than one agent could accomplish alone. Our project, BabyDungeon, aims to use reinforcement learning to train multiple agents to collaborate through communication to solve a task, i.e. an escape room. Through environments and reward functions, we aim to create agents that use communication to collaborate and escape from the escape-room style dungeon game.

1 Introduction

One of the great advantages of humans is our ability to consider a social environment and "put ourselves into the shoes of others" by analyzing their actions and extrapolating and simulating their thought processes. Throughout history, humans have had to collaborate in a variety of ways to complete tasks too complicated for any single person to complete on their own. There are many fields where interactions between multiple agents enable tasks more complicated than can be performed by an single-agent system, from physical labor to scientific research. If artificial agents are to work in these areas, it may be beneficial to allow them to collaborate with each other. Communication is a must for collaboration; as the mythical Tower of Babel indicates, it is simply not possible for agents to collaborate effectively if they can't understand or model each other. Tasks where multiple agents have limited viewpoints and abilities, such as in co-operative or team-based games, greatly benefit from effective communication and collaboration between agents.

One such team-based game is the escape room, which is a task where multiple individuals have to complete a variety of puzzles, coordinate, and communicate ideas to finish a larger task [1]. This is typically seen by humans as a "team-building exercise"; working together on these puzzles enables people to see how others communicate and understand the thinking of the others. The escape room can be modeled as a long-horizon collaboration task involving the coordination of actions and communication of information which only a subset of agents are privy to. The agents then have to work together, coordinating and communicating their actions, to complete a variety of tasks until some larger task (in this case "escape the environment") is achieved. Oftentimes, these escape rooms are structured as a series of stages such that each stage consists of a cluster of tasks which can be completed in any order; however, each must have their results coordinated to successfully gain access to the next stage or exit. At other times, the tasks within a stage will be interdependent; finishing one task within a larger stage is necessary to complete another, even though tasks available before the first's completion may be completed after the second.

Our team will examine artificial model collaboration by training agents in such a virtual escape room to work together. These agents can and will be incentivised to communicate to complete their tasks; they will be unable to complete their tasks without information from the other agent's point of view. In our paper we make three contributions. First, we introduce an optimizable formalization for an escape-room style environment and describe its utility for evaluating Multiagent systems with communication. Second, we implement an environment based on this formalization and characterize our tasks along with their relative difficulties. This implemented environment has the capability

to be extended to create optimizable graphs which specify the upper bound solutions to the room and ensure that the environment can be solved with a properly communicating multiagent system Third, we explore the use of variational inference methods for learning about our world in such a multiagent environment and demonstrate the limitations of using such methods to drive the policies of independent agents in such an environment.

2 Background

2.1 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning is a long-studied field. Reinforcement learning is a form of machine learning characterized by allowing an agent to explore and discover which actions gain a reward, training the agent in how to act. When placing multiple agents with agency into a single environment, as these models learn their own behavior, they learn from observation the behavior of other actors in the environment. There are issues, though; while learning the behavior of other agents, a given agent also learns and adjusts its behavior. The behavior learned by the agents with regards to this one no longer applies.

There has also been work done directly with multi-agent reinforcement learning (MARL). The minigrid [2] framework was extended into MarlGrid [3], a small environment designed specifically for multi-agent reinforcement learning. Both environments come pre-loaded with a small number of tasks and minimalistic starting code.

MarlGrid was built upon in the paper "Learning to Ground Multi-Agent Communication with Autoencoders" [4]. This paper uses agents with no pre-existing language and trains them with a self-supervised task so that all agents learn the same grounding, allowing them to communicate. Communication is achieved by splitting each agent into a speaker and listener module and communicating encoded observations between agents at every timestep. Two new environments were also added in this paper, both of which were non-referential but ordinal MARL games: RedBlueDoor and FindGoal. Our paper focuses on implementing novel referential environments in MarlGrid to force communications between agents. Using referential environments forces agents to communicate in order to achieve success with the intention of encouraging multi agent communication. Additionally, our paper introduces a novel model that models an agent's internal world model. Our new agent is trained to predict its next observation using its internal world model and to then communicate that internal world model with the other agent rather than its observation projected into latent space.

2.2 Multi-Agent Communication

When examining the problem of multi-agent collaboration, communication is a common concern. When agents have different bases of knowledge, the only way that they can actively collaborate is to share knowledge about their current state. In essence, communication only works if both speaker and listener have a common understanding of what is meant by the utterance. Different projects seek to mitigate this issue in different ways. The MarlGrid paper mentioned above [4] models language using a self-supervised reinforcement task to create a common grounding to the world environment shared amongst all agents. This ensures that all can communicate on the same footing with the same knowledge base.

The means of communication can also vary. [5] is a MARL problem which side-steps the problem of communication by sharing perception and state between all agents in a constant off-plan adjustment. This has limitations, though; it is best for a problem where agents are learning collaboratively instead of competitively, and it was not tested on agents with dissimilar abilities. In prior works such as [6, 7], the authors modeled communication between agents with shared communication vectors. For [6], a single shared vector was created and all agents given "equal voice" by taking the mean sum of the vector. In a similar case, the authors of [7] concatenated fixed length messages and demonstrated the emergence of different communication patterns for similar obfuscations of knowledge. Other works attempt to train agents to communicate using natural language, as humans do. In [7], where one agent is strictly an information provider and one is strictly a seeker, both units have a dialog policy and demonstrate training of communication policies over a unique natural language medium. But this is highly constrained. Given [8], where the authors find that natural language does not emerge within

multi-agent dialog, these approaches used to keep language natural by utilizing and constraining communication to existing architectures are very reasonable.

3 Methodology

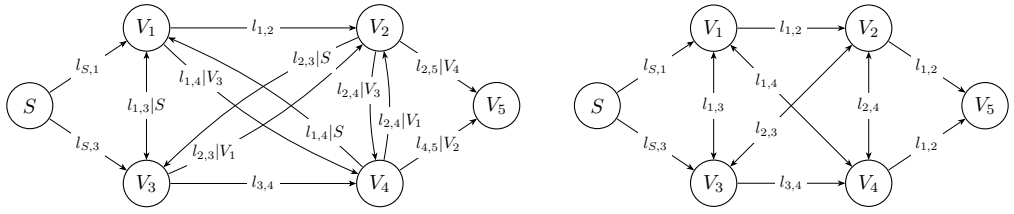
3.1 Formalization of an Escape-Room Style Game

In [9], the author defines escape rooms as live-action team-based games where players have to discover clues, solve puzzles, and accomplish tasks in one or more rooms in order to accomplish some specific goal (usually to escape the room) driven by the story behind the game design. This overall goal must be accomplished within a limited amount of time, and they explain that in addition to teamwork, communication, and delegation, escape rooms also require critical and lateral thinking as well as attention to detail. From this description, along with the characterization of escape room designs from [10], also by the same author, we attempt to further formalize what an escape-room style game is.

For every agent in an escape room-style game, the policy to play this game will be based on a Partially-Observable Markov Decision Process (POMDP), where the partial observations are locally limited at the puzzle level [11]. That is, each agent may not observe the overall system the same way. However, to get a more usable representation, we instead consider all agents as joint policy heads of an larger, abstract agent. We can treat the policy of all agents as a POMDP with partial observability of the overall game instead of each individual puzzle. In this way, we can refactor this game as a coordination-collaboration game and simplify it's analysis by only considering a bounded near-optimal case.

This refactored game is then modeled as a labeled directed graph G of puzzles V, which can also be described as action graph games, and are considered completed upon first visit. The edges of this graph C denotes the longest individual agent path length l from one game to the next completable game, conditioned on the pre-requisites of that next game. The prerequisites for each game V are the visitation requirements, or completion requirements of past games, as each game node is considered completed after first visitation. When this condition is not met, the length of the conditioned edge is infinity, effectively "disconnecting" that edge. Since games are considered completed on first visit, we introduce an additional starting node S to go before the first game or games. Then we can formulate a graph problem which when the minimizing path is found, a near-optimal solution is found. This full graph model can be found in Figure 1a, and can be viewed as a further generalization and formalization of the graphs used in [10] to characterize the organization of an escape room.

Upon further inspection, since game requirements are based on completion requirements, they're directly related to the visitation of previous nodes. We further observe that as a result of this, these conditional requirements are directly based on the edges directed into any target game node. That is, the conditional requirements for all edges into a game are a subset of the games that are connected to that target game, specifically those attached to single direction directed edges. If we add a new



(a) Full graph model for minimization.

(b) Relaxed graph model.

Figure 1: How we model an escape-room style game. S denotes the starting node, while V_5 denotes the final puzzle. By first refactoring the overall game as a series of puzzles with conditioned directed edge requirements, we can create the graph in 1a. To obtain near-optimal game path can be directly minimized from this graph. However, we observe that any conditional requirements for all games are solely from the other edges directed into the target which allows us to revise the optimization problem and relax the edge conditions to obtain the directed graph in 1b. To obtain the same near-optimal solution, we instead minimize the Hamiltonian or Rudrata path from S to V_5.

constraint that each node can only be visited once, we can then relax the requirement of edge conditions as the significant requirements are captured by the directed edges. As a result of this, we can reasonable revise the optimization problem to instead search for the minimizing Hamiltonian or Rudrata path within some tolerance [12], relaxing the requirement of edge conditions for a simpler graph which can be found in Figure 1b. In this problem, we find the minimal path which also visits each node exactly once from our starting node to our final game node [13]. To finish the model, each game node V also has a cost associated with it which we simply estimate, but would be the length of the Nash-equilibrium from the action graph which models the associated game [11]. In this way, we can find a near-optimal solution which can act as an upper bound to game completion-time.

From this, we can make several considerations and better motivate the choice of an escape room. Due to the conditional reliance on past nodes visited, some form of memory will likely be required in each agent dependent on the game. Due to the multiple possible path solutions at higher difficulties, agents will need to coordinate over a larger graph and therefore should be able to reason about their world. Communication of observations is still primarily forced by the design of our games and the limited views of each independent agent. By using the larger escape-room graph to demonstrate coordination, and by choosing games that require cooperation, we can demonstrate that this environment encourages long-horizon collaborative planning, with both cooperation and coordination components.

3.2 Agents Design and Baseline

To maintain a deeper understanding of the world, we introduce a novel MARL agent which uses an internal world model to communicate with other agents and motivate future actions. We chose to explore the usage of a world model from our considerations and motivations we found through formalizing an escape-room style game in 3.1. The new MARL agent was inspired by other work done on reinforcement learning world models [14], and we extend this framework to incorporate multiple agents and communication. The new MARL agent can be viewed in three separate components: the vision, world model, and controller. A diagram showing an overview of these components can be found in 2.

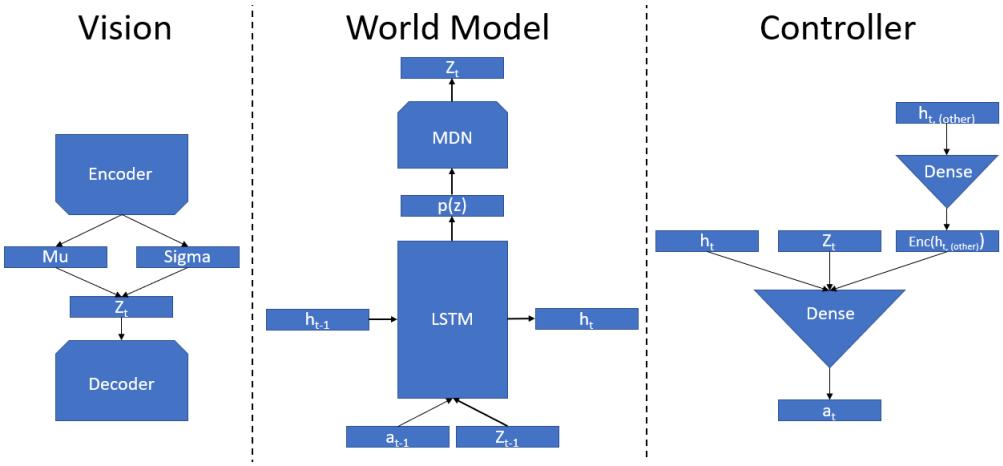


Figure 2: The network models we used for our agent. It consists of all three components, vision, world model, and controller, connected by their respective intermediary vectors.

The vision component consists of a Convolution Variational Autoencoder (VAE) that projects the current agent observation into the mean and variance of a Gaussian, which is then used to sample the resulting latent space observation vector z_t. The encoder and decoder components each consist of 4 convolutional layers. A custom loss function was written to train the VAE on both reconstruction loss and Kullback–Leibler divergence (KL-divergence) loss based on [15]. Reconstruction loss for us was based on the mean squared error between the predicted and original image, and it encourages effective reproduction of the input image. KL-divergence loss specifically attributes loss to the Gaussian distribution used to reproduce the latent vector and is used to reduce the probability distribution used to model the Gaussian that samples the latent observation vector. It describes the KL-divergence, which is the expected difference in the log of two probability distributions under the measure of the

prior, between a diagonal multivariate normal for the latent space and the standard normal distribution we use as a prior [15]. Similar to [14], we choose to use a VAE due to the underlying representation for its resulting encoding. In this way, our next component, the world model, has a stronger probabilistic basis for encoding an imagination of the world.

The world model is a Mixture-Density Recurrent Network (MDN-RNN) [16]. This consists of an LSTM with a Gaussian Mixture Model head that is designed specifically for generation. The world model in each agent uses it's previous time step's latent observation vector z_{t-1}, previous time step's action a_{t-1}, and previous hidden state h_{t-1} to output the next hidden state h_t and a mixture of Gaussian probability distributions $p(z)$. The mixture of Gaussian probability distribution output $p(z)$ is then used by a Mixture-Density Network (MDN) to predict the next latent vector z_t. Again, the use of Gaussian distributions helps mediate noise and provide more stable and effective generation and provides a stronger probabilistic basis.

Finally the controller contains a single linear projection to a probability distribution of the next action and handles communication of world states between agents. First, the current hidden layer output from the other agent $h_{t,(other)}$ is linearly projected to a smaller dimension to form the communication vector. In this way, other agents communicate their own world model to the other agent. The current hidden state for a given agent h_t, the projected current hidden state from the other agent $h_{t,(other)}$, and the current agent's latent observation vector z_t were concatenated and used as input for the actor and critic networks for each agent.

The baseline model we compare our agent to is from the "Learning to Ground Multi-Agent Communication with Autoencoders" [4] paper. Each agent is split into a listener and speaker module in order to provide a framework for communication. Using an image encoder and proceeding autoencoder, the input observation for each agent is projected into the communication latent space resulting in a communication vector. Communication vectors are then sent to other agent's listener modules which linearly projects all the incoming communication messages and concatenates the resulting vector with its own encoded observation. The resulting concatenation vector is then fed through a policy network containing an LSTM to predict the next action.[1]

3.3 Environment Design

When we were developing our escape rooms, we decided to prioritize tasks which could easily be implemented into the MarlGrid and MiniGrid frameworks. We selected tasks which were both referential [17] and ordinal. This would require communication referring to the agents' understanding of the world while emphasizing the timing of actions was important. To test the ability of the agents to complete tasks like this, we built a number of single-task puzzle rooms. Each individual room is modeled by a 10 by 10 grid with walls making up the entire perimeter, except for one square where there is a door. For each individual room the agents must learn a task and how to complete it given the objects in the room.

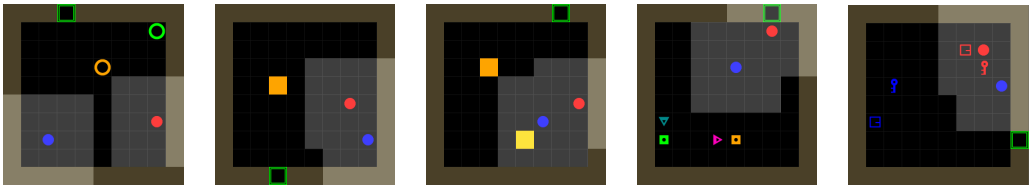
Rooms can then easily be linked together into a larger environment using the custom escape room generator we've created. This allows us to implement escape-room style games in a similar graphical manner to our prior formalization. In this way, the coordination requirements can also be incorporated, making our environment effectively equivalent in analysis to an escape-room style game, where our agents will be encouraged to coordinate their solutions rather than solve them independently.

We briefly introduce and describe our designed puzzle games here. In the appendix, we provide additional images to help better understand the environments we have made.

3.3.1 ColorOrderedLevers

This environment is a modification to the RedBlueDoors task introduced in [4]. In their game, they randomize the placement of red and blue doors on opposite sides of the room, where the goal of the game is to first open the red door before opening the blue door. In our modification, we create two "lever" objects, though they are shown as circles to make them distinct, and they must be toggled in an consistent order based on their color. This task is primarily ordinal as the levers must be flipped

[1]The authors in their paper describe using a GRU; however, when we inspected the code, we could only find their implementation using an LSTM present.



(a) (b) (c) (d) (e)

Figure 3: The puzzle rooms we introduce in this paper. In order from (a) to (e), they are: ColorOrderedLevers, SinglePressurePlate, TwoPressurePlate, ColorCycleBox, ColorBlindKeys.

in a particular order and there is no assigning of roles to it. Figure 3a shows an initial state for this puzzle room.

3.3.2 SinglePressurePlate

This is the first environment we created. In it, a single pressure plate is generated in one half of the room, and on the opposing wall a door is generated. One agent must stand on the pressure plate in order to open the door and to keep the door open the other agent must walk through the door. Figure 3b shows an example of this room. Rewards are given out to agents for initially stepping on the pressure plate and again for walking through the door. This task is referential because the pressure plate is almost always generated out of view of the door, meaning the agents must synchronize their actions and communicate to get one through the door. This task is also ordinal because the pressure plate must be stepped on first to allow the door to be opened.

3.3.3 TwoPressurePlate

The second environment we created requires both agents to step on two separate pressure plates at the same time in order to open a door. Figure 3c shows an initial state for this game. Two pressure plates are randomly generated in the room as well as a door along the perimeter. When both agents are standing on different pressure plates the door permanently opens and the agents can walk through. Agents get rewarded for initially stepping on a pressure plate, when the door is opened, and for walking through the door. Commonly the pressure plates are out of view of one another, necessitating efficient communication and making this task referential. This room is also ordinal because both pressure plates must be simultaneously activated before the door will open.

3.3.4 CycleBoxColor

The third novel environment, shown in figure 3d, deals with agent perception. It has two agents in distinct roles: an observer and a actor. The observer can see both triangles (keys) but not values (squares) and cannot act; the actor can toggle the colors of the squares, but cannot see the triangles. Both can move freely. The goal is for the colors of the squares to match the colors of the triangles. This task is referential, as the observer must communicate about the world state to the actor. It is also ordinal, as all the squares must match all the triangles at a given time for a reward to be given.

3.3.5 ColorBlindKeys

Our final novel environment also deals with agent perceptions. It is similar to Red/Blue door, in that it has two agents and two "doors". However, in this one, agents are selectively blinded; they cannot see keys which are the same color which they are, and cannot pick up keys of different colors. The "doors" in this case are keyholes, and they are color-coded and locked. Each can be unlocked with a key, which then unlocks the overall room door. To complete the task, the agents must share information about the location of the keys with each other; this is clearly referential. It is less ordinal than the others; it is only ordinal in that the keys must be picked up and carried before the door can be unlocked. An example for an initial room state can be found in figure 3e.

3.4 Implementation Details

To create the environments we trained in, we extended the MarlGrid environment created in [4]. Similar to [18], we introduce the generators into this environment with preset layouts that puzzle rooms would be generated into. With this, we create two combined environment, OnePuzzleRoom and TwoPuzzleRoom, which will randomly generate (combinations) of puzzle(s) and chain them together in the latter case. Three examples of our TwoPuzzleRoom environment can be found in 4.

We utilize the training method from the prior MarlGrid implemented in [4] environment, and introduce our agent model as an additional implementation to evaluate. To train our models we utilized a variety of modern multicore computer systems with a minimum of a Nvidia 1070 GPU and 16GB of RAM, and each training run of 500k epochs took approximately 10 to 20 hours to complete. At the high end, we utilized a server system with 48 logical cores, 512GB RAM, and 4 Nvidia A100 40GB GPUs. Training within the Nvidia Container Runtime[2] on this system took anywhere from 5 to 20 hours to complete depending on scheduling availability. Due to computational and time restrictions, we do not continue training beyond 500k epochs.

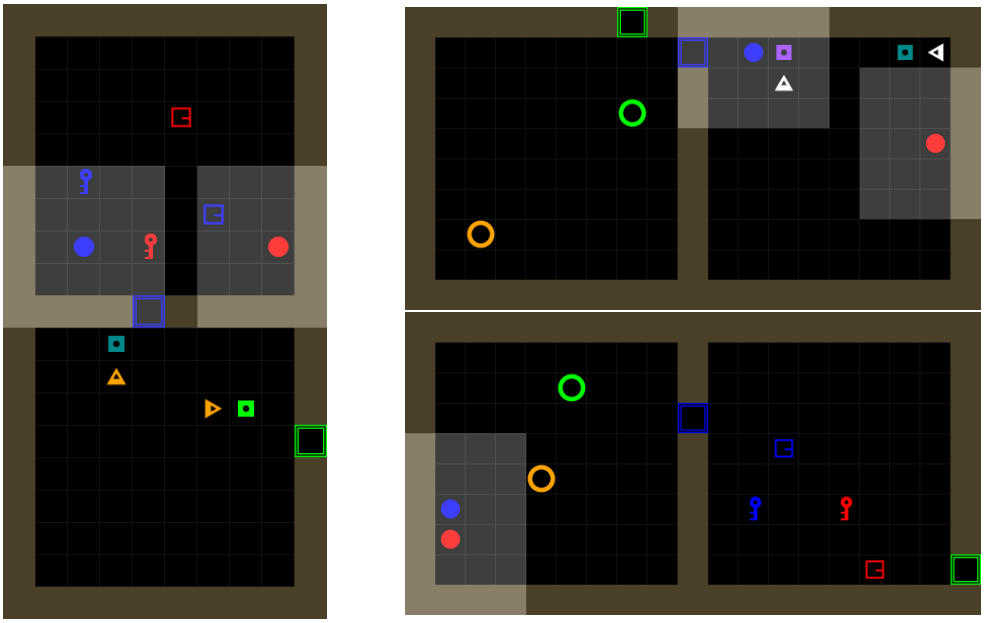


Figure 4: **TwoPuzzleRoom Environment:** following the ideas of [18], we also implement a generator method for creating our escape rooms. Here we show three different generated versions of this environment. In this environment, the above puzzle rooms are generated randomly into a random vertical or horizontal two-puzzle room. The blue door is the intermediary door while the green door is the final goal.

Our agent's vision network was trained with latent observation vectors z of size 32 on 40 by 40 images. We use 4 convolutional layers in both the encoder and decoder. We use a kl loss tolerance of 0.5, capping the maximum of the kl loss at 16. Gradients are clipped by value between -20000 and 20000. Our agent's world model network used 1 layer in its LSTM with hidden vectors of size 256 and 5 Gaussians were used in the MDN. Latent observation vectors in the baseline model was of size 10. All agents were trained with a learning rate of 1e-4 and the VAE and MDN-RNN were jointly trained with the final policy, with a hyperparameter of 1 for both. When the VAE was frozen, the losses for the VAE were no longer calculated.

4 Evaluation

For a comparison study we ran both our world model agent based on [14] and an AE communication agent (MIT) from the "Learning to Ground Multi-Agent Communication with Autoencoders" paper

[2]https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch

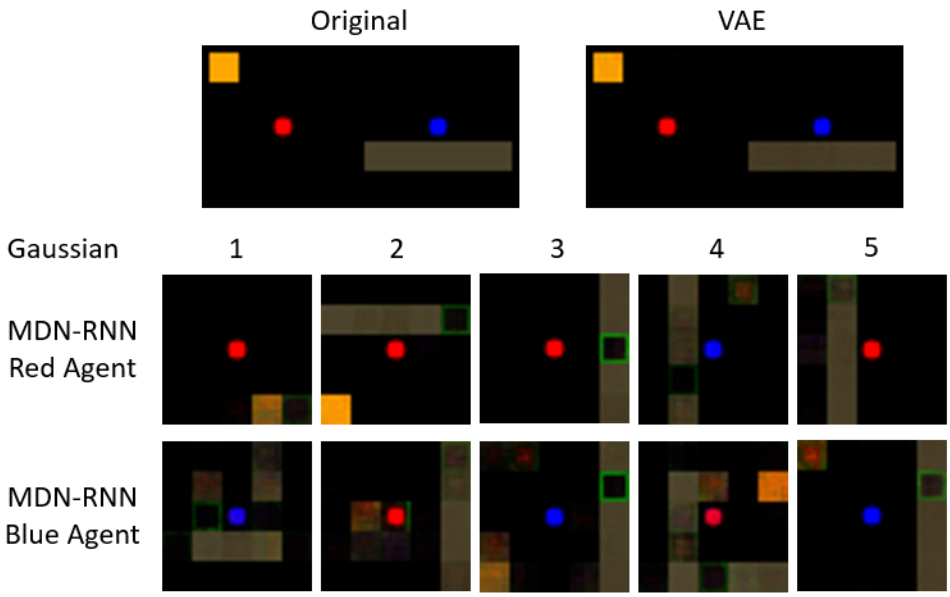


Figure 5: Images from the VAE and the predicted next scene from the MDN-RNN world model. Each multivariate Gaussian used in the mixture is interpreted below.

[4] on our TwoPressurePlate room. We pretrained our agent's autoencoder by choosing random actions for 25k epochs and then freeze this for the remainder of our training run. In this environment the VAE in our model was able to reproduce accurate images of the original agent observation as shown in Figure 5. The decoded MDN-RNN next step predictions for all 5 Gaussians mixtures are also shown for both the red and blue agent in Figure 5. The MDN-RNN had significantly more trouble avoiding noise in its reproduction even after around 400 thousand runs in the double pressure plate room. Recall that these images are the agent's own prediction of the next observation based on the previous observation and the previous action taken.

A few odd phenomena crop up in the decoded LSTM observations. Some of the Gaussians actually model the other agent (red Gaussian 4 and blue Gaussian 2 and 4). Additionally, walls are commonly predicted even if there are no walls in sight or invalid walls are predicted such as in blue agent Gaussian 1. While the MDN-RNN predictions are recognizable as belonging to the two pressure plates environment, there is still obvious noise in the world model most likely due to the unpredictable multi agent nature of the environment and constantly shifting observation window.

Our agent our agent was able to succeed about 50% of the time in the TwoPressurePlate puzzle room; however, it was not able to outperform the MIT agent which converged to a 100% success rate as seen in Figure 6. Additionally the entropy of agent's action probability distribution were recorded to measure the confidence in their actions as seen in Figure 7. When compared to the MIT model our agent was not able to reduce its action entropy significantly over the course of training. We did observe a slight reduction in entropy, however, and this is most likely what led to our success rate rising over the course of training.

We also evaluate training performance for all puzzle rooms and both implemented environments. Table 1 tabulates epochs to success rates for all puzzles and environments. Furthermore, we tabulate the epochs to success rates for each puzzle in the OnePuzzleRoom environment, and we tabulate epochs to success rates for the game order in the TwoPuzzleRoom environment. Those can also be found in Table 1.

In Table 2, we see the highest success rate each puzzle room and environment was capable of achieving with each agent. Although we were forced to stop training at 500k epochs, we see that almost all puzzles and environments have an upwards tending trajectory that suggests that with more training we would have better results.

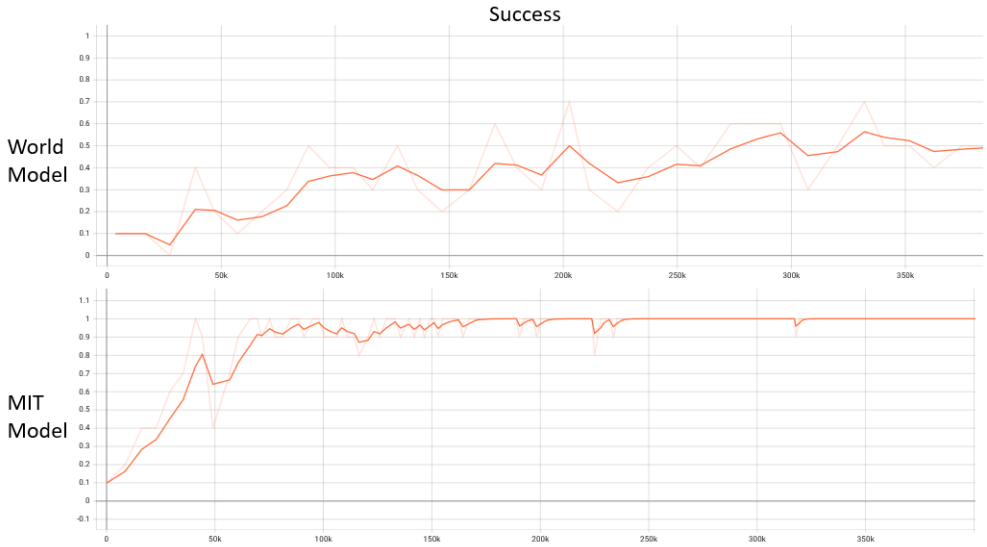


Figure 6: Comparison of game success rates in the TwoPressurePlate environment between our model on top and the MIT model on the bottom.

Figure 7: Comparison of agent action entropies in the TwoPressurePlate environment between our model on top and the MIT model on the bottom.

Table 1: Table of epochs to success for all puzzles and environments

	Agent Model							
	MIT_AE				OURS			
Success Rate:	**25%**	**50%**	**75%**	**100%**	**25%**	**50%**	**75%**	**100%**
Game	**Epochs (Approx.)**							
ColorOrderedLevers	0	5.6k	49k	88k	0	12k	250k	360k
TwoPressurePlate	12k	25k	37k	40k	20k	123k	-	-
SinglePressurePlate	24k	44k	52k	60k	220k	292k	-	-
CycleBoxColor	72k	162k	270k	-	400k	-	-	-
ColorBlindKeys	-	-	-	-	-	-	-	-
OnePuzzleRoom	50k	247k	500k	-	-	-	-	-
ColorOrderedLevers	0	52k	57k	65k	0	0	-	-
TwoPressurePlate	0	50k	64.4k	104k	100k	300k	-	-
SinglePressurePlate	64.4k	297k	419k	-	-	-	-	-
CycleBoxColor	135k	359k	-	-	-	-	-	-
ColorBlindKeys	-	-	-	-	-	-	-	-
TwoPuzzleRoom	-	-	-	-	-	-	-	-
Game 1	30k	232k	-	-	300k	-	-	-
Game 2	-	-	-	-	-	-	-	-

Table 2: Highest success rate attained by each agent

Env	**ColOrdLev**	**TwoPrePlate**	**SingPrePlate**	**CyBoxCol**	**ColBlindKeys**	**OnePuzzle**	**TwoPuzzle**
Agent	**Greatest Success Rate Achieved (Most Recent Epoch of 500k)**						
MIT_AE	100% (500k)	100% (500k)	100% (500k)	70% (299k)	0% (-)	68% (419k)	12% (500k)
OURS	100% (480k)	70% (311k)	50% (455k)	30% (408k)	0% (-)	20% (464k)	0%(-)

5 Discussion

In some of the environments, a sparseness of detail made training difficult. The detail for the keyhole used in the color blindness environment may not have been significant enough to be picked up by the agent vision models, which would have impeded training, as agents could never associate a reward with the keyhole.

Based on the results in Table 1, we find that the relative order of difficulty for our puzzles from least difficult to most difficult are: ColorOrderedLevers, TwoPressurePlate, SinglePressurePlate, CycleBoxColor, and finally ColorBlindKeys. Also, given that ColorBlindKey had trouble converging, we are unsure whether it is due to a poor reward function, agent model, or if with additional training, it would eventually converge.

We faced numerous challenges with the designing and training of our agent. The Gaussian probability distribution produced by the VAE in our agent after training for a high number of iterations eventually tends toward infinity. We took a number of preventative measures to lower the probability of this phenomena significantly. Gradient clipping to clamp the gradient at the maximum gradient value seen during initial training steps and multiple iterations of refactoring the KL-loss made the phenomena significantly less likely. Utilizing these preventative measures, we were able to pretrain the VAE to a stable state that was then frozen and used for longer training runs. Despite this phenomena, the trained VAEs were still able to perform well and generate high quality reproductions of the observed states.

We believe our agent's inability to converge on 100% success rate was mostly due to the amount of uncertainty and noise in the MDN-RNN world model. The ability of an agent to model the world when it only has partial observance is extremely challenging and introduces a lot of unpredictability. The unpredictability of using an agent's observations is especially on the edges of an agent's field of view that changes drastically at every time step. We have come up with a number of potential solutions that future work could use to improve this model and improve results from our model. The MDN-RNN could model the entire 10 by 10 world and predict the entire global view given both agent's actions. This solution would reduce noise significantly and possibly provide very accurate

generation; however, it would also break the partially observable paradigm setup to encourage communication. Slightly tweaking this idea, a location vector could be provided to the MDN-RNN world model to assist in grounding the agent in the environment and reducing the unpredictability of walls. Another idea is to also communicate the other agent's encoded observation similar to the MIT model. World model hidden states may not mean much without also being given the context of the observation that was being predicted. Finally, we could provide the MDN-RNN world model with more information such as the other agent's encoded observation vector. This could potentially prevent the MDN-RNN from trying to model where the other agent will go, as seen in some of the output in Figure 5. Ultimately, using a world model for communication purposes proved to be too noisy for an agent in a partial observable multi-agent environment and inhibited training and success in the long run; some alternative must be used.

6 Conclusion

We think that our work shows a lot of promise in the field of long-horizon collaboration tasks, with plenty of work yet to be done from even in current state-of-the-art models such as the MIT model. We present our new tasks as a benchmark for limited training, and our framework for combining rooms as a method for encouraging agent collaboration. With our formalization, we're able to build increasingly complex larger environments, and we demonstrate the working capability of a framework we introduce to achieve this. We also demonstrate the working framework with agents learning to complete tasks within this implementation. We also demonstrated the limitations of our initial approach within variational inference methods while also discussing methods to improve using them in the future.

References

[1] T. N. Cohen, A. C. Griggs, F. F. Kanji, K. A. Cohen, E. H. Lazzara, J. R. Keebler, and B. L. Gewertz, "Advancing team cohesion: Using an escape room as a novel approach," *Journal of Patient Safety and Risk Management*, vol. 26, pp. 126 – 134, 2021.

[2] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for openai gym," https://github.com/maximecb/gym-minigrid, 2018.

[3] K. Ndousse, "Marlgrid: Gridworld for marl experiments," 2022, https://github.com/kandouss/marlgrid.

[4] T. Lin, M. Huh, C. Stauffer, S.-N. Lim, and P. Isola, "Learning to ground multi-agent communication with autoencoders," 2021.

[5] F. Christianos, L. Schäfer, and S. V. Albrecht, "Shared experience actor-critic for multi-agent reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2006.07169

[6] S. Sukhbaatar, A. D. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *NIPS*, 2016.

[7] J. Kim, "Emergent communication under varying group sizes and connectivities," 2021.

[8] S. Kottur, J. M. F. Moura, S. Lee, and D. Batra, "Natural language does not emerge 'naturally' in multi-agent dialog," in *EMNLP*, 2017.

[9] S. Nicholson, "Peeking behind the locked door: A survey of escape room facilities," 2015.

[10] ——, "The state of escape: Escape room design and facilities," 2016.

[11] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[12] M. Libura, "Sensitivity analysis for minimum hamiltonian path and traveling salesman problems," *Discrete Applied Mathematics*, vol. 30, no. 2, pp. 197–211, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166218X9190044W

[13] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani, *Algorithms*, 1st ed. USA: McGraw-Hill, Inc., 2006.

[14] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014.

[16] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

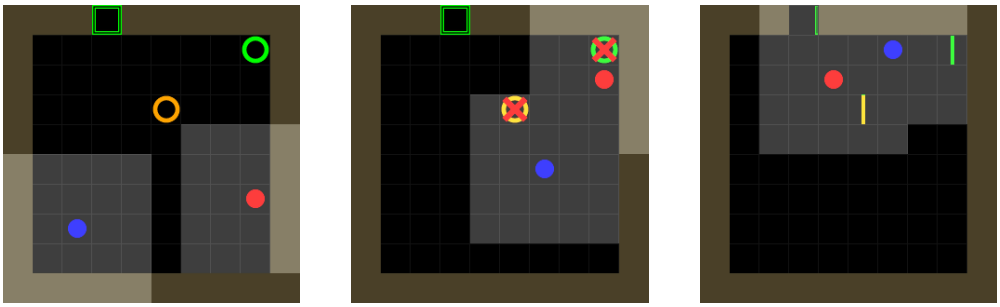
[17] L. Yuan, Z. Fu, J. Shen, L. Xu, J. Shen, and S.-C. Zhu, "Emergence of pragmatics from referential game between theory of mind agents," *ArXiv*, vol. abs/2001.07752, 2020.

[18] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio, "Babyai: A platform to study the sample efficiency of grounded language learning," *arXiv preprint arXiv:1810.08272*, 2018.

A Appendix

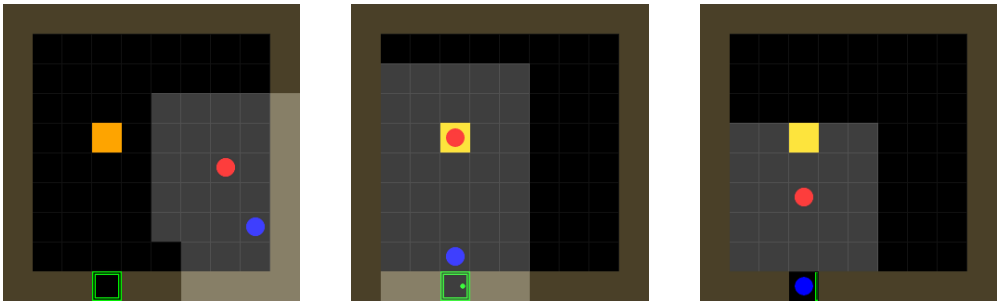
A.1 Additional Environment Details

Here we provide additional graphics for each of the environments. For the environments which have knowledge obfuscation of the world, we also provide the views of the agents. All images for an environment are from the same instance.



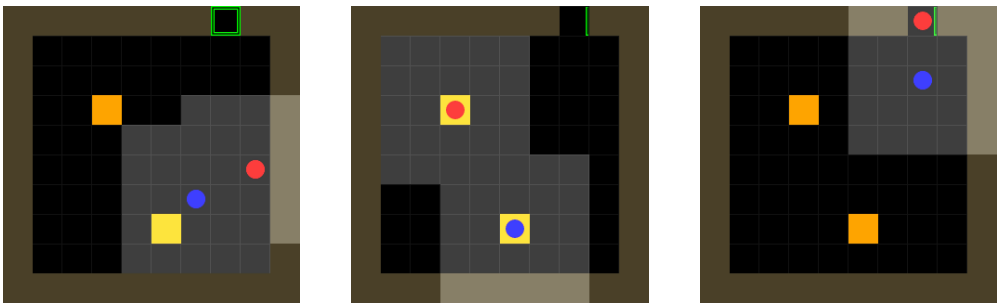
(a) (b) (c)

Figure 8: **ColorOrderedLevers:** (a) The initial state of the puzzle room. (b) If a lever is flipped at the wrong time, making a wrong sequence, or a correctly flipped lever is unflipped, the levers are disabled for 10 steps. (c) On correct sequential flipping of the levers, the door is unlocked and agents can escape.



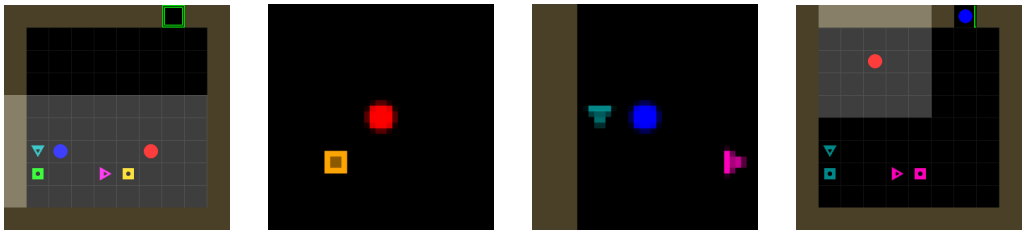
(a) (b) (c)

Figure 9: **SinglePressurePlate:** (a) The initial state of the puzzle room. (b) If any agent is stepping on the yellow pressure plate, the door enters a middle state where the other agent can proceed to open it. This is shown by adding a doorknob to the door, and it disappears again if an agent steps off the pressure plate. If the door is opened in this state, it proceeds to (c). (c) When the other agent opens the door, the agents are then able to escape.



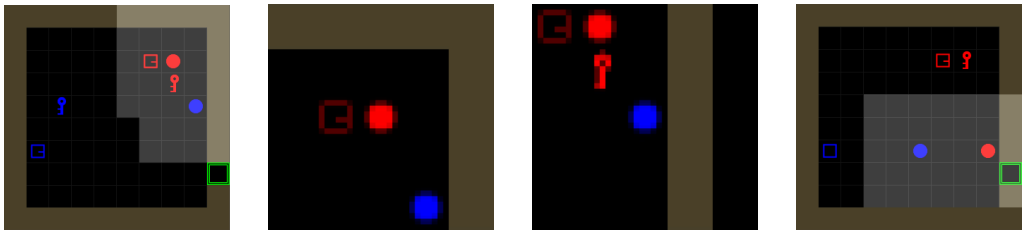
(a) (b) (c)

Figure 10: **TwoPressurePlate:** (a) The initial state of the puzzle room. (b) The door is only opened when both agents are simultaneously stepping on the pressure plates. (c) When the doors are open, the agents can escape.



(a) (b) (c) (d)

Figure 11: **CycleBoxColor:** (a) The initial state of the puzzle room. (b) The view that agent 1 has. In this case, agent 1 is the acting agent, so it's able to see the boxes, which are supposed to be cycled. (c) The view that agent 2 has. Agent 2 is the teacher agent, so it can see the triangular color coded key, which it then has to instruct to agent 1 what color to cycle the box to. (d) When the colors are correctly cycled to, the door opens and the agents can escape.



(a) (b) (c) (d)

Figure 12: **ColorBlindKeys:** (a) The initial state of the puzzle room. (b) The view that agent 1 has. Since agent 1 is the red agent, it can't see the red key, but it can see the red keyhole. (c) The view that agent 2 has. As the blue agent, it can see both the red key and the red keyhole, and it has to instruct agent 1 to the location of the red key. (d) When a key is correctly inserted into a keyhole, it results in that keyhole being "unlocked" and the key consumed, as seen with the blue key. This environment was found to be too difficult for either agent model to complete, and one reason for that may simply be how hard it is to see the red keyhole.

Additionally, in training and all resulting videos appear with the following interface in Figure 13.

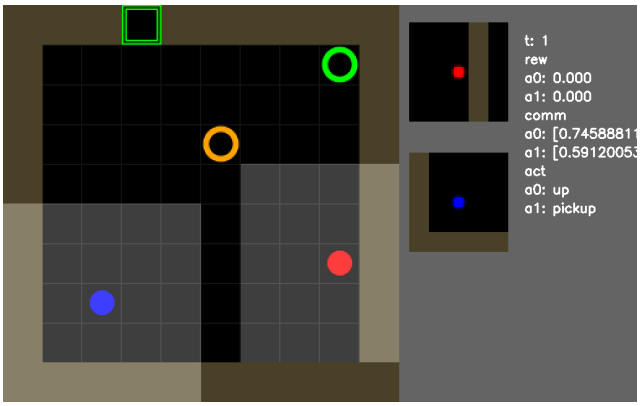


Figure 13: The output rendered from our environment. The code for this was originally written by [3] and [4].

A.2 Division of Work

All team members contributed to presentations, presenting, and proposals/reports. All team members did research on related work. All team members attended the weekly meetings at the agreed upon time and date with the rest of the team. In the end, we did not extend BabyAI's MiniGrid environment

to allow for multiple agents. Instead, during the course of all team members performing research into the matter, we found MarlGrid to serve well as a framework for multi-agent reinforcement learning in a gym environment. All team members worked to build individual escape rooms. Both Adam and Mollie worked on ways to selectively-blind the agent, while Barrett worked on the pressure plate environments. Adam worked on the formal description of an escape room and implemented the final version of the environment designs discussed by the team along their generators following the ideas of [18]. Barrett worked on the design, building, and training of the new agent by extending the model used in [14]. Adam worked out the details of the training hyperparameters, and he and Barrett used their computing resources to train the model. Finally, all teammates worked on completing the final report together before the deadline.

A.3 Reproducibility

To reproduce our work, clone our git repo here[3]. Read the README.md in /marl-grid/env to install our MarlGrid environment (we recommend doing this in a venv), then install the requirements.txt in /marl-grid/one-puzzle-room as well as pytorch for your system. Run the command specified in the README.md file in the one-puzzle-room folder to train.

Data we generated for this report can be found here[4].

[3] https://github.com/lattimer7/EECS692Final

[4] https://1drv.ms/u/s!Ah5imxFvSbVIgpRpCGYWynJgYV9vvg?e=i0aNF7