Construing the Reward Function: Maximum Likelihood Inverse Reinforcement Learning with Limited Cognitive Resources

Sam Liang
Advisors: Mark K. Ho and Thomas Griffiths
Princeton University
saml@princeton.edu

Abstract

Humans lack the cognitive resources to plan optimally when faced with tasks. We cannot consider every factor of a task when determining a plan to solve the task and achieve our goal. Yet, many work on inferring the plan of an expert assume that they are planning optimally. This is unrealistic when humans are the experts. When applying these algorithms to the real world, they may perform poorly, because they fail to take into account that humans do not plan optimally. We would like inference algorithms to learn the optimal plan for a task even when they only have access to sub-optimal data. I present a revised Maximum Likelihood Inverse Reinforcement Learning algorithm that assumes the expert may not be acting optimally, using construals to model an expert acting sub-optimally in which they consider a subset of features when planning. It aims to learn the optimal plan from sub-optimal experts. Performance between the original and construed Maximum Likelihood algorithm on an expert that considers a subset of features is presented.

Contents

1 Introduction

Learning from demonstration is learning how to do something by examining an expert perform the task. It can be divided into two categories: imitation learning and intention learning [6]. Most work in imitation and intention learning assumes the expert is planning optimally [8]. However, this assumption may lead to learning a sub-optimal plan, because the expert is not planning optimally, when we want to learn the optimal plan.

In this paper, I present a construal version of the Maximum Likelihood Inverse Reinforcement Learning algorithm that seeks to address this issue. It eliminates the assumption that the expert must be acting optimally by introducing construals [4] into the algorithm. A construal is a subset of all the details or features of a task, and the idea is that humans form construals when planning for a task [4]. Equally, this paper seeks to understand imitation and intention learning. Thus, I also implement a classifier trained with supervised learning and the Maximum Likelihood algorithm as baselines to compare performance between each other and the construed Maximum Likelihood algorithm.

1.1 Markov Decision Process

A Markov Decision Process (MDP) is a model of the expert's environment and task. It defines a set of states S; a set of initial states S_0; a set of actions A that the expert can take in each state; a transition function $T : S \times A \times S \rightarrow [0, 1]$ that defines the probability of going to state i when taking action a in state j; and a reward function $R : S \times A \times S \rightarrow \mathbb{R}$ that defines the reward of taking an action in state i and arriving in state j. The expert traverses through the MDP by taking actions in each state and arriving in another state until it reaches the goal state, mimicking the expert performing an action in their environment and potentially changing environment conditions before accomplishing their goal in the task.

1.2 Learning From Demonstration Terminology

Formally, in learning from demonstration, an agent takes in as input examples of the expert traversing through the MDP and tries to learn a policy π from those examples that allows it to act and perform like the expert in the MDP [6]. A policy π is a mapping from states to a probability distribution over actions. It specifies with what probability an agent takes an action in a certain state. With a policy, an agent will be able to decide on which actions to take as it traverses

the MDP. Furthermore, an optimal policy is one that maximizes the expected discounted reward which is how much reward on average the expert will acquire if they take this action in a state. As mentioned, most work in imitation and intention learning assume that the expert is using an optimal policy which may not be the case. Furthermore, the examples of the expert traversing through the MDP are called a set of trajectories D. A trajectory $d_i \in D$ is a sequence of state-action pairs $d_i = \langle (s_0, a_0), (s_1, a_1), ..., (s_n, a_n) \rangle$ [6] starting from an initial state s_0, ending at the goal state, and specifying in order which states the expert visited and what actions they took in those states. In both imitation and intention learning, once trajectories are acquired, a modification of them through a process called featurization is required as the data must only contain numbers. A state-feature vector function $\phi : S \to \mathbb{R}^n$ converts states to state-feature vectors [6] and an action-feature vector function $\varphi : A \to \mathbb{R}$ converts actions to numbers (labels). A state-feature vector is a vector of features, usually numbers, that identifies the state. For example, in Figure 1, some gridworld features are {lightgreen, green, red, lightblue, and blue}. A red state could be featurized as ⟨x-coordinate, y-coordinate, a number corresponding to its color⟩ or as a 1-hot vector where each index corresponds to a color.

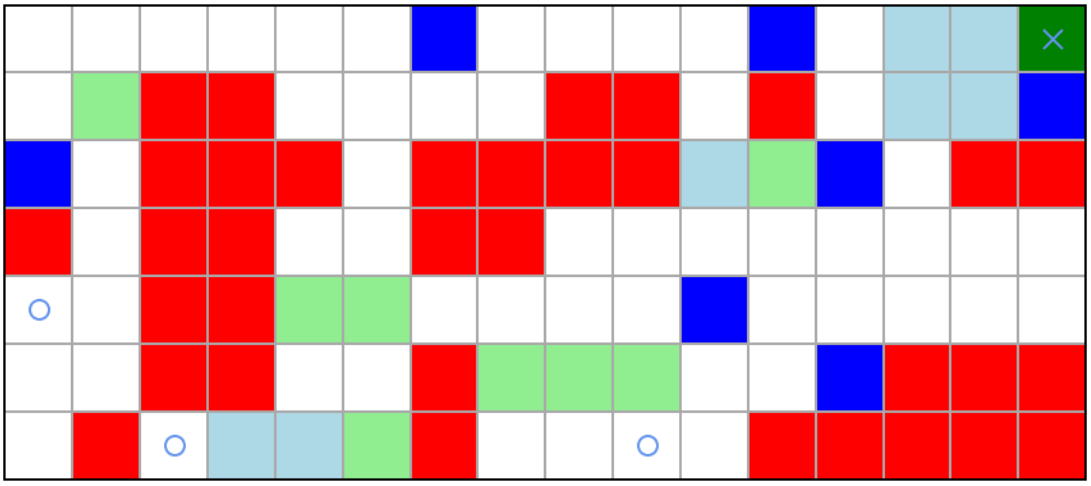

Figure 1: A gridworld example consisting of states with different features indicated its color. Created with MSDM [5].

2 Methodology

2.1 Research Goals

The main goals for this project were two-fold: first to understand imitation learning and IRL, and secondly to extract the optimal policy from watching a sub-optimal expert.

2.2 Gridworld MDPs, Trajectories, and Policies

In this paper, all algorithms, models, and experiments are performed on gridworlds which are created through the MSDM codebase [5]. See Figure 1 for an example gridworld. Gridworlds are MDPs and have all the characteristics of MDPs: states are square tiles; actions are up, left, right, down, and stay; start states are indicated by a blue circle on a white tile; and the goal state is indicated by a blue "x" on the green tile. The goal of the expert is to reach the goal state from the starting states. All gridworlds in this paper have a discount rate of 0.99 and step cost of -1. The discount rate is how much future rewards are discounted and the step cost is the cost of taking any action.

In gridworlds, each action and state has a reward when performing the action and landing on the state. In my MSDM [5] gridworlds, the only action reward is the step cost, and state rewards are associated with its color. In Figure 1, states that are red have a reward of -500. Furthermore, gridworld features are characteristics of states. In a gridworld, some features are the (x, y) coordinates and color of a state. These are the only features I consider in the gridworlds and use in the featurization process. A note on the 1-hot vector featurization process in this paper is that white is not a feature and green (goal state) can be a feature. This is because a vector of zeros will correctly identify a white state. Green is considered the same as white when it has a reward of 0, because the goal state is a known global attribute. If it has a nonzero reward, it is considered a feature.

Meanwhile, the trajectories dataset contains trajectories generated by an expert executing their plan on a gridworld. Trajectories begin at a start state and end at the goal state. All trajectories were generated from scratch by repeatedly running the expert's policy on the gridworld using MSDM [5]. Policies are computed using MSDM's [5] Maximum Entropy Reinforcement Learning (RL) algorithm.

3 Related Work

While much work in imitation learning and inverse reinforcement learning model the expert as acting optimally, there is research on experts acting sub-optimally like humans. The Receding Horizon IRL [6] algorithm introduces a horizon parameter as a form of sub-optimal behavior. The horizon specifies how far into the future an expert plans which is done by limiting how many steps into the future from the current state that the expert takes into account when calculating the action which maximizes the expected discounted future reward from the current state [6]. This certainly addresses humans' cognitive resources as they do not plan infinitely ahead. While I did not use this algorithm as the baseline IRL algorithm, construals can be introduced as another form of sub-optimal behavior alongside the horizon. Meanwhile, online Bayesion goal inference [8] models the expert as a boundedly-rational planning agent that is limited by how far ahead they can plan (horizon idea), and they also continue to update their previously computed plan on-the-fly as they execute it and traverse the MDP. There is also the Bayesian Theory of Mind [2] model that uses a partially observable MDP (POMDP) to outline a human expert's environment. A POMDP is an MDP that has unobservable states and in their case, unobservable states are those that are not in the human's line-of-sight. They model an expert as someone who forms beliefs based on prior knowledge and what they can currently perceive in the POMDP and plans based on their beliefs [2]. The delineated related work all touch upon sub-optimal expert behavior, but they model the problem differently than I do. I use the idea of construals, maintain an infinite horizon planning scheme with a fully observable MDP, and do not recompute the policy.

3.1 Imitation Learning

In imitation learning, the agent learns a function that maps states to actions, $F : S \rightarrow A$ [6]. This is akin to the agent memorizing the correct action to take in all states, because to map states to actions, each state must be uniquely featurized. It is commonly setup as a supervised learning problem in which we model the agent as a classifier and train it to learn this function [6]. The classifier accepts a state as input and returns action scores. The classifier is trained on a dataset of featurized trajectories $X = \{\phi(s) \mid (s, a) \in \bigcup_{d_i \in D} d_i\}$ [6]. After training the classifier, the learned policy is built by deriving a probability distribution over actions from the action scores over all the states in the MDP.

3.1.1 Implementation Details

I implemented a classifier with the Python deep learning library PyTorch [7]. The classifier has a convolutional layer and a fully connected layer. The convolutional layer consists of three convolutional blocks performing 1D convolution. The first block consists of sixteen 3x1 filters, a BatchNorm layer, and a ReLU layer. The second block consists of thirty-two 3x1 filters, a BatchNorm layer, and a ReLU layer. The last block consists of sixty-four 3x1 filters, a BatchNorm layer, and a ReLU layer. The output is flattened before being passed into the final two fully connected layers, the first with a ReLU layer and the second with no extra layers. The final output is scores over the actions. See Figure 2 for an architecture visualization.

I use a {x-coordinate, y-coordinate, state color} feature representation. White is excluded as a color. To train the classifier, I used Cross Entropy Loss and Stochastic Gradient Descent (SGD) with a learning rate of 0.1, weight decay of 0.0001, and momentum of 0.9. I also decreased the learning rate by 0.1 every 10 epochs. For results, see Table 1. For the specific implementation, see Section 8.

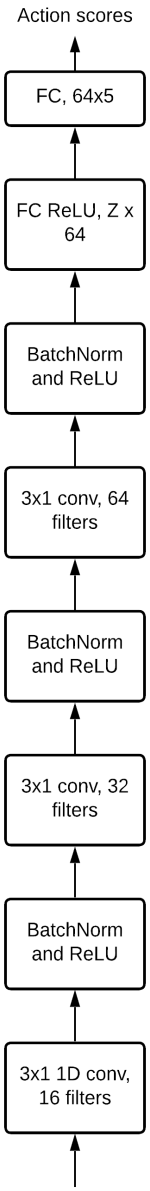

Figure 2: The Classifier Architecture

3.2 Intention Learning

Intention learning, also called inverse reinforcement learning (IRL) when working with MDPs, tries not to memorize which actions to take in which states, but

tries to learn the expert's reward function [6]. The reward function is like an explanation of why the expert takes some action in some state, because we derive policies from the reward function by using any RL policy algorithm. Thus, if we can learn the same reward function as the expert, we will be able to compute the same policy as the expert, and the agent will have successfully learned from the expert. Furthermore, if the expert is planning optimally, then learning a reward function close to the expert's means the agent will be able to plan optimally in any MDP that has the same features and rewards, making intention learning more generalizable. There are many IRL algorithms. In this paper, I implement the Maximum Likelihood IRL (MLIRL) algorithm [1]. In MLIRL, the reward function $r_\theta(s, a)$ is parameterized by a vector of reward weights θ in the following way:

$$r_\theta(s, a) = \theta^T \phi(s, a, s') \tag{1}$$

where ϕ returns a feature vector [1]. Each weight in θ corresponds to one feature in the MDP. The data preparation is similar to imitation learning. We have a dataset of trajectories D that the algorithm learns from. However, we are no longer training a classifier's parameters, but rather, learning θ. MLIRL tries to find the θ that maximizes the log likelihood of seen trajectories (trajectories in D). Mathematically, the log likelihood is

$$L(D|\theta) = \prod_{i=1}^{N} \prod_{(s,a)\in d_i} \pi_\theta(s, a)^{w_i} = \sum_{i=1}^{N} \sum_{(s,a)\in d_i} w_i \log \pi_\theta(s, a) \tag{2}$$

where $L(D|\theta)$ is the log likelihood of the trajectories given the current reward weights θ and w_i is a weight encoding of the frequency of trajectory i in the dataset [1]. The solution to the MLIRL problem is the θ that maximizes $L(D|\theta)$: $\theta^* = \text{argmax}_\theta L(D|\theta)$ [1]. After learning the reward weights, the learned policy is built by passing in the learned reward function into the RL algorithm used during the algorithm. The learned reward function is derived from the reward weights according to Equation (1). See Figure 7 for the full algorithm.

3.2.1 Implementation Details

In my implementation of MLIRL, I use a 1-hot vector encoding the color of the state as the feature representation. The 1-hot vector excludes white as a color. Because of this featurization, the reward weight is equivalent to the actual reward of a feature by Equation 1. π_θ is computed using MSDM's [5] Maximum Entropy

RL algorithm. w_i was selected to be $\frac{1}{N}$ for all i where N is the number of trajectories. In the original paper, the trajectories are all unique so any duplicates are factored into w_i. I allow for duplicate trajectories to be processed, so I do not weight a trajectory during the loss calculation and reward weights update. In my implementation, w_i acts as a scaling factor, because without it, the gradients are too large. Reward weights θ were learned using PyTorch's [7] SGD with a learning rate of 1, weight decay of 0, and momentum of 0.9. No weight decay is used because learning the reward weights correctly means the agent will be able to generalize well. The loss function is the negative log likelihood $-L(D|\theta)$ instead of the log likelihood, because PyTorch minimizes the loss. Minimizing $-L(D|\theta)$ is equivalent to maximizing $L(D|\theta)$. For results, see Table 1. For the specific implementation, see Section 8.

4 Construal

When humans plan for a task, we do not consider all details relevant to the task to come up with a plan. We lack the computational resources required to do so. Although we face this limitation, we still plan to achieve our goals by planning as optimally as possible with our given resources. This can be interpreted as humans selecting the most important details of the task to use during planning. To model this sub-optimal behavior, I use the idea of construals [4]. A construal c is a subset of all the details, or cause-effect relationships, of a task [4] that the expert considers during planning. More importantly, because we operate on gridworlds, a construal can be considered as a subset of all the features of the gridworld. Following from the logic established, experts form construals when planning for a task, because they have limited cognitive resources and cannot consider all task features [4]. Thus, they choose the most important features to still solve the task well [4]. For example, in Figure 1, there are five features: red, blue, lightgreen, lightblue, and green. A construal c from the set of all possible construals C is any subset of these five features in this gridworld. It can be the set {red, blue}, {lightblue}, {red, green, lightgreen}, or {red, blue, lightgreen, lightblue, and green} among others.

4.1 Value of Representation

How does the expert choose the best construals? Each construal has a value of representation (VOR) [4] which is a numerical value that indicates the importance of this construal. The VOR is defined in the following way:

$$VOR = U(\pi_c) - |c| \tag{3}$$

where π_c is the policy derived from construal c, $U(\pi_c)$ is the utility of π_c, and $|c|$ is the size of set c [4]. The utility of π_c can be thought of as the expected future discounted reward over the initial starting states when following the policy. A higher utility means the policy is better. Once the VOR for all construals are calculated, the best construals are the ones with the highest VOR value.

5 Construed Maximum Likelihood IRL Algorithm

5.1 Construing the Reward Function

What does it mean for an expert to disregard features not in their construal? Because I adopt the reward function representation $r_\theta(s, a) = \theta^T \phi(s, a, s')$ from MLRIL, I chose to define this as the expert disregarding and ignoring the rewards associated with those features. Thus, when an expert construes their reward function, they zero out the weights in θ of the features not in their construal. This leads to a reward function where the reward of taking an action and landing in another state changes because the reward weights of those features of the state are now 0, so they do not contribute to the reward anymore. They are ignored. Hence, the Construed Maximum Likelihood IRL algorithm (CMLIRL) adds the assumption that the expert construes their reward function in this manner to introduce construals into MLIRL [3].

In order for the reward weights to be learned, an extra assumption that the expert chooses their construals according to a differentiable probability distribution based on the VOR must be included [3]. This is so that the gradients can be computed. For example, if they only chose the best construal which is equal to a max operation over the construals, then the gradient cannot be calculated because max is not differentiable [3].

5.2 Loss Function

Starting from MLIRL, we still find the $\theta^* = \text{argmax}_\theta L(D|\theta)$ to maximize the likelihood of seen trajectories [1]. However, the likelihood depends on all the construals now. This is modelled through this probability [3]:

$$P(D|\theta) = \sum_{\pi_c} P(D, \pi_c|\theta) = \sum_{\pi_c} P(D|\pi_c)P(\pi_c|\theta) = \sum_{\pi_c} \sum_{i=1}^{N} \sum_{(s,a)\in d_i} \log \pi_{c,\theta}(s,a) P(\pi_c|\theta) \tag{4}$$

This is the loss function used to update the reward weights. We sum over all the construed policies because the likelihood of a trajectory depends on which construed policy the expert uses which we do not know. $P(\pi_c|\theta)$ is the expert's probability distribution over construals from the algorithm assumption.

5.3 Implementation Details

Putting everything together, the algorithm is outlined below. Construed_policies is a mapping from construals to policies derived from those construals.

Algorithm 2 Construed Maximum Likelihood IRL Algorithm

Input: MDP, features ϕ, trajectories $\{d_1, d_2, ..., d_N\}$, number of epochs M, learning rate α, SoftMax function σ, utility function U.
Initialize: Choose random initial reward weights θ.
for $i \in$ [1, M] **do**
  Initialize VOR and construed_policies δ
  Compute set of all construals C
  for $c \in C$ **do**
    Compute $\pi_{c,\theta}$
    $u = U(\pi_{c,\theta})$
    δ[c] $= \pi_{c,\theta}$
    VOR[c] $= u - |c|$
  endfor
  $L = 0$
  for $\pi_{c,\theta} \in \delta$ **do**
    $L \leftarrow L + \sum_{i=1}^{N} \sum_{(s,a)\in d_i} \sigma(VOR)_c \ \log \pi_{c,\theta}(s,a)$
  endfor
  $\theta \leftarrow \theta + \alpha \nabla L$
endfor

In my implementation of Algorithm 2, I imposed a further assumption that the expert uses construals of size 2. This mimics the behavior that the expert can only consider 2 details of the task. This is not a necessary assumption for the algorithm but makes it so that computing all possible construals C is computationally easier during algorithm execution. I use a 1-hot vector encoding the color of the state except white as the feature representation. I use the initial value as the utility function. The initial value is the expected future discounted reward from start states. $\pi_{c,\theta}$ is computed using MSDM's [5] Maximum Entropy RL algorithm. Moreover, I assume the expert chooses construals according to a SoftMax distribution $\sigma(VOR)$ where $\sigma(VOR)_c$ is the probability associated with construal c. Reward weights θ were learned using PyTorch's [7] SGD with a variable learning rate, weight decay of 0, and momentum of 0.9.

6 Experiments and Evaluations

6.1 Imitation Learning Classifier and Maximum Likelihood IRL

Evaluations are done using a policy's initial value as the quantitative measure. The initial value is defined as the following [3]:

$$\sum_{s_0 \in S_0} P(s_0)V(s_0) \tag{5}$$

where S_0 is the set of initial states, $P(s_0)$ is the probability of starting at s_0, and $V(s_0)$ is the expected future discounted reward at s_0. A more positive initial value is better. We want the classifier's and MLIRL's learned policies' initial values to match the expert's initial value.

Moreover, policy graphs which are the learned policies and expert's policies plotted on their respective gridworlds and their state value maps are generated for qualitative comparison. State value maps display the expected future discounted reward on every state using the specified policy: $V_\pi(s)$ for all $s \in S$. See Sections 10.2 to 10.5 for these results.

For MLIRL, the learned reward weights are also compared to the actual rewards. The goal is to see how well the classifier and MLIRL can learn an expert's policy and how well their learned policies apply to novel gridworlds.

Table 1 presents the initial values of the policies that the classifier and MLIRL

learned from four different gridworlds (GW) against the initial values of the expert policy. This tests how well the classifier and MLIRL can learn from the expert. Initial values are calculated with MSDM [5]. GW1 has 5 features {lightgreen, green, red, lightblue, and blue}. GW2 and GW3 have the same 5 features {lightgreen, green, red, black, and blue}. GW4 has 8 features {lightgreen, grey, purple, magenta, yellow, black, orange, and red}. For each gridworld example, both the classifier and MLIRL algorithm were trained from the same trajectories generated from the expert who acts optimally, using batches of size 128 for 50 epochs.

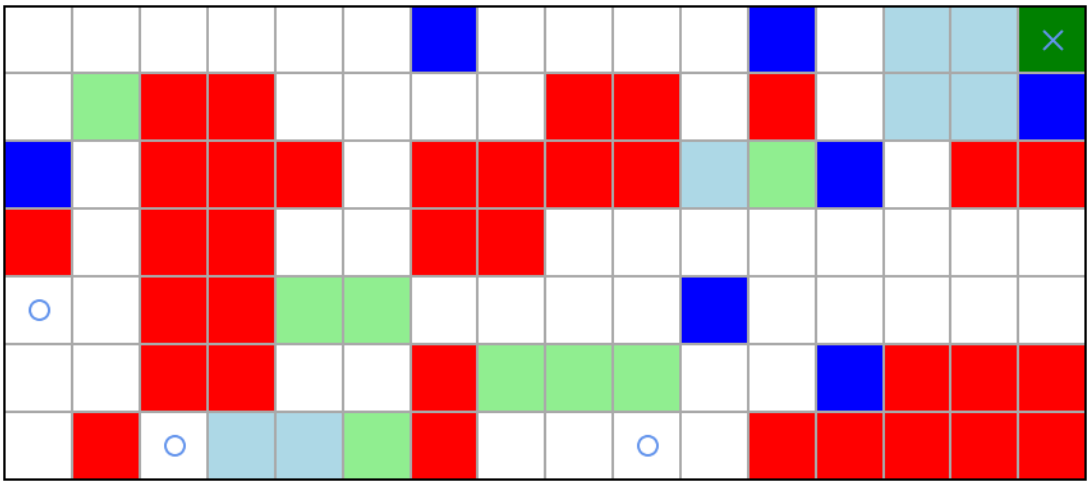

Figure 3: Gridworld 1. Rewards are: {lightgreen: 0, green: 10, red: -500, lightblue: 0, and blue: -5}. Created with MSDM [5].

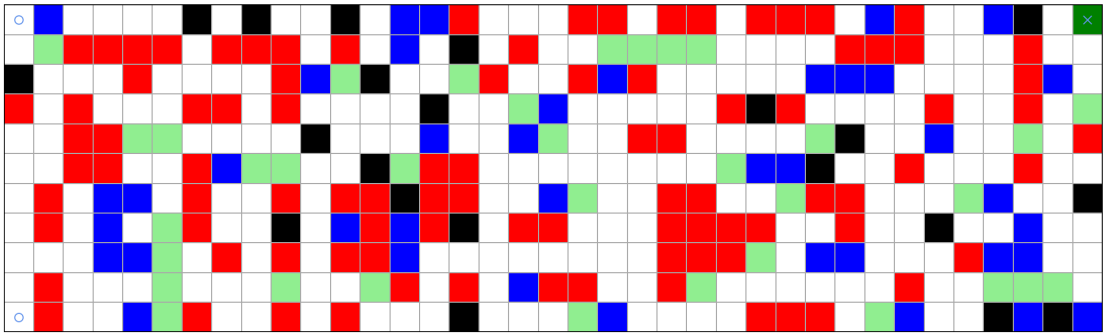

Figure 4: Gridworld 2. Rewards are: {lightgreen: 0, green: 10, red: -12, black: -5, and blue: -1}. Created with MSDM [5].

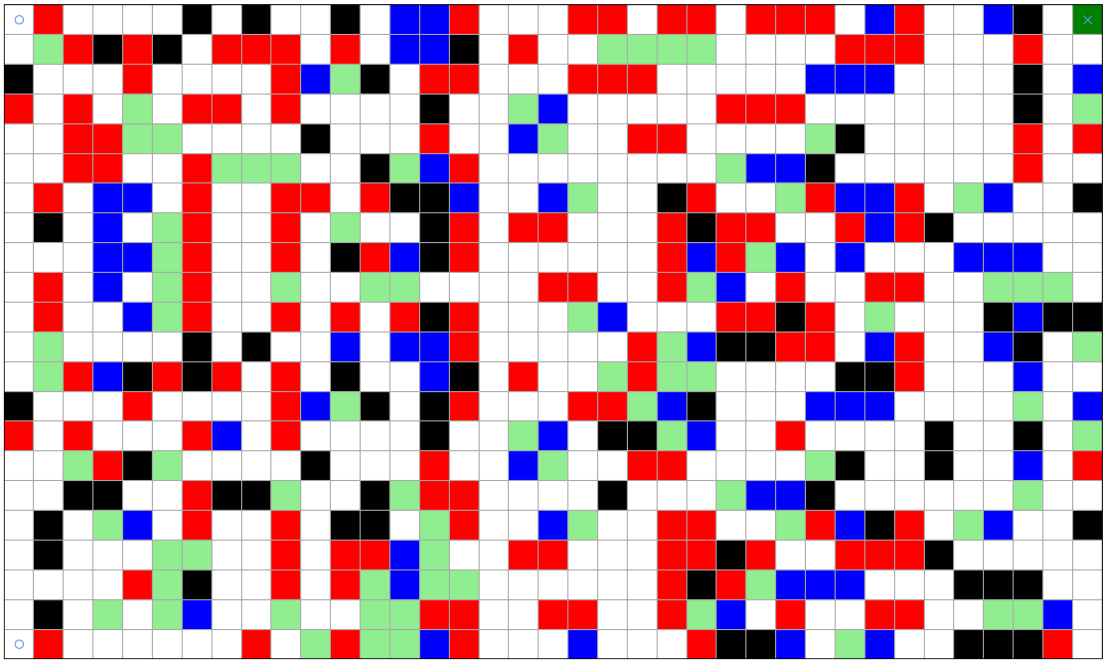

Figure 5: Gridworld 3. Rewards are: {lightgreen: 0, green: 10, red: -12, black: -5, and blue: -1}. Created with MSDM [5].

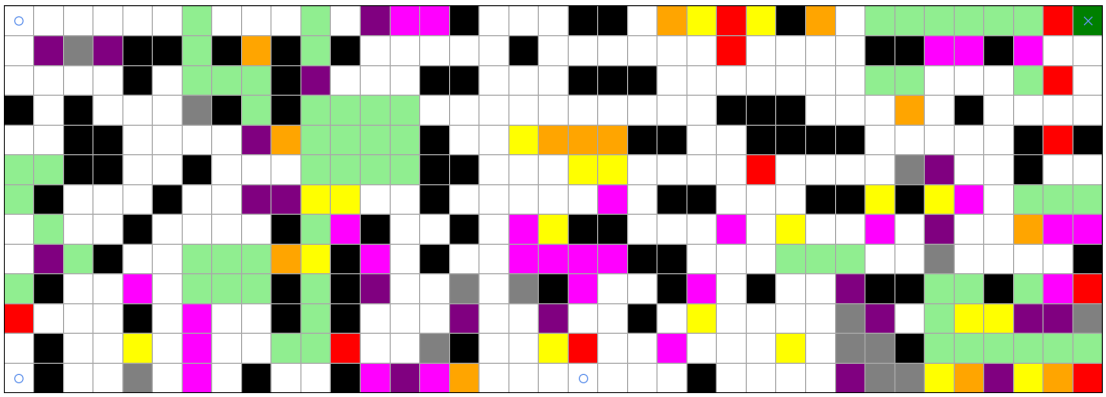

Figure 6: Gridworld 4. Rewards are: {lightgreen: 0.5, grey: -1, purple: -2, magenta: -5, yellow: -7, black: -10, orange: -13, and red: -20}. Created with MSDM [5].

Policy	10000 Trajectories	100000 Trajectories
Classifier	-42.59 | -68.92 | -81.34 | -63.07	-27.84 | -64.24 | -79.76 | -
Maximum Likelihood	-38.29 | -63.63 | -71.33 | -60.43	-38.41 | -63.64 | -71.22 | -
Optimal Expert	-24.21 | -62.72 | -70.41 | -58.63	-24.21 | -62.72 | -70.41 | -

Table 1: Initial values of my implementations compared to the expert policy's initial values. For a given cell, the initial values are for the learned policy of gridworld 1, 2, 3, and 4, respectively.

Reward Weights	GW1
Learned	-0.11, -0.06, -4.93, 0.89, -20.19
Actual	0, 0, -5, 10, -500

Reward Weights	GW2
Learned	-0.02, -1.01, -4.93, 5.45, -12.03
Actual	0, -1, -5, 10, -12

Reward Weights	GW3
Learned	0.00, -0.94, -4.95, 7.14, -11.95
Actual	0, -1, -5, 10, -12

Reward Weights	GW4
Learned	0.54, -1.05, -2.09, -4.95, -7.33, -9.83, -8.41, -17.51
Actual	0.5, -1, -2, -5, -7, -10, -13, -20

Table 2: Learned rewards compared to the actual rewards. In GW1, from left to right, the rewards correspond to lightgreen, lightblue, blue, green, red. In GW2 and GW3, the rewards correspond to lightgreen, blue, black, green, red. In GW4, the rewards correspond to lightgreen, grey, purple, magenta, yellow, black, orange, and red.

From Table 1, we see that the classifier benefits from seeing more trajectories per epoch. In each gridworld example, the initial values of the classifier's policy became more positive by training on 100000 trajectories. This occurs because the classifier is a deep neural network, so it has 10000+ parameters that need to be learned. Seeing more new trajectories helps it learn. On the other hand, MLIRL does not benefit from seeing more trajectories. It's initial values do not change from 10000 to 100000 trajectories. In these experiments, the MLIRL deals with 5 reward weights, because a 1-hot vector encoding on only the colors is used. It

only needs to learn 5 reward weights instead of 10000+, so it needs less data to train on for good results.

Between the classifier and MLIRL, performance is similar with MLIRL doing better than the classifier. There is an anomaly in gridworld 1 on 100000 trajectories: the classifier does better than MLIRL. This offers insight into how MLIRL works. This anomaly occurs because of the -500 reward associated with the red feature. Because for a given set of trajectories, there are multiple reward weights that can maximize its likelihood, the MLIRL will never learn the correct reward weight of -500 unless the random weights are initialized close to -500. I used a Gaussian distribution with mean 0 and standard deviation of 1 for initializing reward weights, so the MLIRL algorithm never learned a reward weight close to -500 for that feature. From Table 2, it learned a reward weight of -20.19. Thus, it thinks that its not as bad to step on a red state, when in reality it should never step on red. This leads to a more negative initial value. The classifier does not suffer from this issue, because it does not care about the weights and tries to copy the expert in every state.

From Table 2, most of the learned rewards match closely to the actual rewards. This means the MLIRL does well in learning the expert's reward function and generates an optimal policy similar to the expert's. This is reflected in the MLIRL initial values being close to the expert's initial values.

Additionally, the differences between gridworlds 1, 2, and 3, are the size of the gridworld and the state arrangement. Gridworld 1 is the smallest and gridworld 3 is the largest. They have the same number of features. From the results, the classifier and MLIRL perform well even as the size increases. Meanwhile, the difference between the first three gridworlds and gridworld 4 is the number of features. Gridword 4 has 8 features and the others have 5. Again, it shows that they both still perform well as the number of features increases. We can conclude that as the number of features increases, the number of reward weights also increases, so the MLIRL algorithm should benefit from training on more trajectories.

Moving on to novel gridoworlds, Table 3 shows the initial values of the learned policies applied to novel gridworlds. This tests if the learned policies can generalize to other gridworlds. Note, these novel gridworlds can only vary in size and state configuration. The number of features must remain the same as I do not deal with how to handle unseen features. Each learned policy derived from the same gridworld is tested on the same novel gridworld. Learned policies from different gridwords are tested on different novel gridworlds for a total of four novel gridworlds. The novel gridworlds are in Sections 10.2 to 10.5 as well as the policy

graphs and state value maps.

Policy	GW1	GW2	GW3	GW4
Classifier	-2241.32 | -2179.14	-200.11 | -208.63	-262.47 | -183.63	-436.25 | -
Maximum Likelihood	-48.06 | -48.42	-38.18 | -38.20	-61.26 | -61.05	-65.24 | -
Optimal Expert	-24.02	-37.75	-60.25	-64.43

Table 3: Initial values of the learned policies on new, unseen gridworlds. For a given cell, the initial values are for the learned policy of that gridworld trained with 10000 or 100000 trajectories applied to unseen gridworlds, respectively. Each policy trained from the same gridworld is applied to one novel gridworld, so the expert policy row only has one initial value.

Results demonstrate that the MLIRL is generalizable, while the classifier is not. On every novel gridworld, the classifier performs poorly. This is because the classifier does not know what to do when it sees a state-feature representation that it has never seen before. For example, using our current featurization, the classifier will not know what to output when it sees a state-feature representation with (x, y) coordinates that it has never seen before during training. In an effort to make the classifier more generalizable, I used the MLIRL 1-hot vector featurization for the classifier and trained it on Figure 3. See subsection 10.6 for the results. With this featurization, the classifier does not learn at all. It's initial value is -4084.27 compared to the expert's -24.21. This reaffirms that the classifier is memorizing which action to take and must featurize every state uniquely. With this featurization, a blue state at coordinates (1, 1) and a blue state at coordinates (4, 3) are treated the same. But, if the expert takes different actions at these blue states, the classifier will be confused because the gradients will push in different directions. Thus, it never learns with this featurization and cannot be made more general this way. On the other hand, the MLIRL does well on novel gridworlds, because it learned the expert's reward function well. Since a policy is derived from the reward function, it will do well on any novel gridworld that has the same features and rewards associated with those features.

6.2 Construed Maximum Likelihood IRL

Evaluations are done using a policy's initial value as the quantitative measure and policy graphs and state value maps as the qualitative measure. Learned and actual rewards are also compared. The goal is to see if CMLIRL can learn the actual rewards when learning from trajectories of an expert that construes their reward function.

Table 4 compares the initial values of CMLIRL and MLIRL and Table 5 compares the learned rewards of CMLIRL and MLIRL with the actual rewards. Trajectories are from a construed expert. Each trajectory is generated by the expert after they sample a construal according to a SoftMax distribution over their VOR. The same 10000 trajectories are used for training the CMLIRL and MLIRL for 10 epochs with batches of size 128. See Sections 10.7 to 10.10 for the policy graphs, state value maps, GW5, GW6, and GW7.

Policy	GW1	GW5	GW6	GW7
CMLIRL	-25.48	-193.63	-580.05	-407.83
MLIRL	-90.09	-130.21	-135.35	-191.22
Optimal	-24.21	-70.90	-79.26	-104.13

Table 4: Initial values of the policies that CMLIRL and MLIRL learned.

Reward Weights	GW1
CMLIRL	0.59, -2.89, -4.99, 0.16, -40.13
MLIRL	-0.09, -0.08, -4.78, -0.18, -14.09
Actual	0, 0, -5, 10, -500

Reward Weights	GW5
CMLIRL	-1.37, 0.49, -0.19, 0.31, 0, -9.65, 0.55, -11.14
MLIRL	-0.04, 0.05, -0.01, 0.04, 0.01, -9.66 -0.01 -11.17
Actual	0.5, -1, -2, -5, -7, -10, -13, -50

Reward Weights	GW6
CMLIRL	-5683.33, 58.57, 335.17, 50.06, -2159.2, 98.96
MLIRL	-14.42, 0.44, -1.19, 0.38, -1.15, -0.47
Actual	-3.8, -4, -5, -4.75, -3, -5

Reward Weights	GW7
CMLIRL	0.38, -1.82, 0.05, -2.01, 0.26, 1.13
MLIRL	-0.09, -1.96, 0.02, -1.83, 0.05, -0.27
Actual	-3.8, -4, -5, -4.75, -3, -5

Table 5: CMLIRL, MLIRL, and actual rewards compared. From left to right in each GW, the rewards correspond to the same features for all three.

The initial experiment showed promising results. On GW1, CMLIRL outperformed MLIRL and closely matched the optimal policy's initial value. Looking

at the rewards though, it only learns a more negative reward for the last feature that is closer to -500. Subsequent experiments on GW5, GW6, and GW7 show that CMLIRL fails to learn the actual rewards and performs worse than MLIRL. Surprisingly, MLIRL performs decently.

Looking at the learned reward weights for GW5, we see that MLIRL pushes all reward weights close to 0 except for two reward weights. There are two reasons for this: MLIRL treats all ignored features as if they were white squares and the SoftMax distribution for the construals was highly skewed. In this gridworld, there was one construal that had a probability of nearly 1. This means that although the expert is construing, they are almost always picking 1 construal each time they traverse the gridworld. Because of this, they consistently visit the states with the ignored features which the MLIRL algorithm thinks then is a white square because the expert does not avoid it. Thus, this pushes the reward weights to zero. Why does CMLIRL fail to learn the true rewards? Initially, I thought that the issue was the highly skewed SoftMax distribution. If the SoftMax policy heavily leans towards a select few construals where many construals have probabilities close to 0 of being chosen, then certain features may not be updated, because they are always masked out.

I tried to make the SoftMax distribution less skewed. Howoever, looking at GW6 where 5 construals had a probability greater than 0.1 of being chosen, CMLIRL still performs poorly. The large reward weights are due to the learning rate used for CMIRL. Finally, I removed most of the white states from GW6 to put more importance on the states with rewards associated to them and the construals, but the CMLIRL still did not perform well.

7 Conclusion

7.1 Imitation and Inverse Reinforcement Learning

In conclusion, I have showed the differences between imitation learning and IRL through implementation of a classifier and MLIRL algorithm. From the results, both perform well in learning from the expert, but MLIRL performance depends on initialization as learning the reward function is a non-convex problem, whereas the classifier can copy the expert exactly. Furthermore, MLIRL is generalizable to any MDP with the same features, and will perform almost optimally on MDPs with the same rewards as well, if it learned well from an expert acting optimally.

7.2 Construed Maximum Likelihood IRL Algorithm

To combat the problem of trying to learn the real rewards from a sub-optimal expert, I presented the CMLIRL algorithm that uses the idea of construals to model a sub-optimal expert. Results show that CMLIRL fails to solve this problem. Further investigation is required to improve and iterate upon this version of CMLIRL.

7.3 Future Work

Many researchers have already explored other ways of modelling sub-optimal behavior like those in Section 3. As mentioned, there must be more work done on this version of CMLIRL. It is also imperative to speed up policy generation from construals as calculating a policy for all possible construals is computationally expensive. Similarly, one can integrate the idea of construals into other algorithms that have other forms of sub-optimal human behavior included already to see how they will perform. Finally, researching the best ways to featurize states is also important. For example, how can we make intention learning generalize better? Overall, IRL with sub-optimal behavior is a new and growing field with potential and many avenues of to explore and build upon.

8 Code

The code for the implementations of the classifier, MLIRL, and CMLIRL is available at https://github.com/iamsamliang/IRL_IW in algorithms.py and dataset.py.

9 Honor Code

This project represents my own work, in accordance with the University regulations.
/s/ Sam Liang

References

[1] M. Babes, V. N. Marivate, K. Subramanian, and M. L. Littman. Apprenticeship learning about multiple intentions. In *ICML*, pages 897–904, 2011. URL https://icml.cc/2011/papers/478_icmlpaper.pdf.

[2] C. L. Baker, J. Jara-Ettingera, R. Saxe, and J. B. Tenenbaum. Rational quantitative attribution of beliefs, desires and percepts in human mentalizing. *Nat Hum Behav*, 1, 0064, 2017. URL https://doi.org/10.1038/s41562-017-0064.

[3] M. Ho. email, Dec. 2021.

[4] M. K. Ho, D. Abel, C. G. Correa, M. L. Littman, J. D. Cohen, and T. L. Griffiths. Control of mental representations in human planning. *CoRR*, abs/2105.06948, 2021. URL https://arxiv.org/abs/2105.06948.

[5] M. K. Ho, C. G. Correa, and D. Ritter. Models of Sequential Decision Making (msdm), 5 2021. URL https://github.com/markkho/msdm.

[6] J. MacGlashan and M. L. Littman. Between imitation and intention learning. In *IJCAI*, pages 3692–3698, 2015. URL http://ijcai.org/Abstract/15/519.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[8] T. Zhi-Xuan, J. L. Mann, T. Silver, J. B. Tenenbaum, and V. K. Mansinghka. Online bayesian goal inference for boundedly-rational planning agents. *CoRR*, abs/2006.07532, 2020. URL https://arxiv.org/abs/2006.07532.

10 Appendix: Algorithms, Policy Graphs, and State Value Maps

All graphs created with MSDM [5].

10.1 Maximum Likelihood IRL Algorithm

Algorithm 1 Maximum Likelihood IRL

Input: MDP$\backslash r$, features ϕ, trajectories $\{\xi_1, \ldots, \xi_N\}$, trajectory weights $\{w_1, \ldots, w_N\}$, number of iterations M, step size for each iteration (t) α_t, $1 \leq t < M$.

Initialize: Choose random set of reward weights θ_1.

for $t = 1$ **to** M **do**

Compute Q_{θ_t}, π_{θ_t}.

$$L = \sum_i w_i \sum_{(s,a)\in\xi} \log(\pi_{\theta_t}(s,a)).$$

$\theta_{t+1} \leftarrow \theta_t + \alpha_t \nabla L$.

end for

Output: Return $\theta_A = \theta_M$.

Figure 7: Maximum Likelihood IRL Algorithm. Taken from [1].

10.2 Classifier and MLIRL Results on Gridworld 1

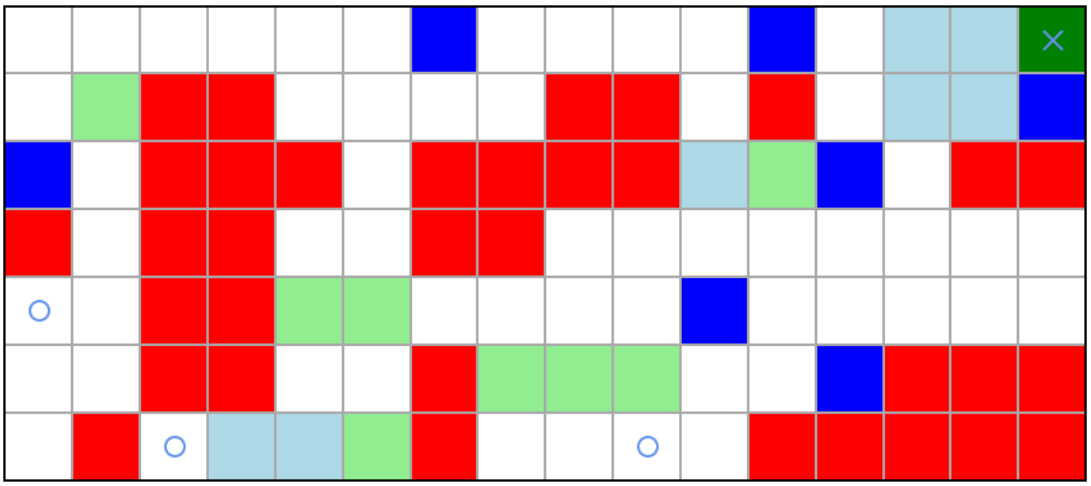

Figure 8: Gridworld 1

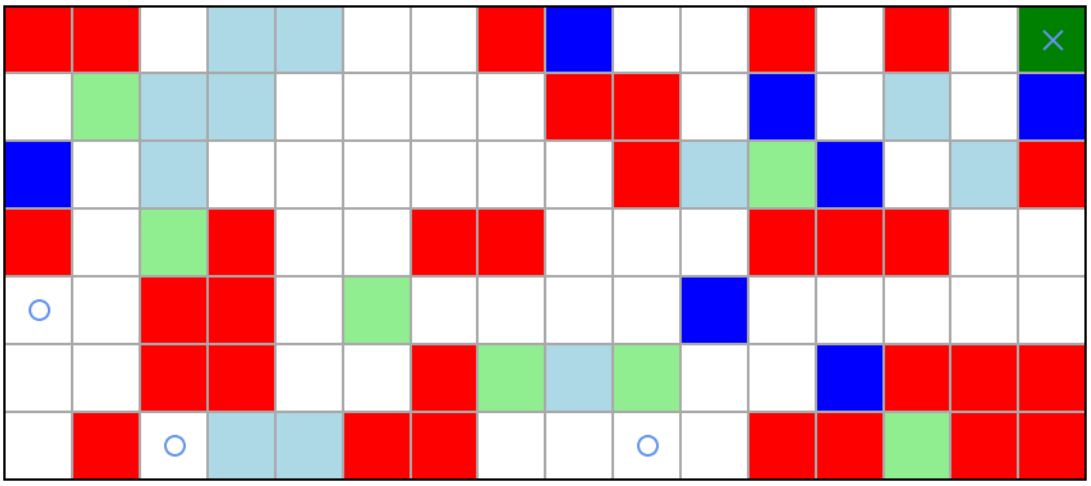

Figure 9: Novel Gridworld for Gridworld 1

Trained on 10000 trajectories with batch size of 128 for 50 epochs.

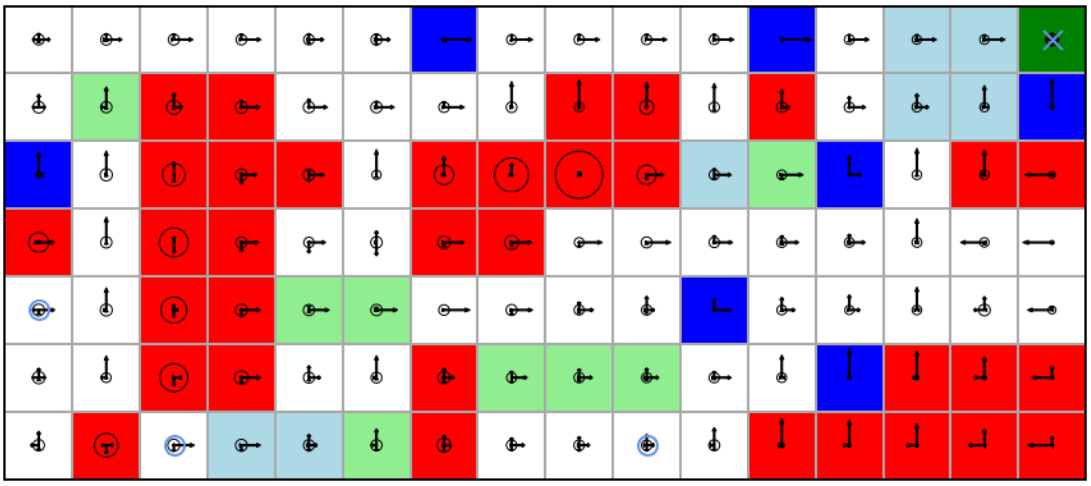

Figure 10: Classifier Policy. The arrows within a state represent the action to take: left, right, up, and down. The stay action is a circle. The length of the arrow corresponds to the probability of taking this action in this state.

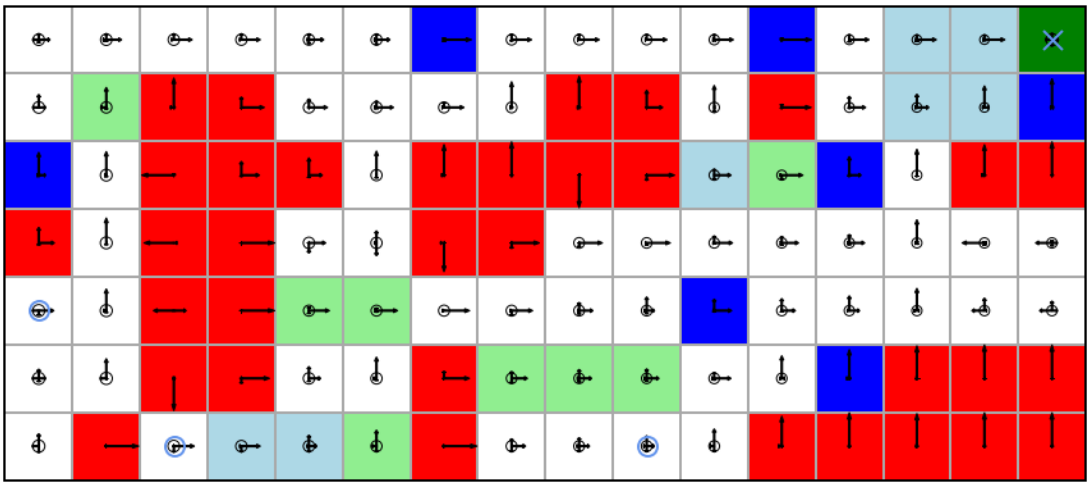

Figure 11: MLIRL Policy

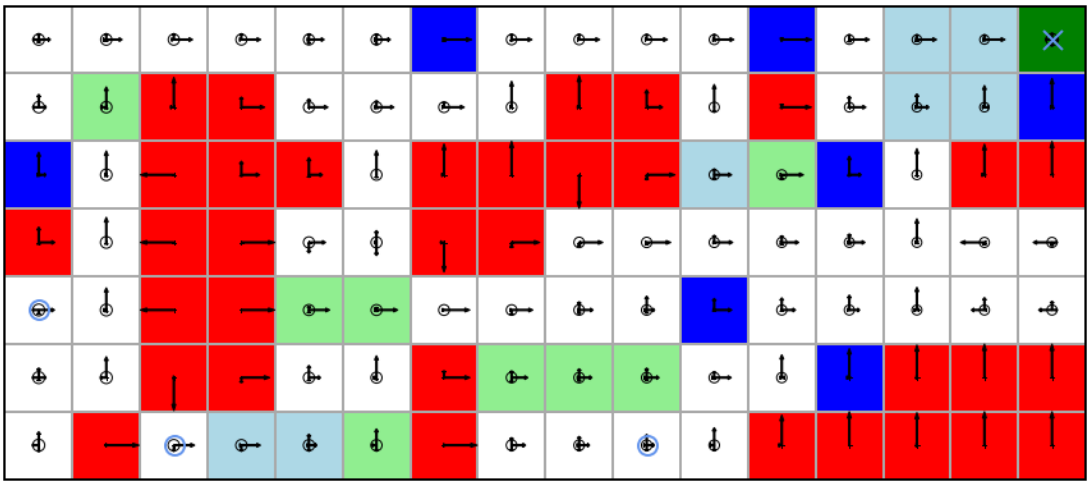

Figure 12: Expert Policy

-37.74	-35.62	-33.57	-31.28	-29.52	-26.66	-18.72	-16.46	-14.67	-10.51	-5.67	1.58	3.48	5.64	7.90	0.00
-38.93	-37.60	-667.70	-210.51	-33.16	-28.34	-26.60	-21.46	-107.42	-182.28	-9.60	-108.44	1.27	2.99	3.28	6.55
-42.30	-39.90	-2238.62	-887.11	-159.06	-34.87	-344.21	-889.40	-2384.64	-324.14	-10.65	-8.20	-0.27	1.17	-91.20	-627.67
-451.12	-45.38	-2985.69	-473.76	-48.60	-37.91	-687.62	-222.13	-22.64	-12.46	-9.63	-7.06	-4.34	-1.27	-12.18	-14.26
-60.30	-52.06	-2149.19	-366.55	-38.03	-33.89	-31.76	-29.40	-20.69	-15.83	-10.57	-8.11	-6.03	-4.87	-12.46	-16.03
-83.67	-98.63	-920.58	-211.87	-38.65	-35.98	-426.81	-58.40	-24.46	-18.44	-15.52	-12.58	-9.93	-48.22	-230.56	-497.54
-117.39	-955.79	-44.18	-41.81	-40.00	-38.93	-913.55	-117.93	-32.82	-23.08	-22.16	-64.65	-156.14	-609.49	-936.36	-1324.86

Figure 13: Classifier State Value Map

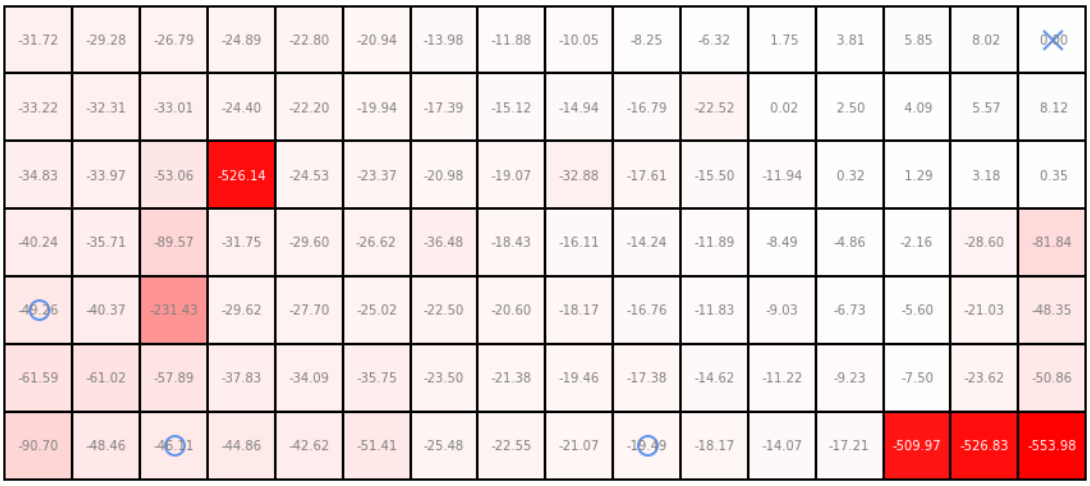

Figure 14: MLIRL State Value Map

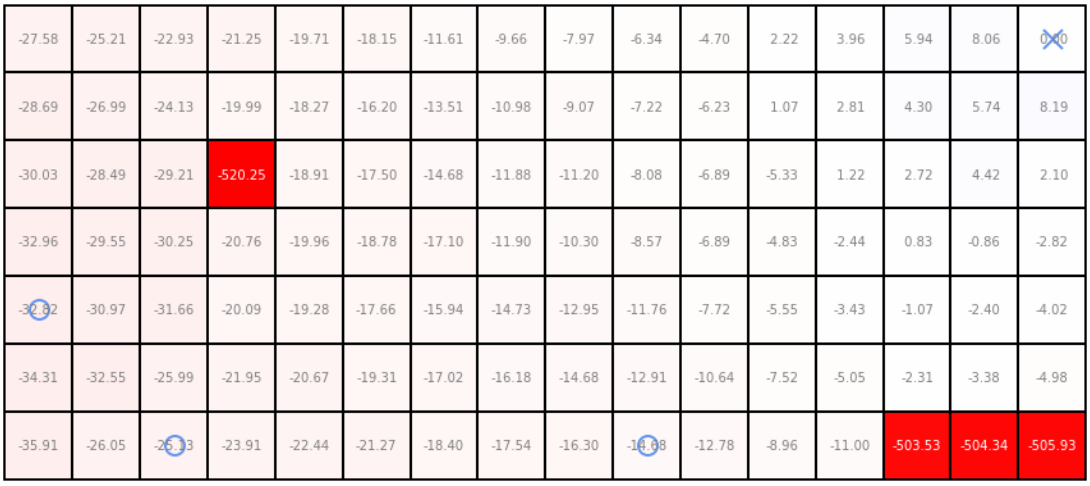

Figure 15: Expert State Value Map

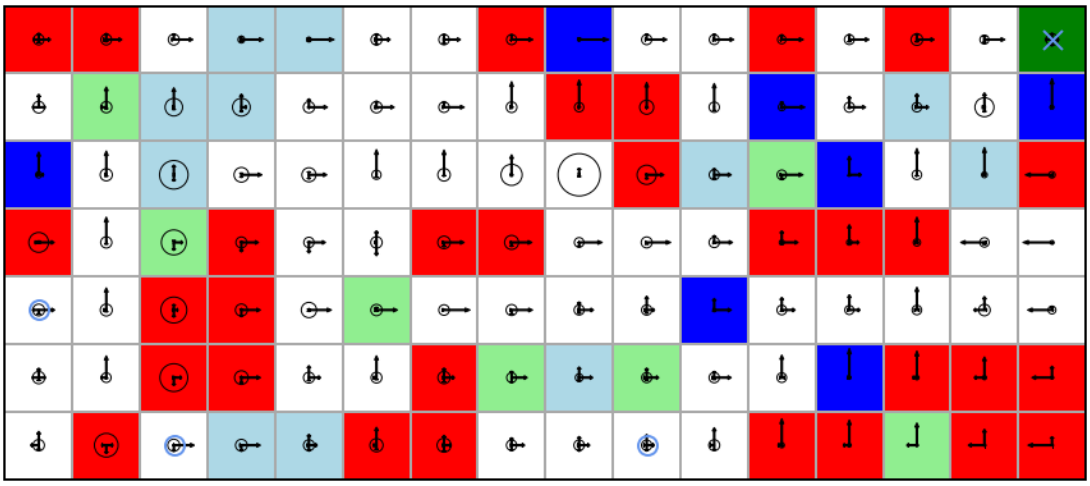

Figure 16: Classifier Policy on Novel Gridworld

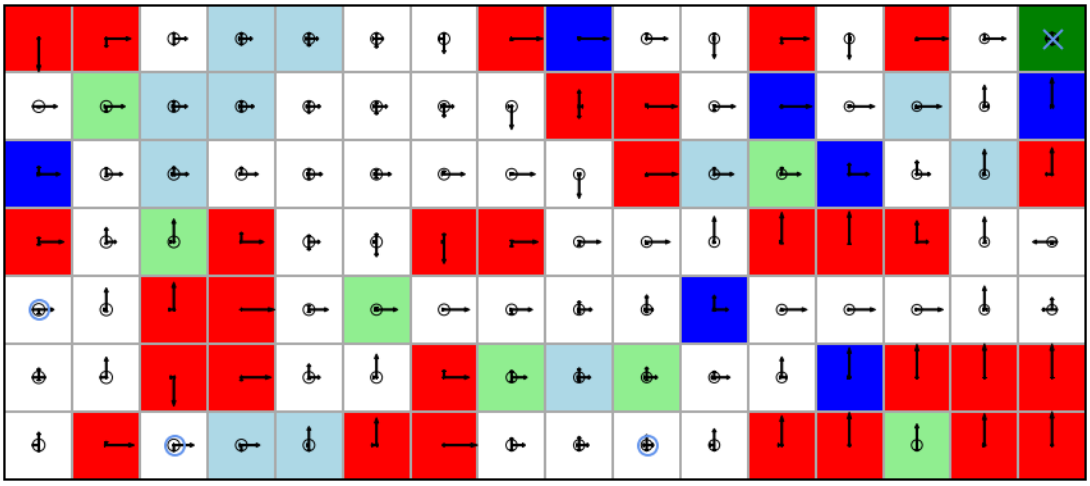

Figure 17: MLIRL Policy on Novel Gridworld

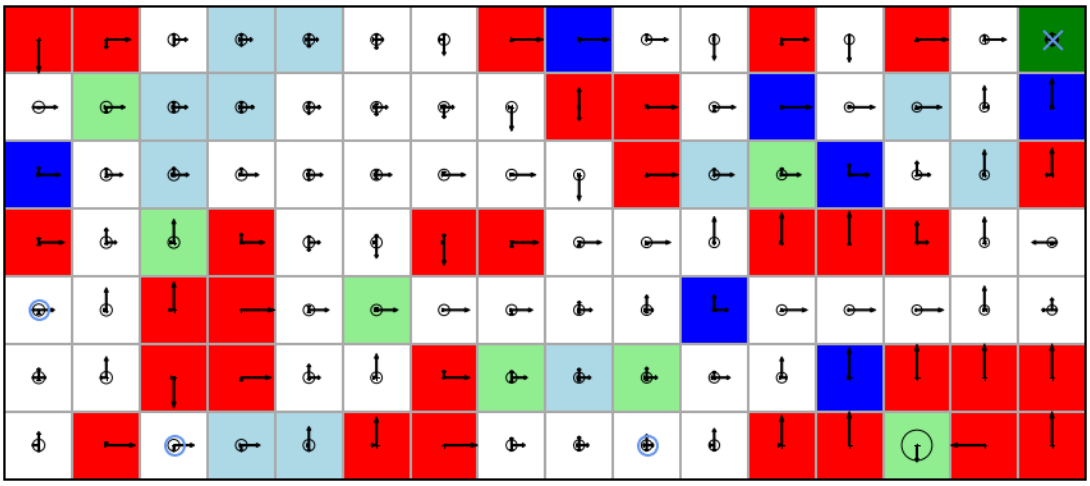

Figure 18: Expert Policy on Novel Gridworld

-4222.91	-3052.06	-2296.12	-2181.02	-2208.82	-2239.35	-2298.06	-1846.01	-1585.71	-1591.71	-1613.77	-1141.47	-911.73	-437.29	-34.08	0.00
-4281.56	-3657.48	-2383.39	-2137.39	-2199.89	-2234.86	-2282.82	-2318.55	-1687.65	-1736.82	-1532.41	-873.71	-693.67	-559.89	-116.30	0.15
-4117.64	-3631.14	-2588.44	-2083.05	-2054.40	-2161.98	-2267.18	-2274.15	-2000.74	-1488.85	-1145.09	-902.45	-672.79	-584.70	-169.22	-312.50
-3962.08	-3583.38	-3005.07	-1993.76	-1656.40	-1780.84	-2180.28	-1729.06	-1562.65	-1582.73	-1607.13	-1311.74	-992.10	-759.69	-1230.28	-1216.92
-3426.37	-3514.92	-3266.27	-1835.69	-1496.97	-1506.68	-1486.66	-1502.11	-1525.49	-1548.67	-1570.52	-1579.98	-1396.96	-1261.04	-1229.04	-1216.33
-3357.39	-3451.21	-2617.78	-1777.55	-1559.88	-1569.32	-1866.96	-1499.81	-1492.22	-1507.29	-1524.88	-1540.08	-1394.79	-1308.19	-1448.34	-1696.16
-3292.07	-3268.22	-1820.93	-1847.61	-1880.80	-1881.97	-2392.39	-1535.01	-1466.50	-1476.67	-1498.02	-1568.83	-1548.96	-1870.87	-1911.80	-2348.61

Figure 19: Classifier State Value Map on Novel Gridworld

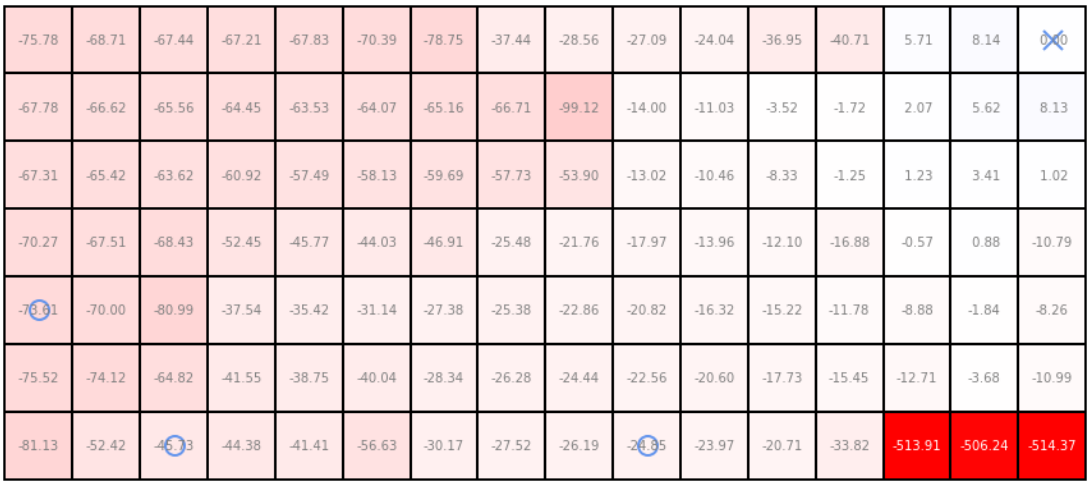

Figure 20: MLIRL State Value Map on Novel Gridworld

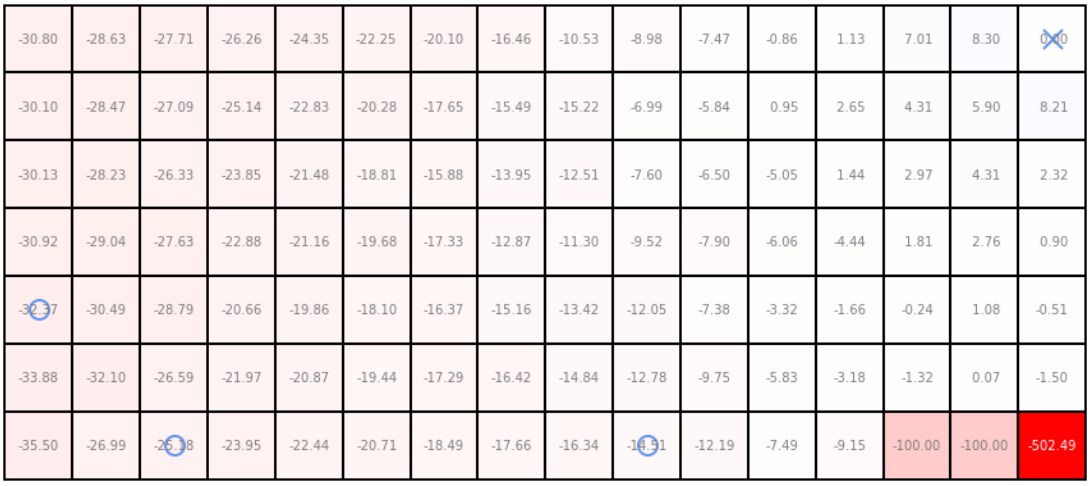

Figure 21: Expert State Value Map on Novel Gridworld

Trained on 100000 trajectories with batch size of 128 for 50 epochs.

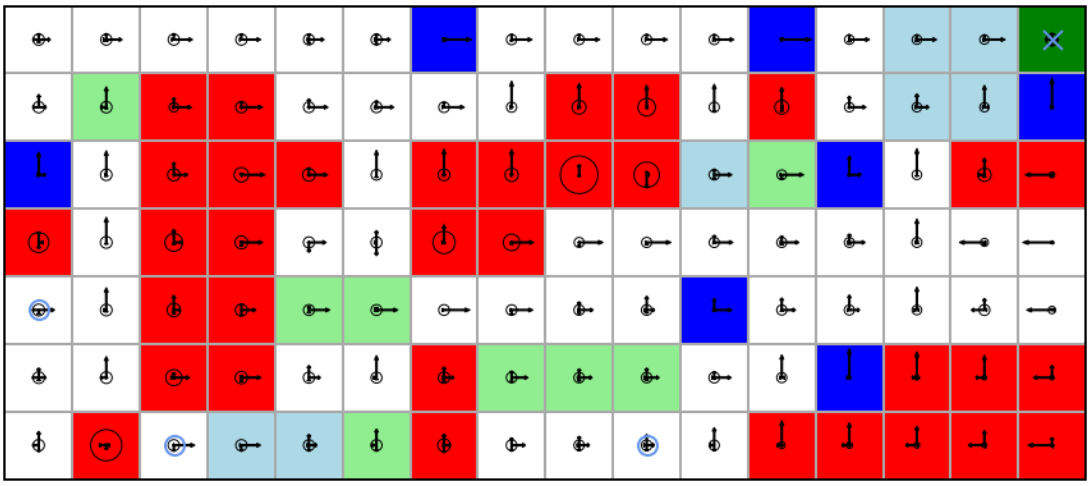

Figure 22: Classifier Policy

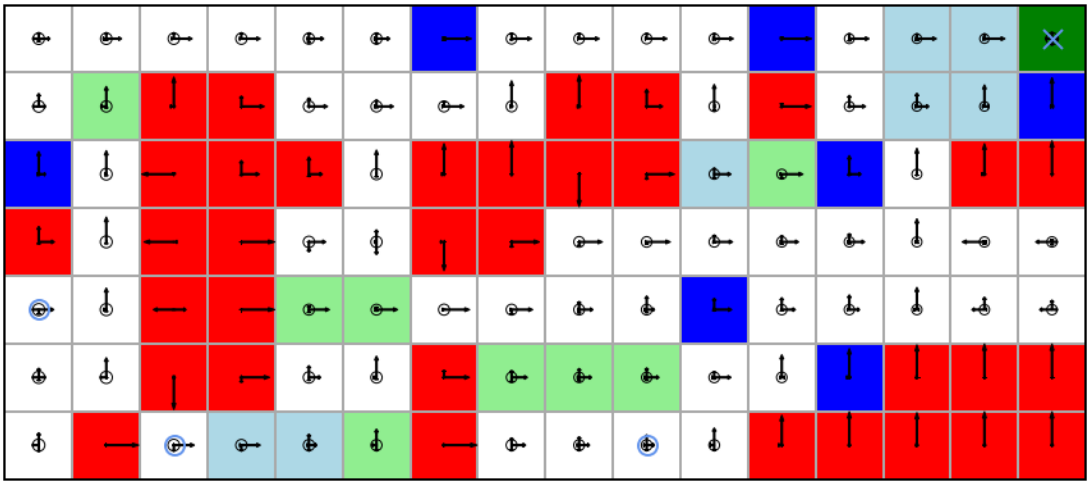

Figure 23: MLIRL Policy

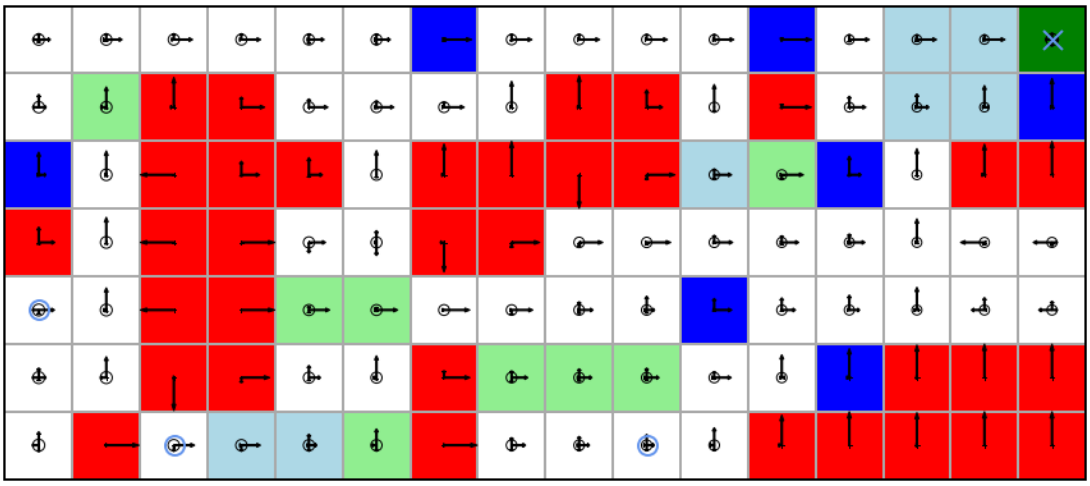

Figure 24: Expert Policy

-29.39	-27.04	-24.77	-22.80	-21.16	-19.17	-12.32	-10.29	-8.58	-6.80	-4.90	2.09	3.92	5.91	8.04	0.00
-30.56	-28.97	-502.85	-138.59	-21.26	-17.94	-15.21	-12.02	-163.71	-254.24	-7.22	-153.66	2.56	4.10	5.39	8.15
-32.39	-30.74	-1281.54	-807.22	-155.07	-21.74	-141.42	-174.12	-1384.59	-496.77	-7.88	-5.91	0.89	2.30	-156.64	-668.17
-307.24	-32.27	-1696.69	-357.22	-27.75	-23.07	-832.34	-210.86	-12.37	-9.60	-7.63	-5.50	-3.13	-0.08	-10.16	-11.61
-37.07	-34.55	-1881.43	-508.73	-23.30	-20.79	-18.89	-17.23	-14.47	-12.79	-8.56	-6.40	-4.39	-2.47	-7.61	-9.09
-40.32	-39.45	-1128.44	-274.14	-24.76	-22.74	-359.88	-21.74	-16.47	-14.16	-11.87	-8.85	-6.50	-50.53	-202.54	-465.23
-45.67	-903.91	-30.17	-28.07	-26.41	-25.15	-906.26	-28.08	-18.89	-16.28	-14.56	-75.58	-212.42	-662.23	-982.34	-1381.69

Figure 25: Classifier State Value Map

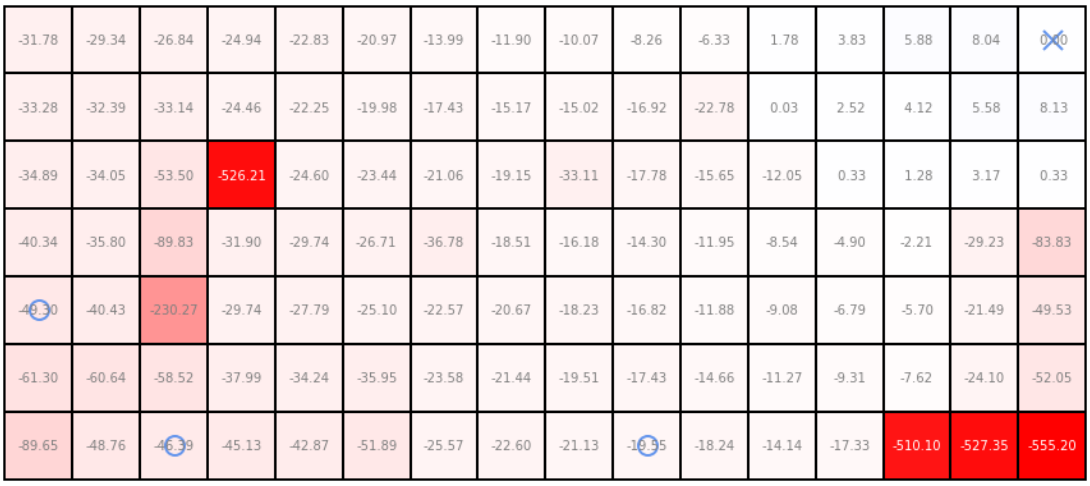

Figure 26: MLIRL State Value Map

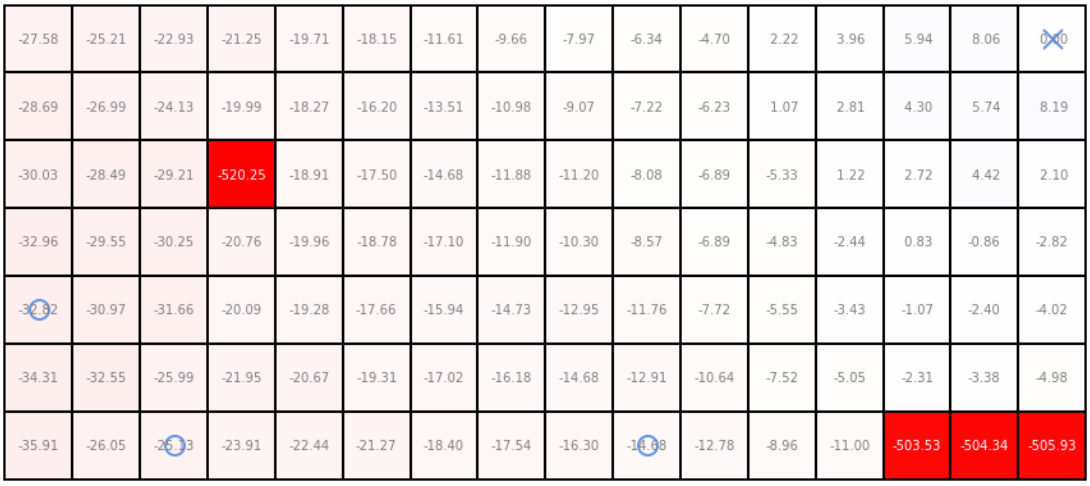

Figure 27: Expert State Value Map

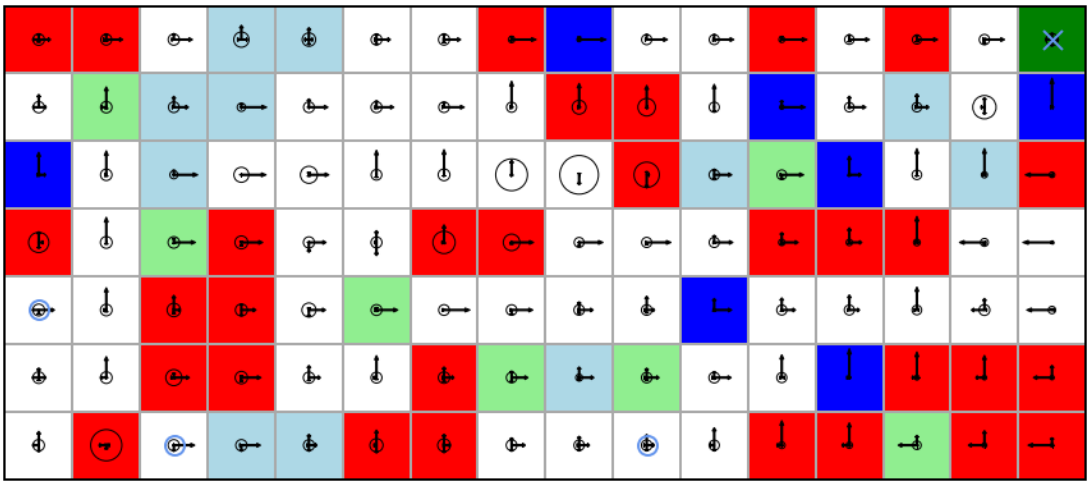

Figure 28: Classifier Policy on Novel Gridworld

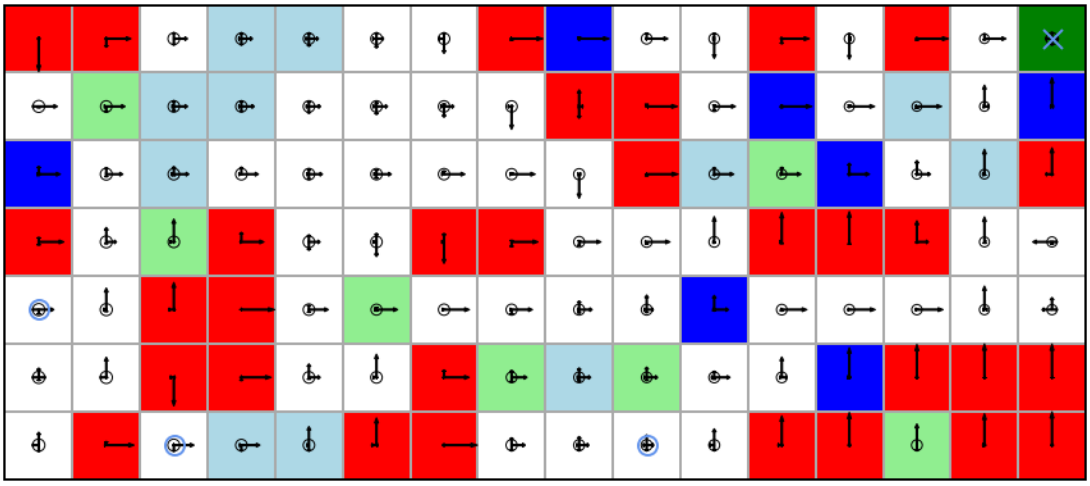

Figure 29: MLIRL Policy on Novel Gridworld

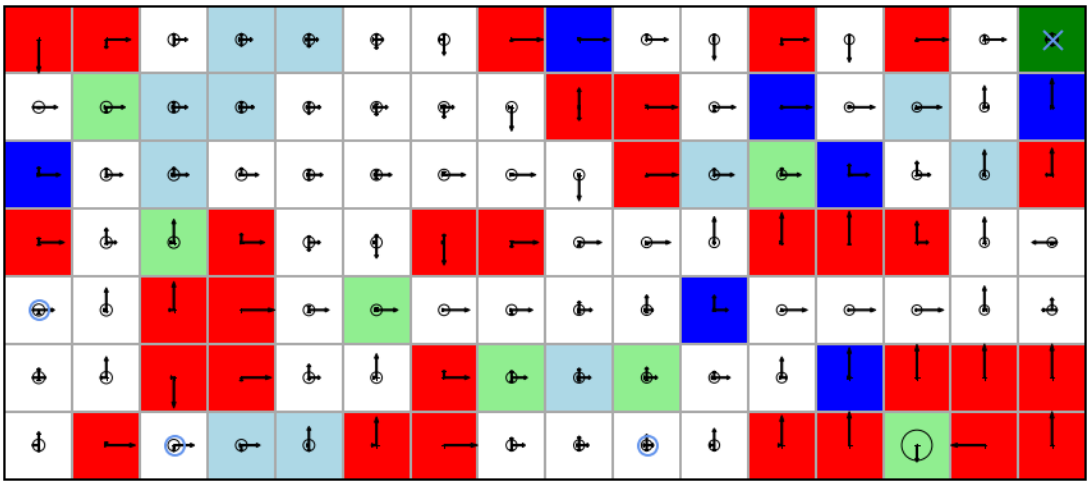

Figure 30: Expert Policy on Novel Gridworld

-3911.06	-2714.02	-2004.50	-1885.27	-1902.38	-2007.62	-2070.16	-1612.09	-1448.13	-1436.18	-1456.37	-981.61	-843.92	-363.40	-92.82	0.00
-3940.77	-3322.27	-2023.23	-1923.32	-1952.87	-2008.06	-2055.67	-2089.13	-1630.21	-1663.87	-1377.97	-851.89	-677.70	-578.27	-254.78	-5.65
-3748.73	-3303.55	-1889.33	-1883.06	-1905.69	-1952.54	-2062.01	-2051.96	-1655.48	-1975.06	-1098.15	-887.73	-672.08	-605.26	-312.06	-445.96
-3681.25	-3256.88	-2315.82	-1880.13	-1622.19	-1699.39	-2260.58	-1769.85	-1600.94	-1627.06	-1652.52	-1393.14	-1058.11	-759.60	-1235.75	-1222.55
-3110.55	-3193.50	-2654.59	-2023.88	-1536.26	-1526.00	-1519.99	-1537.65	-1566.97	-1594.15	-1620.25	-1633.48	-1442.01	-1266.70	-1235.22	-1220.49
-3045.03	-3124.51	-2752.56	-1877.43	-1599.78	-1599.96	-1847.51	-1513.40	-1535.17	-1553.46	-1573.72	-1590.98	-1441.55	-1328.29	-1430.84	-1673.63
-2977.99	-3483.98	-1907.29	-1935.24	-1970.25	-2058.61	-2425.71	-1490.67	-1501.19	-1509.56	-1541.24	-1630.51	-1661.09	-2027.83	-2006.71	-2438.62

Figure 31: Classifier State Value Map on Novel Gridworld

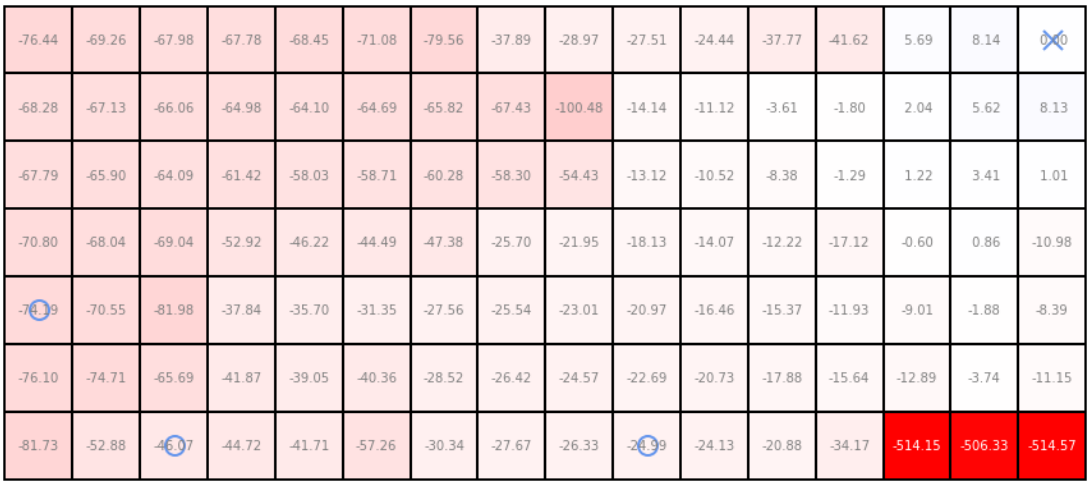

Figure 32: MLIRL State Value Map on Novel Gridworld

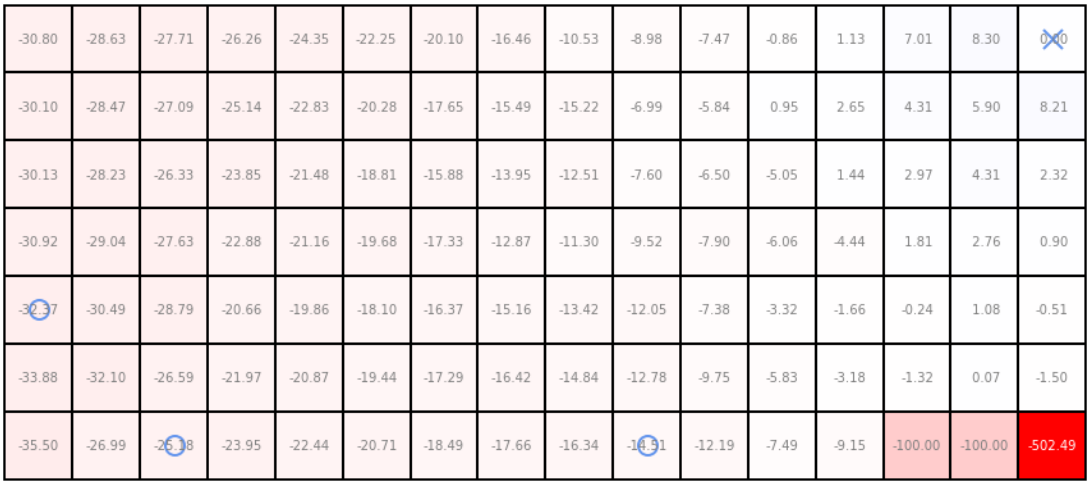

Figure 33: Expert State Value Map on Novel Gridworld

10.3 Classifier and MLIRL Results on Gridworld 2

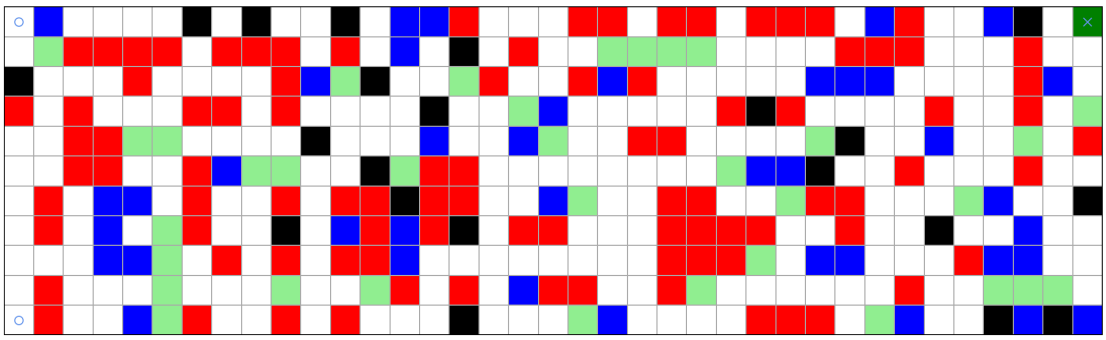

Figure 34: Gridworld 2

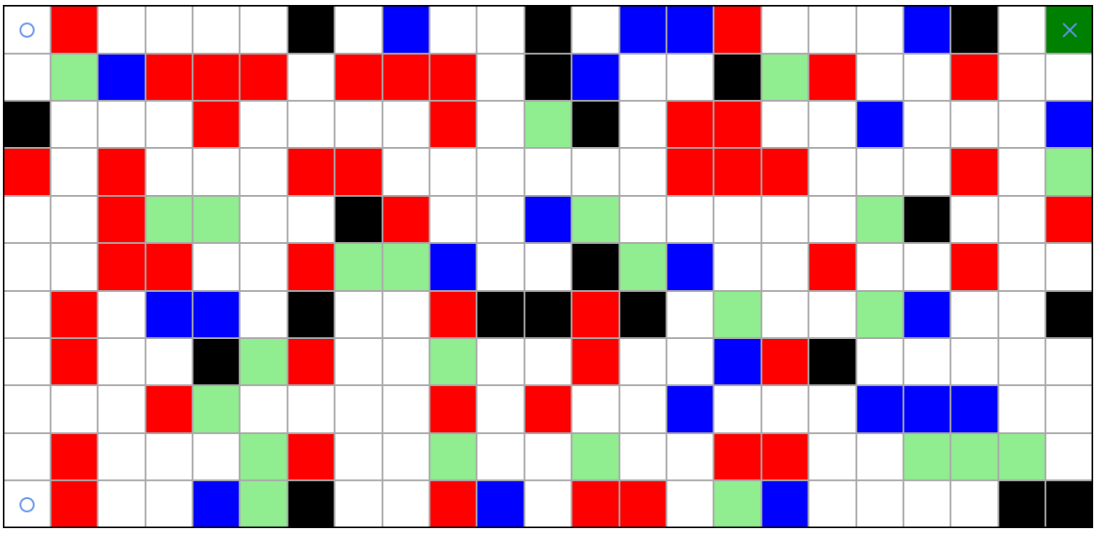

Figure 35: Novel Gridworld for Gridworld 2

Trained on 10000 trajectories with batch size of 128 for 50 epochs.

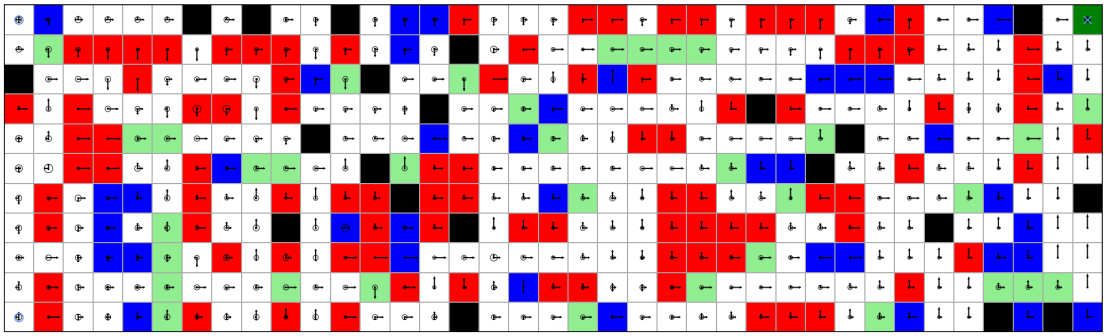

Figure 36: Classifier Policy

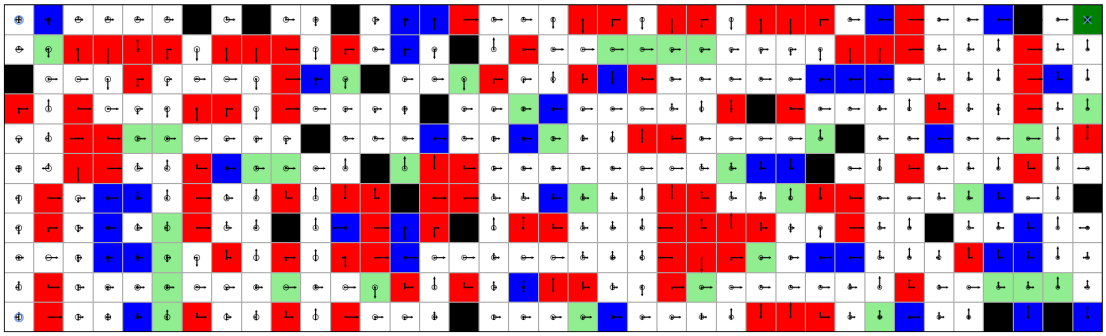

Figure 37: MLIRL Policy

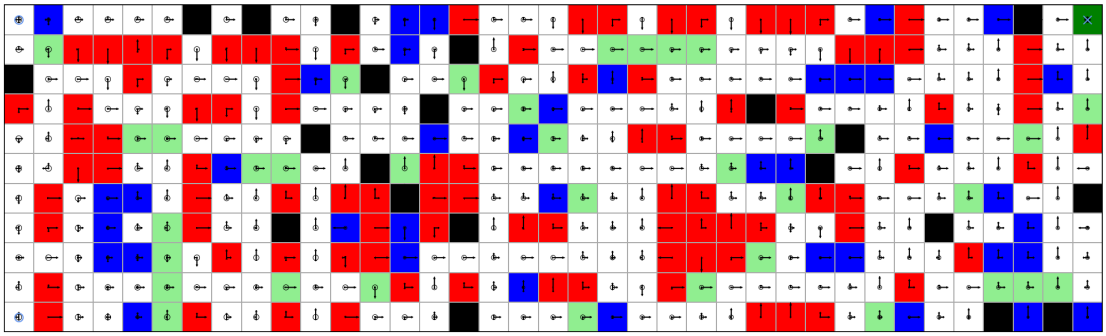

Figure 38: Expert Policy

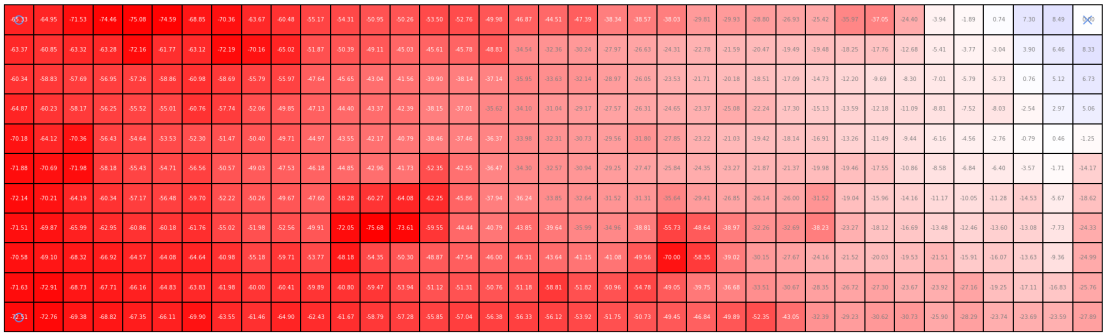

Figure 39: Classifier State Value Map

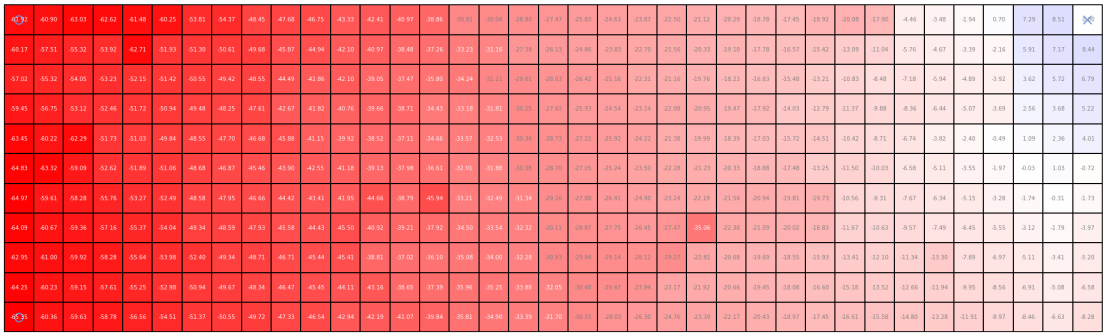

Figure 40: MLIRL State Value Map

Figure 41: Expert State Value Map

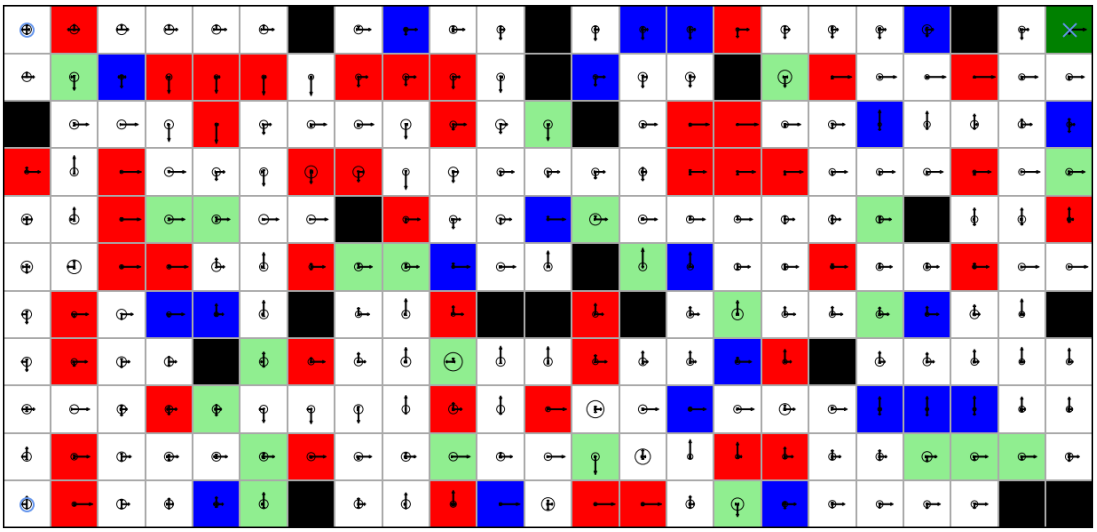

Figure 42: Classifier Policy on Novel Gridworld

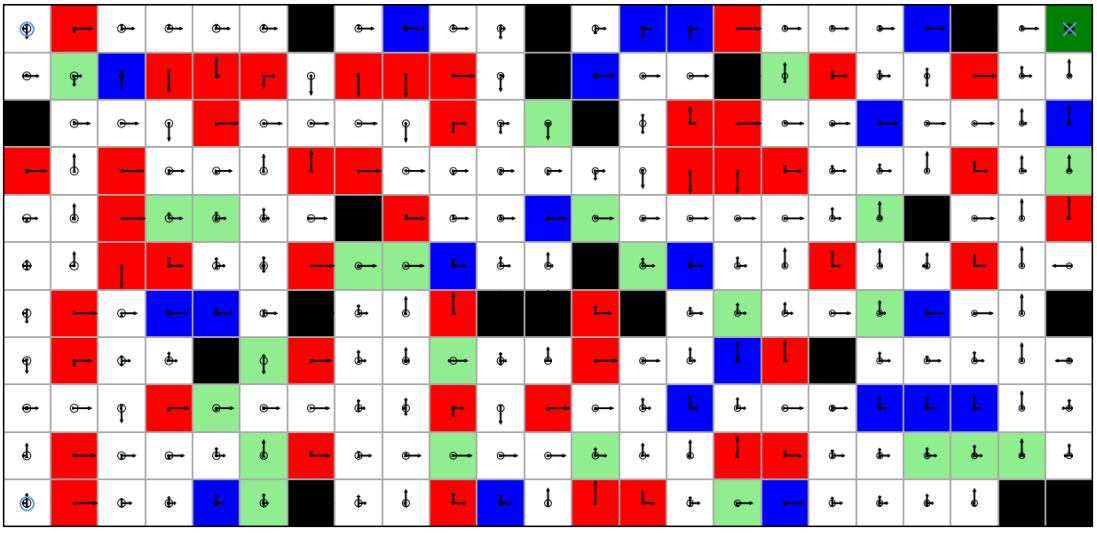

Figure 43: MLIRL Policy on Novel Gridworld

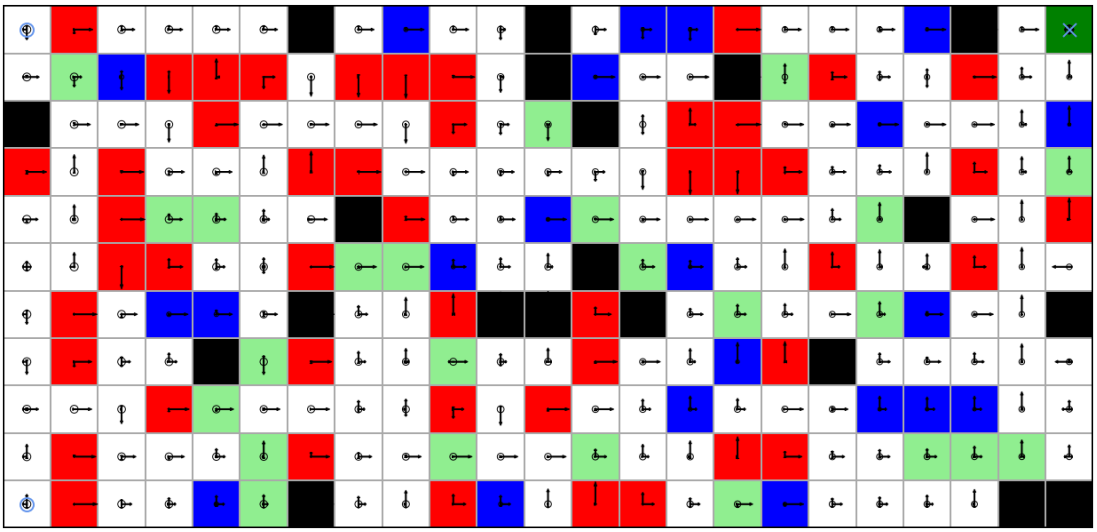

Figure 44: Expert Policy on Novel Gridworld

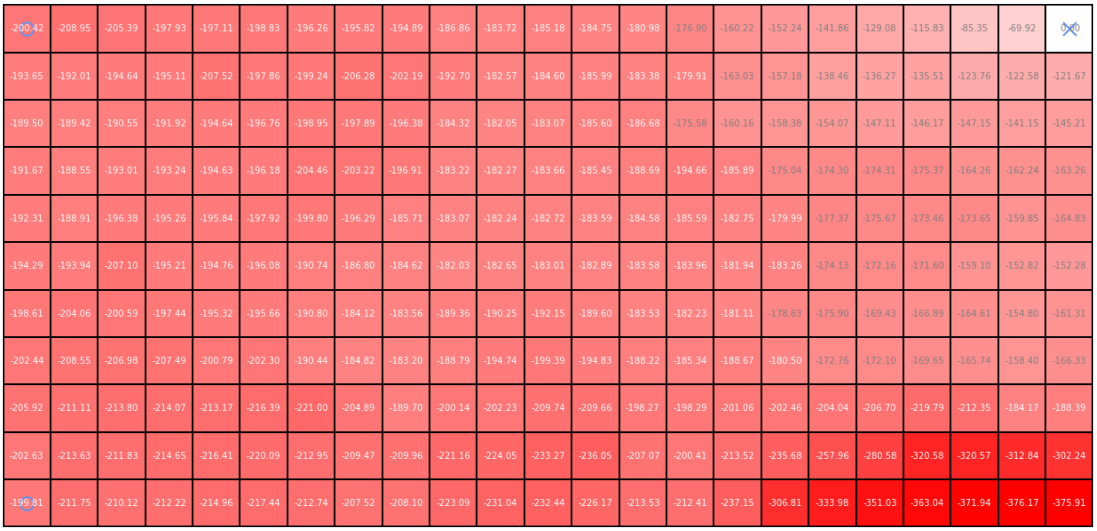

Figure 45: Classifier State Value Map on Novel Gridworld

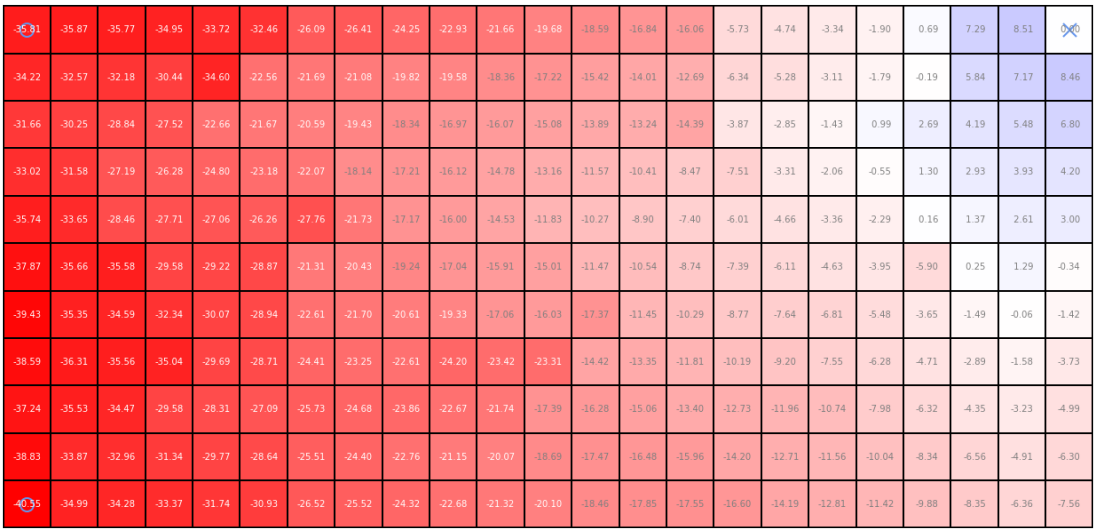

Figure 46: MLIRL State Value Map on Novel Gridworld

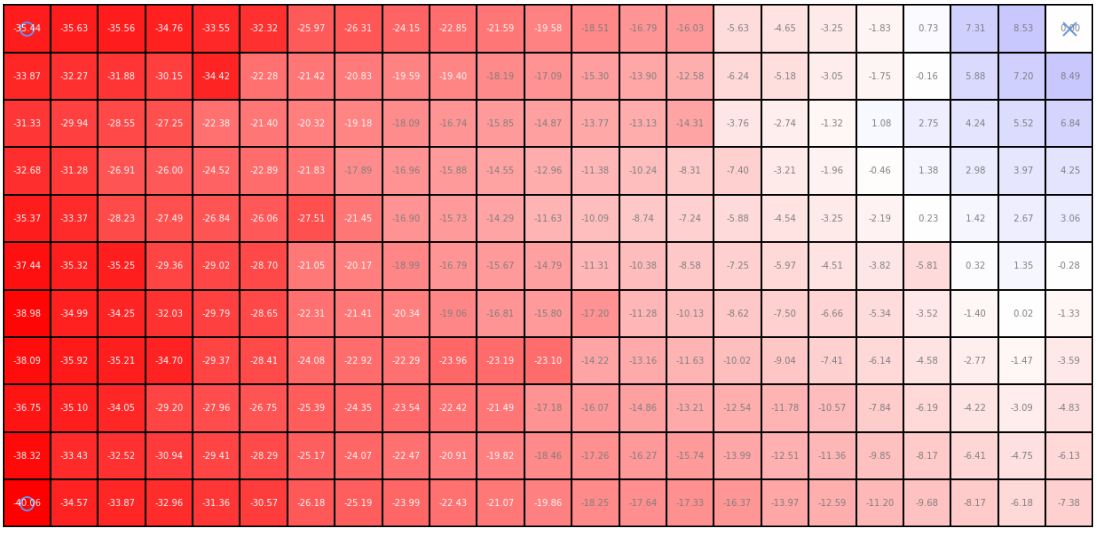

Figure 47: Expert State Value Map on Novel Gridworld

Trained on 100000 trajectories with batch size of 128 for 50 epochs.

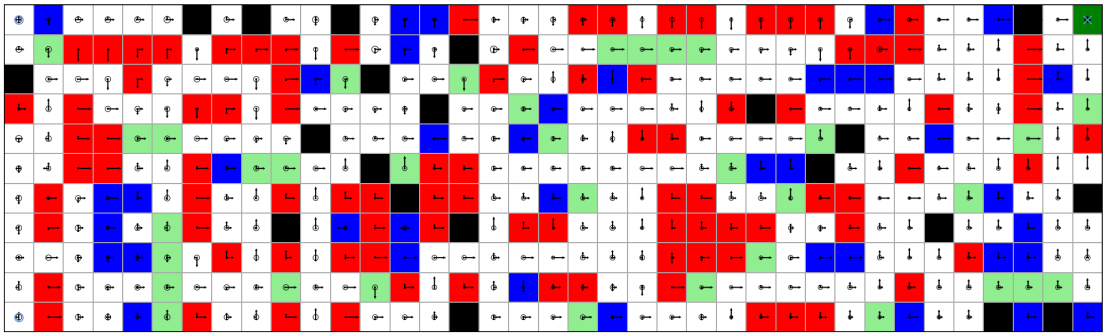

Figure 48: Classifier Policy

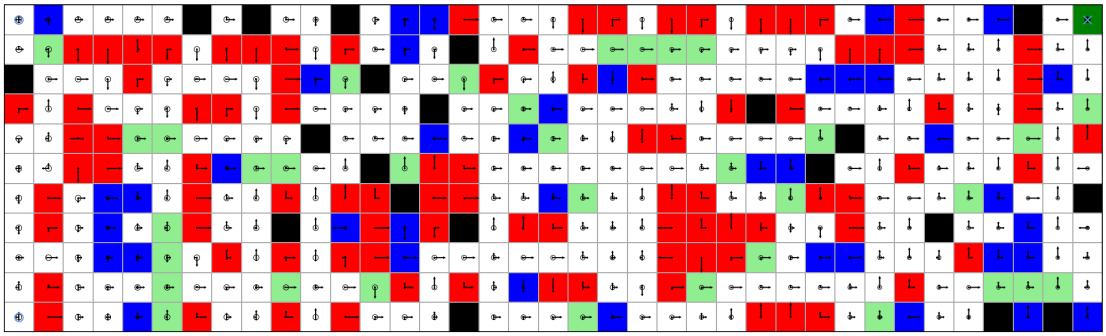

Figure 49: MLIRL Policy

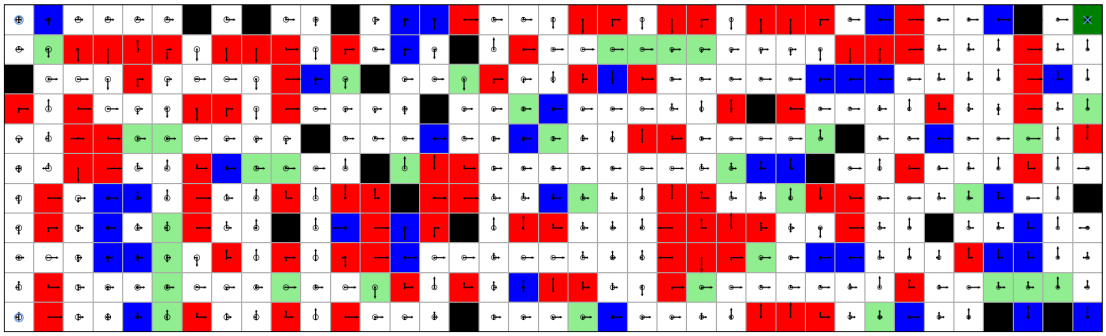

Figure 50: Expert Policy

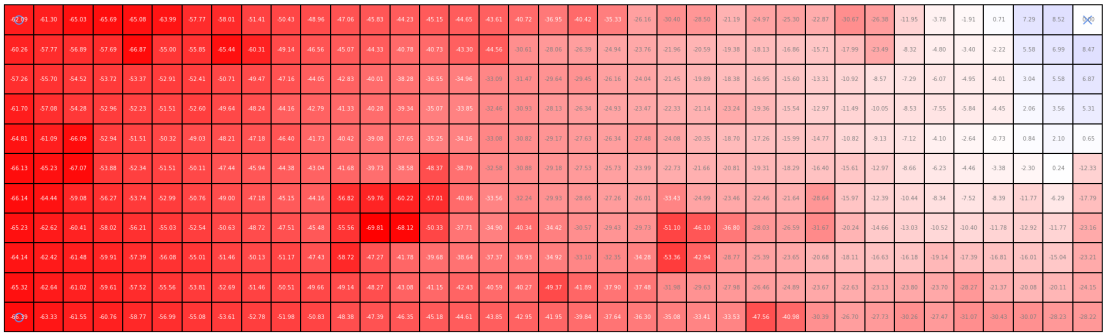

Figure 51: Classifier State Value Map

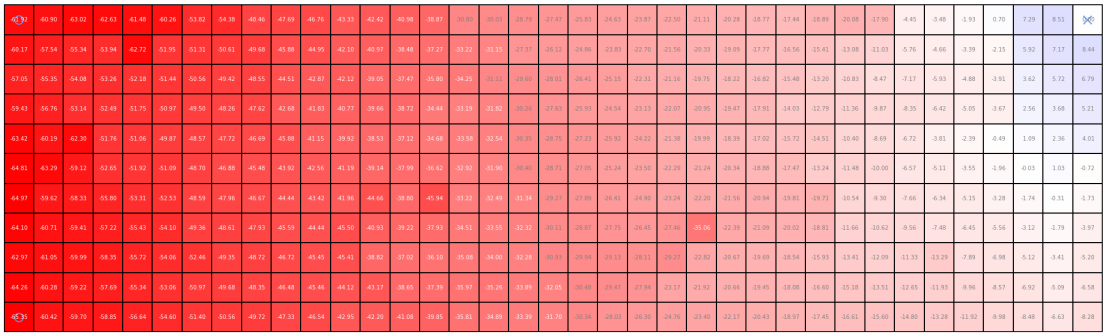

Figure 52: MLIRL State Value Map

Figure 53: Expert State Value Map

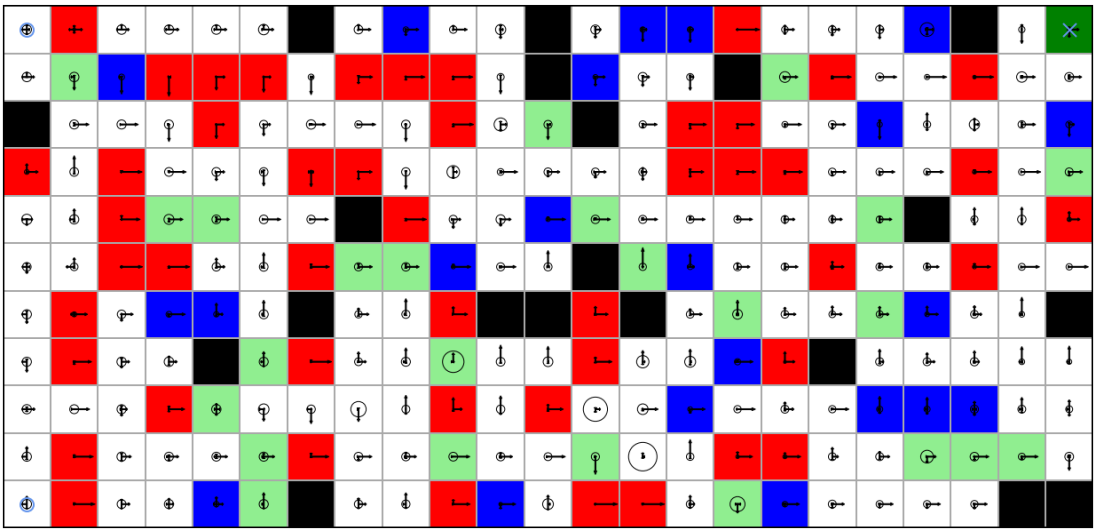

Figure 54: Classifier Policy on Novel Gridworld

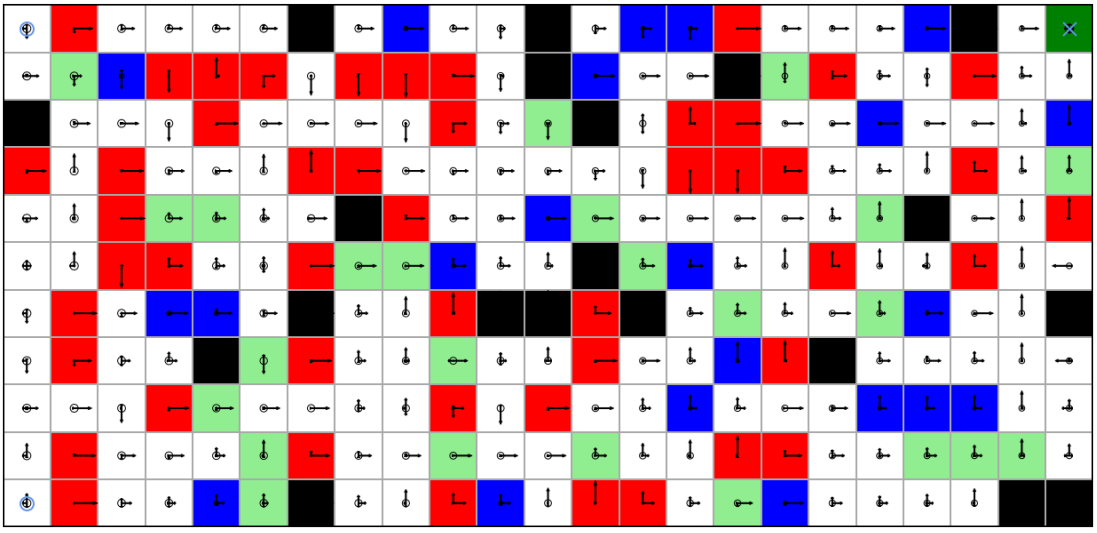

Figure 55: MLIRL Policy on Novel Gridworld

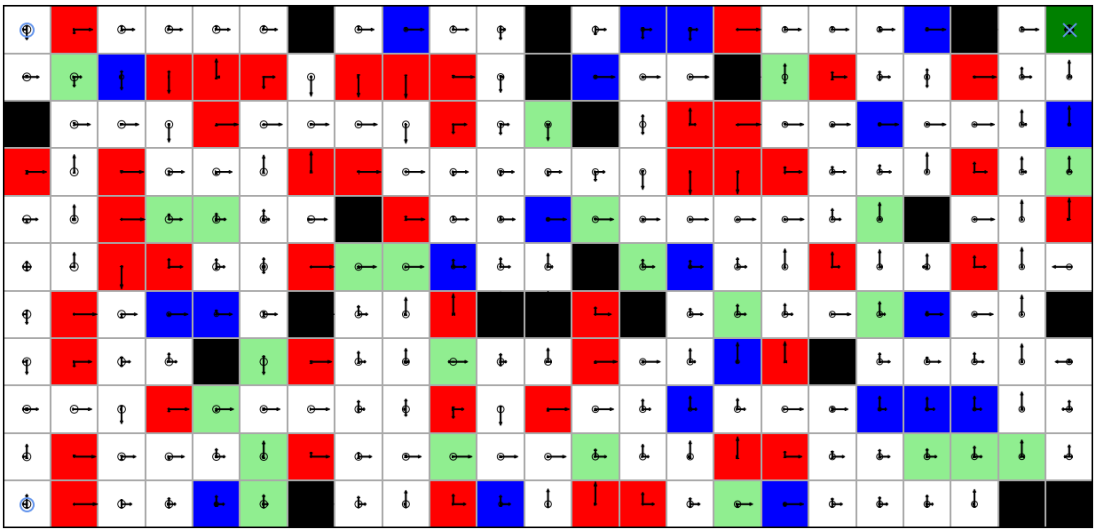

Figure 56: Expert Policy on Novel Gridworld

Figure 57: Classifier State Value Map on Novel Gridworld

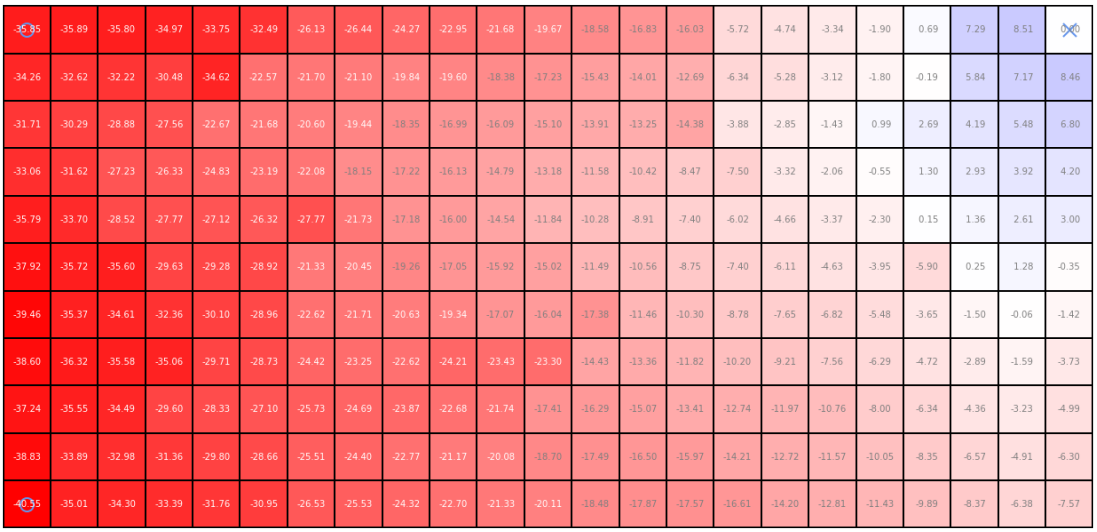

Figure 58: MLIRL State Value Map on Novel Gridworld

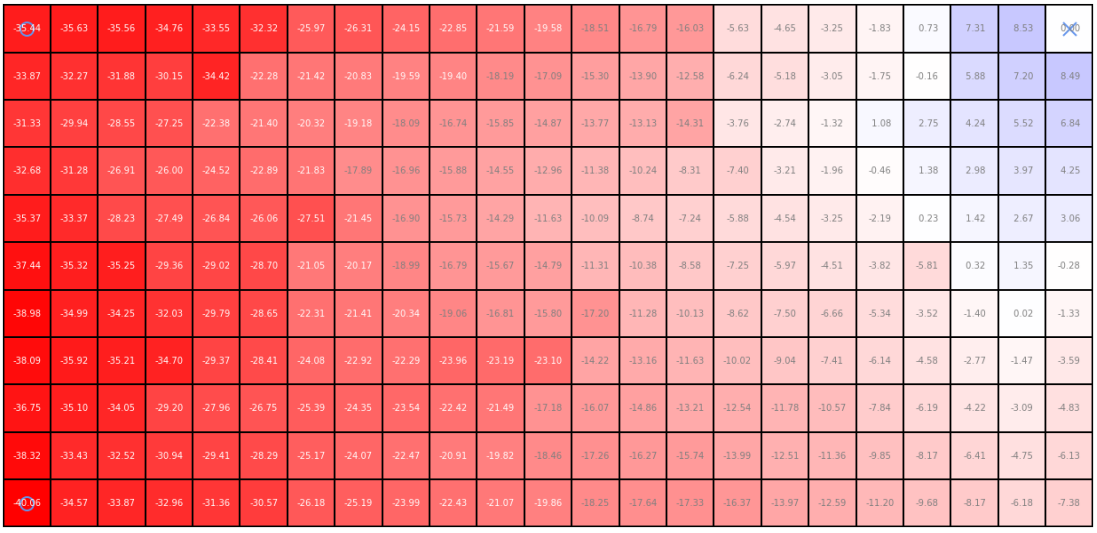

Figure 59: Expert State Value Map on Novel Gridworld

10.4 Classifier and MLIRL Results on Gridworld 3

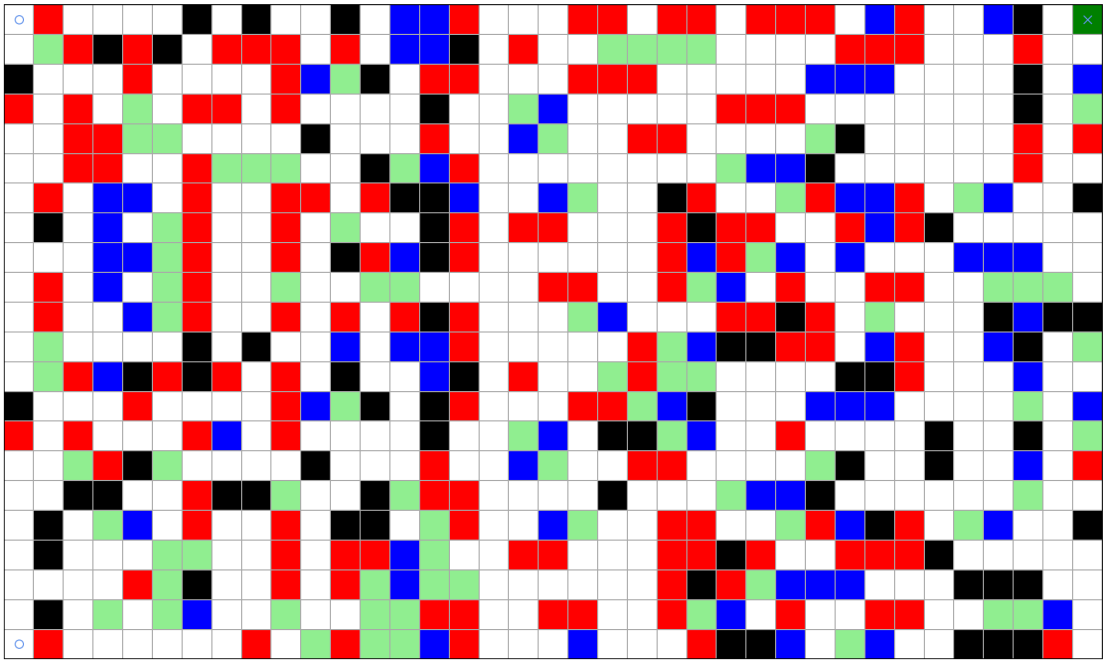

Figure 60: Gridworld 3

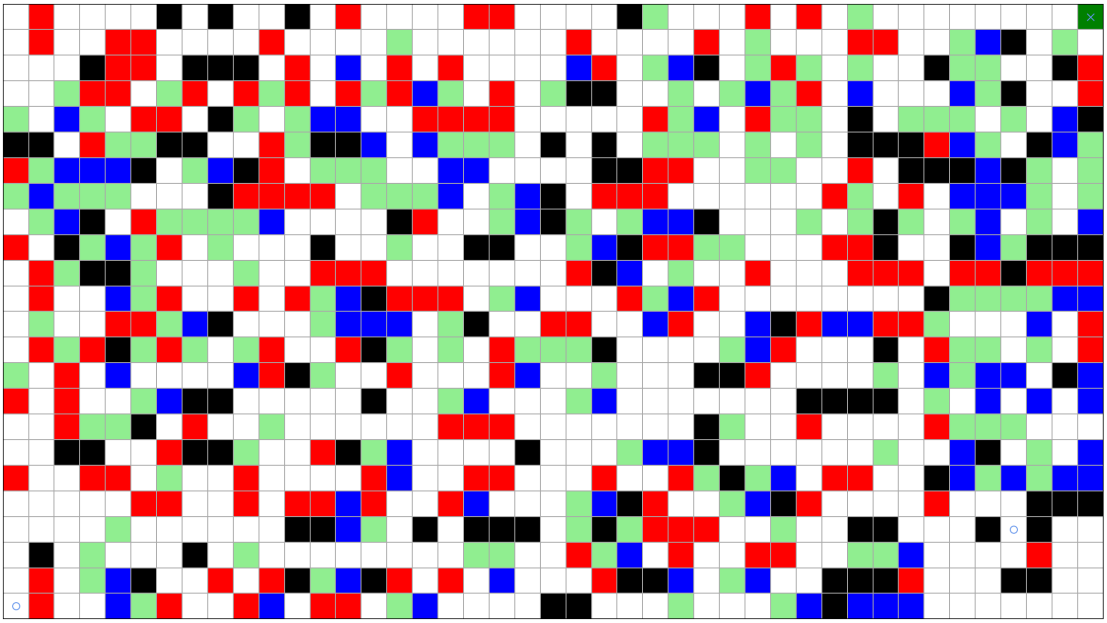

Figure 61: Novel Gridworld for Gridworld 3

Trained on 10000 trajectories with batch size of 128 for 50 epochs.

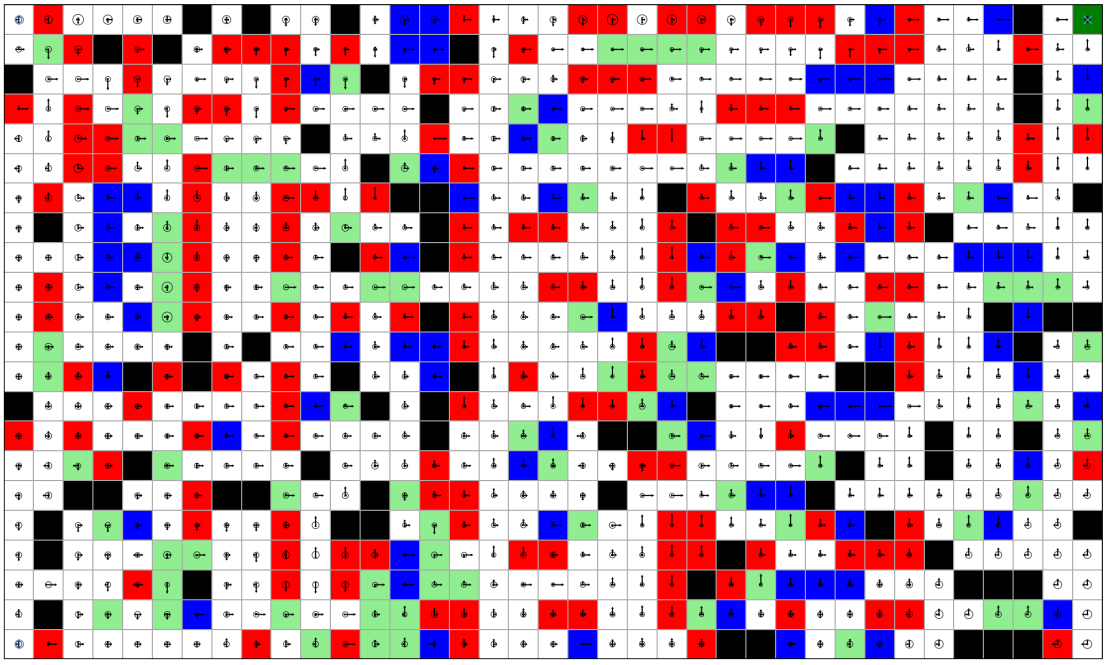

Figure 62: Classifier Policy

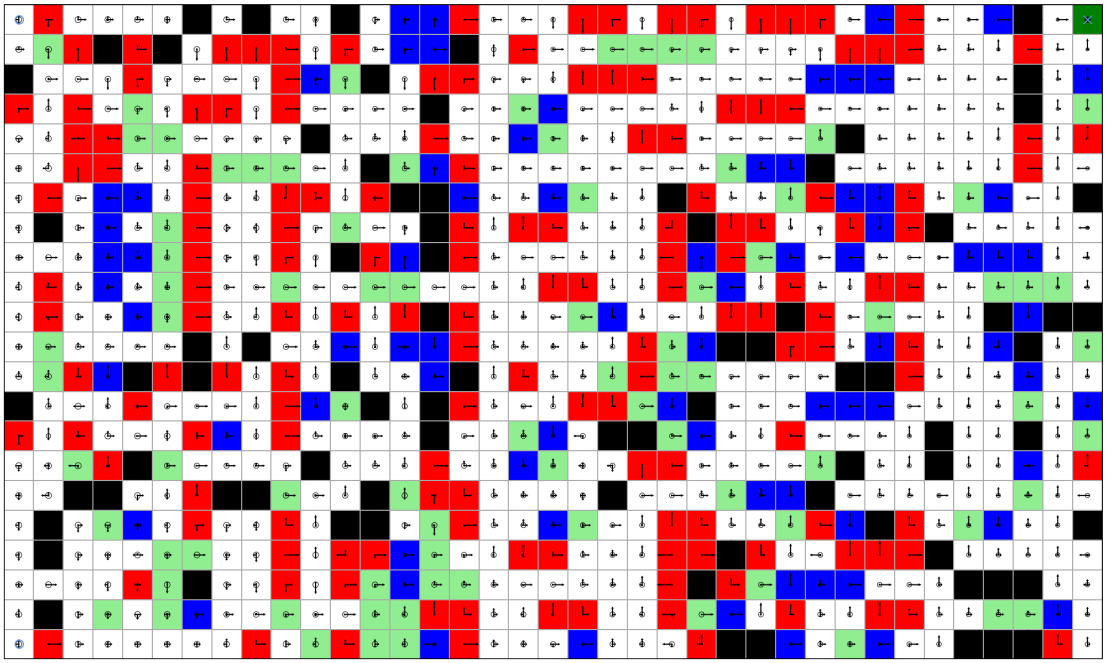

Figure 63: MLIRL Policy

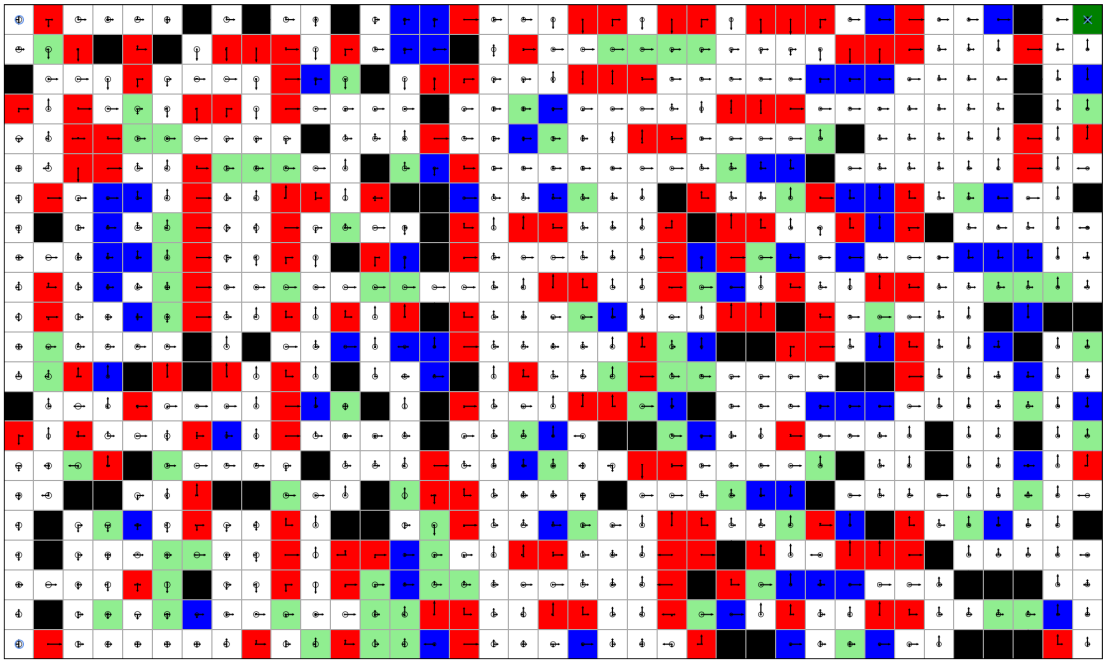

Figure 64: Expert Policy

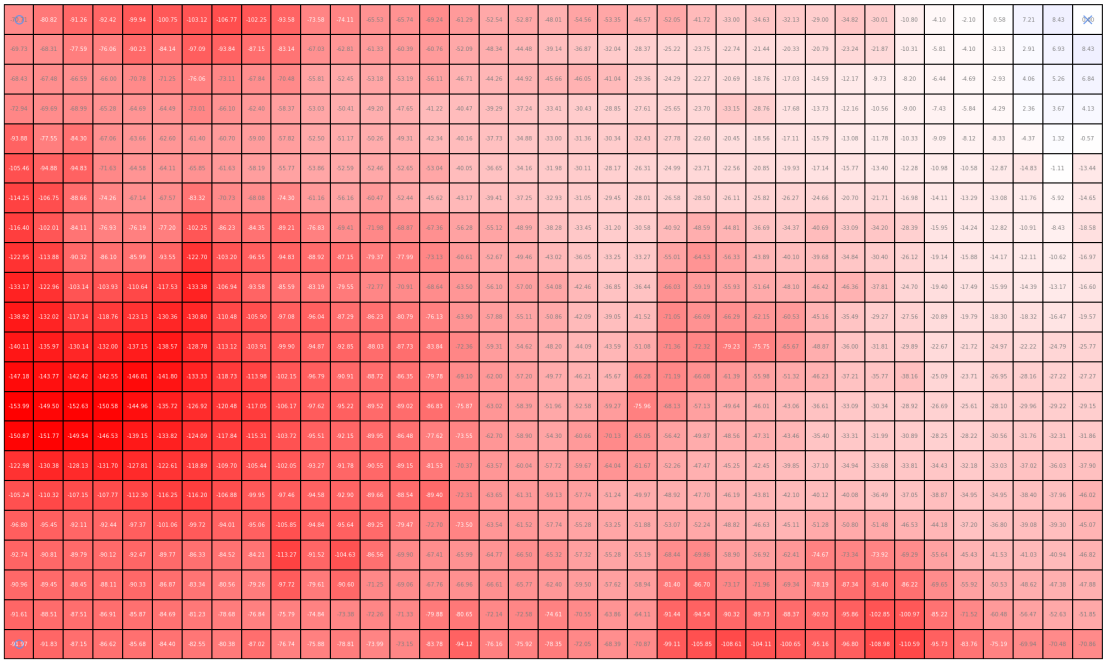

Figure 65: Classifier State Value Map

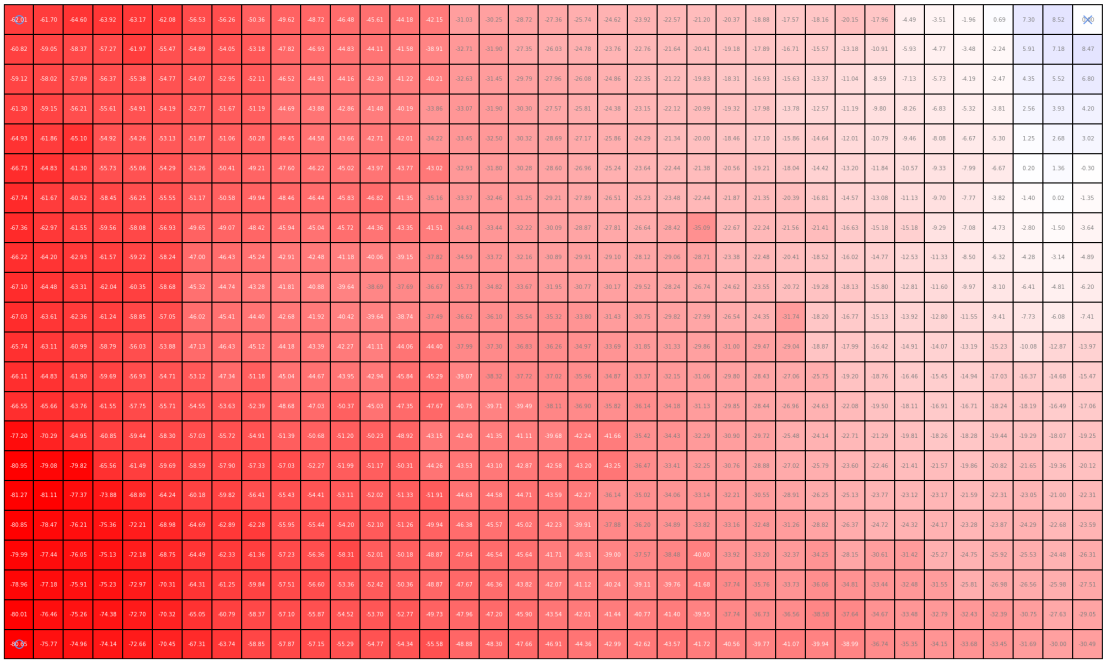

Figure 66: MLIRL State Value Map

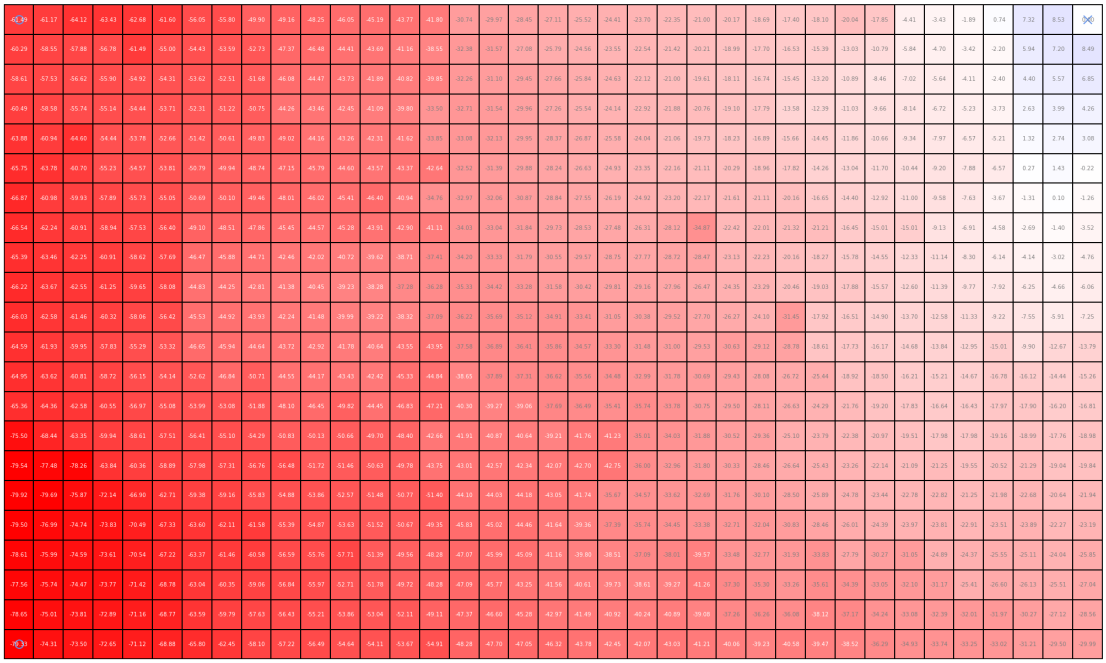

Figure 67: Expert State Value Map

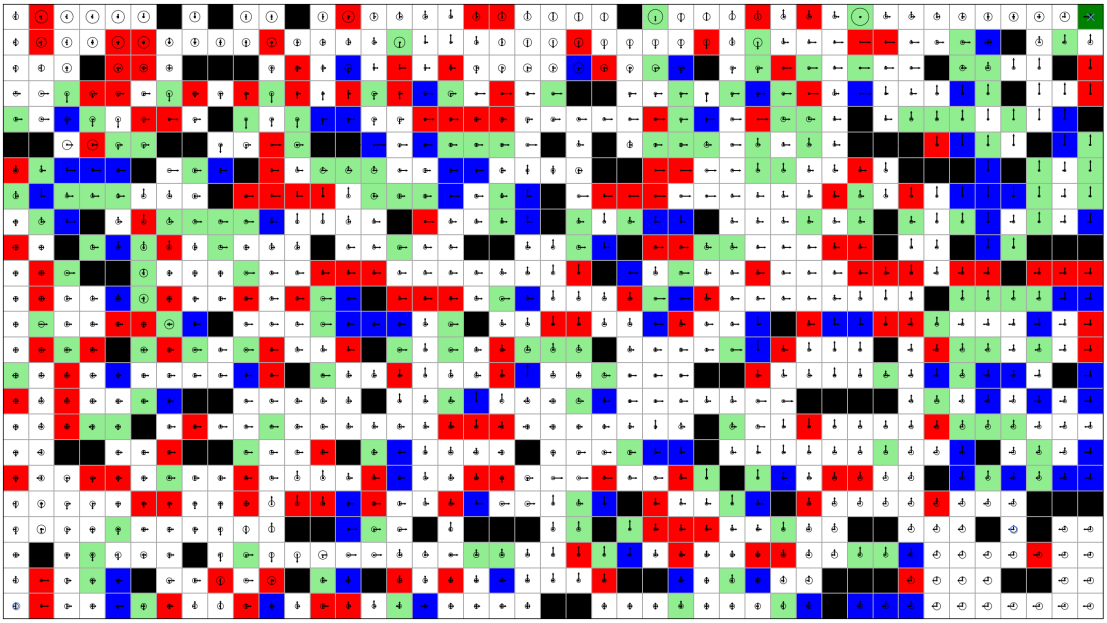

Figure 68: Classifier Policy on Novel Gridworld

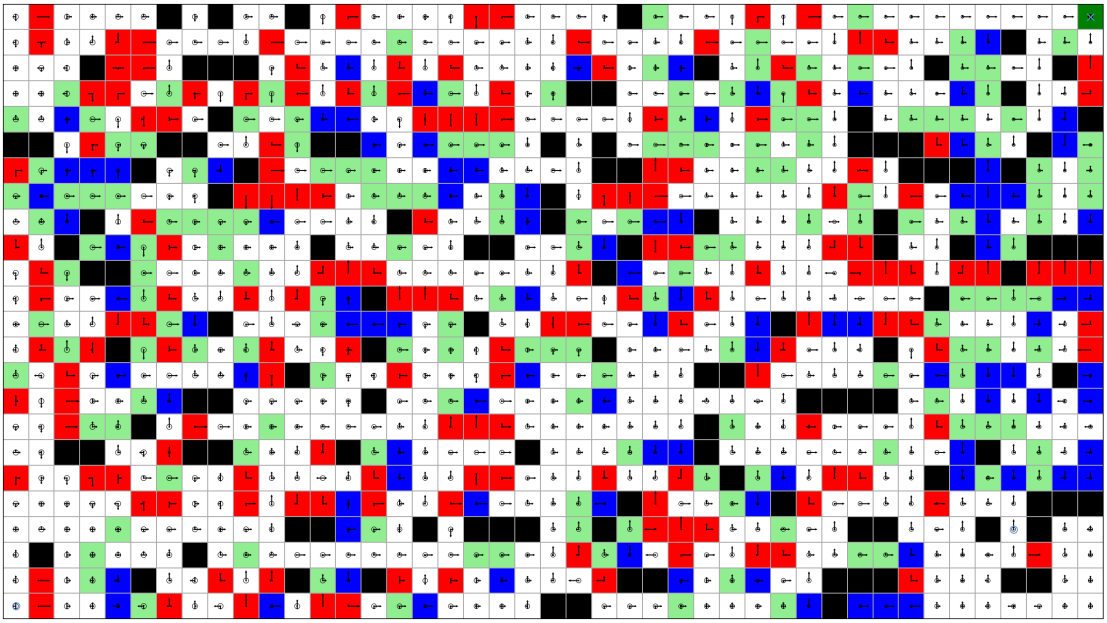

Figure 69: MLIRL Policy on Novel Gridworld

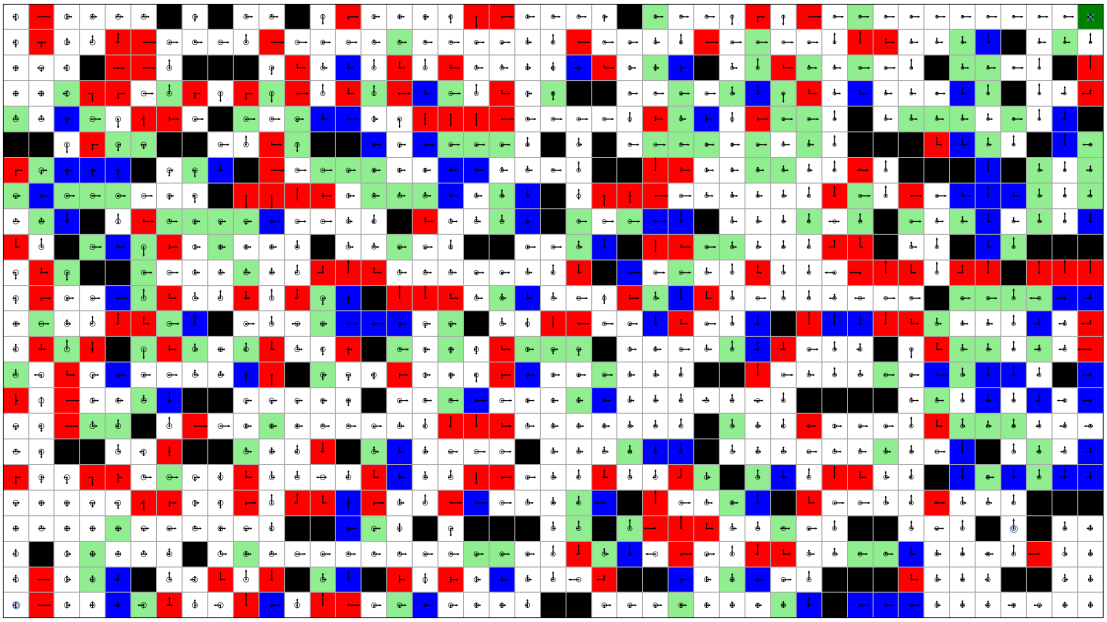

Figure 70: Expert Policy on Novel Gridworld

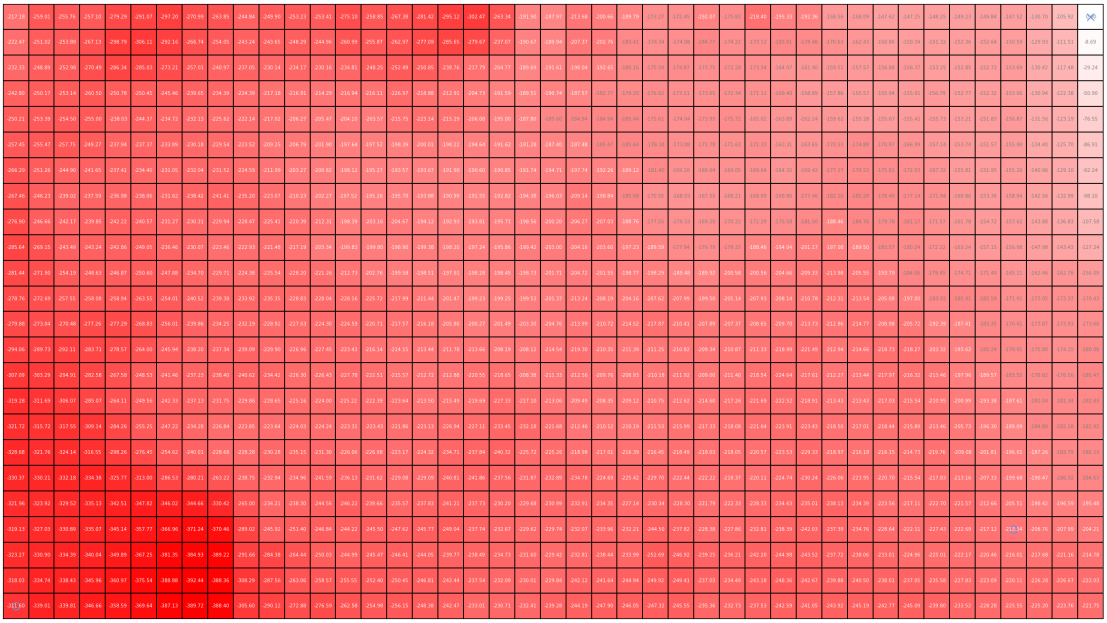

Figure 71: Classifier State Value Map on Novel Gridworld

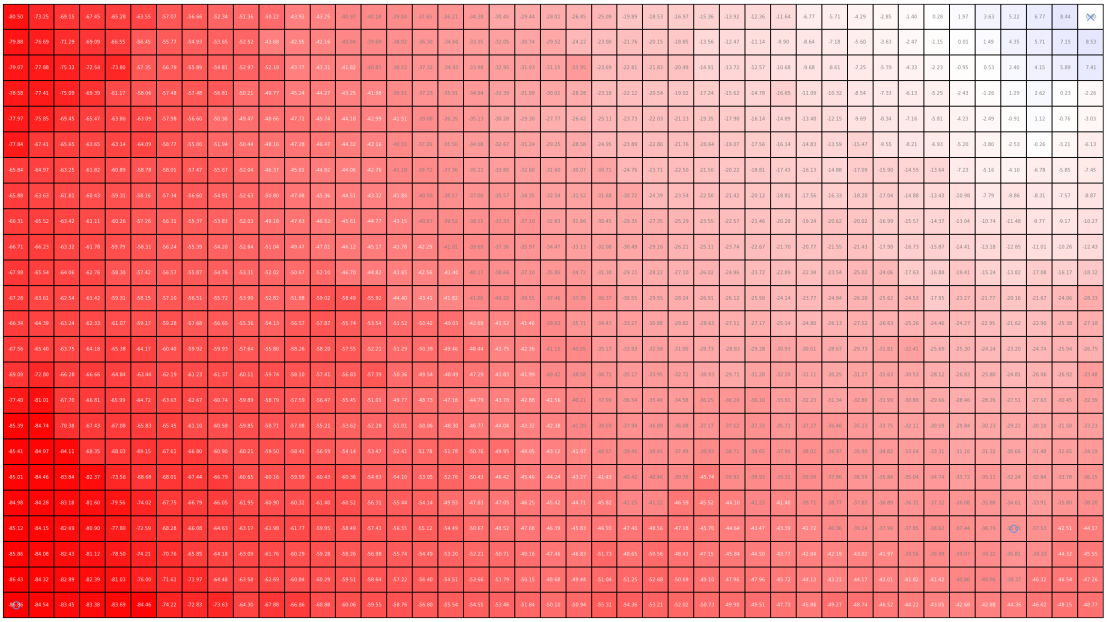

Figure 72: MLIRL State Value Map on Novel Gridworld

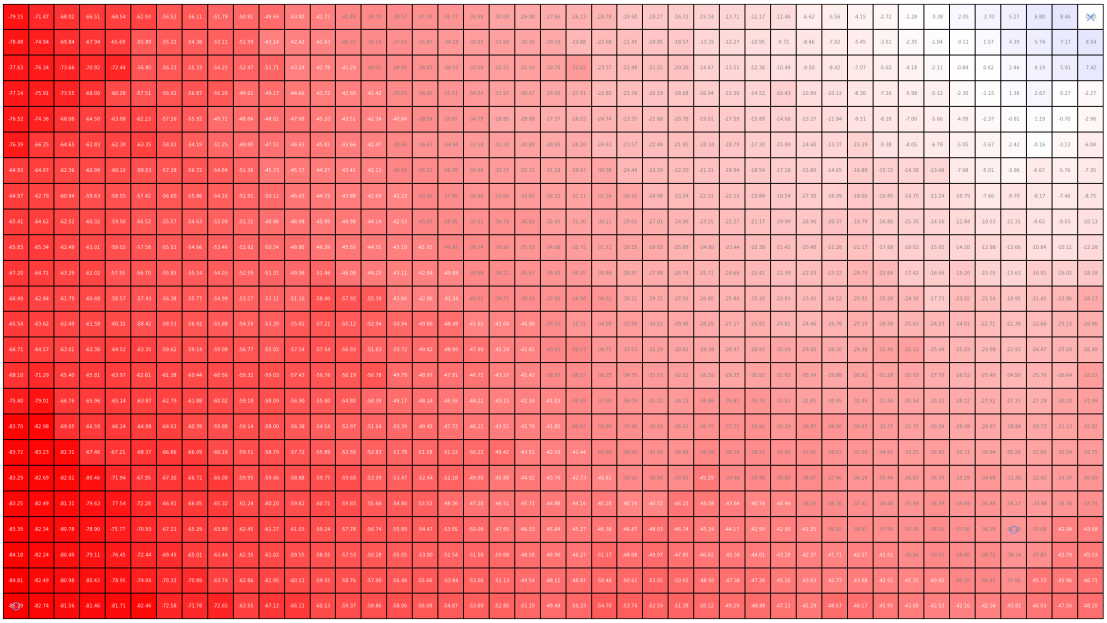

Figure 73: Expert State Value Map on Novel Gridworld

Trained on 100000 trajectories with batch size of 128 for 50 epochs.

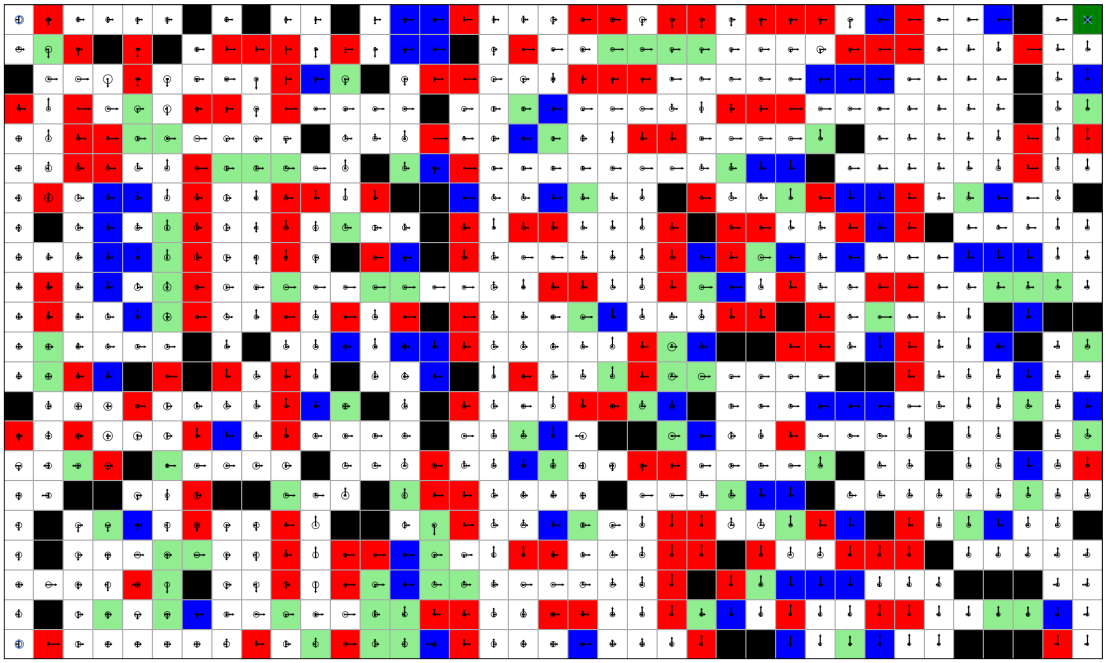

Figure 74: Classifier Policy

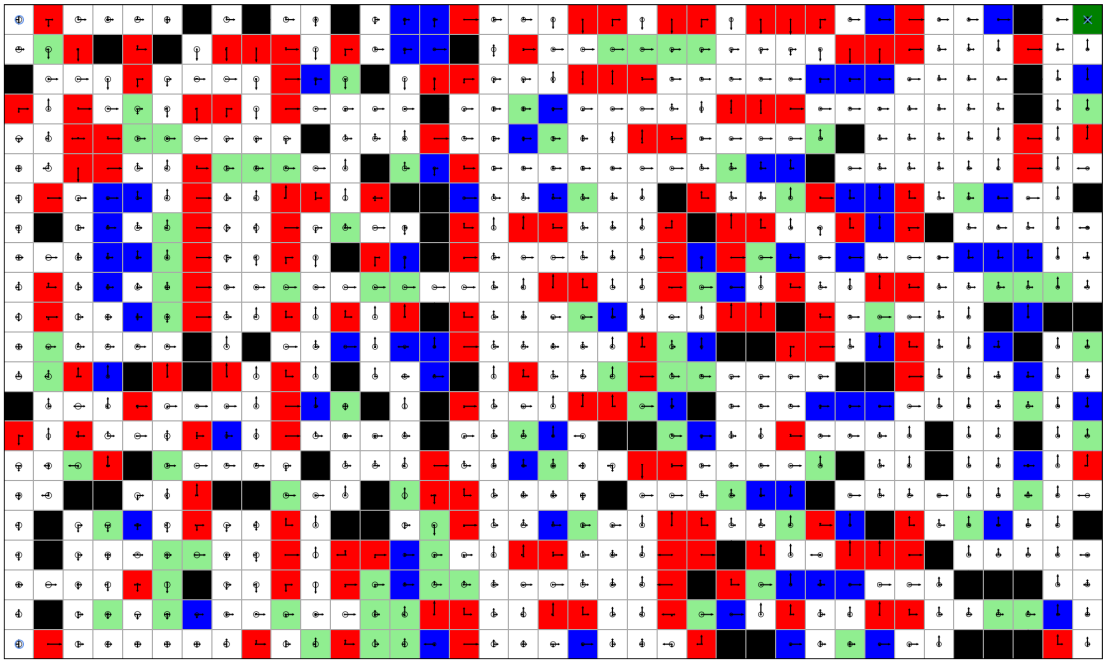

Figure 75: MLIRL Policy

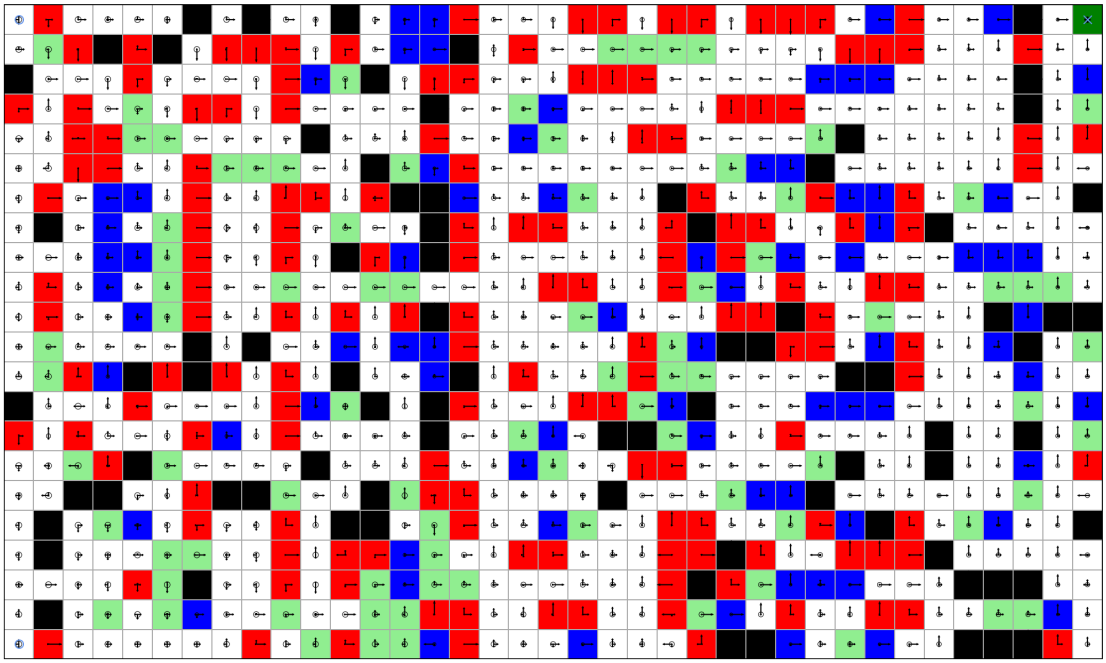

Figure 76: Expert Policy

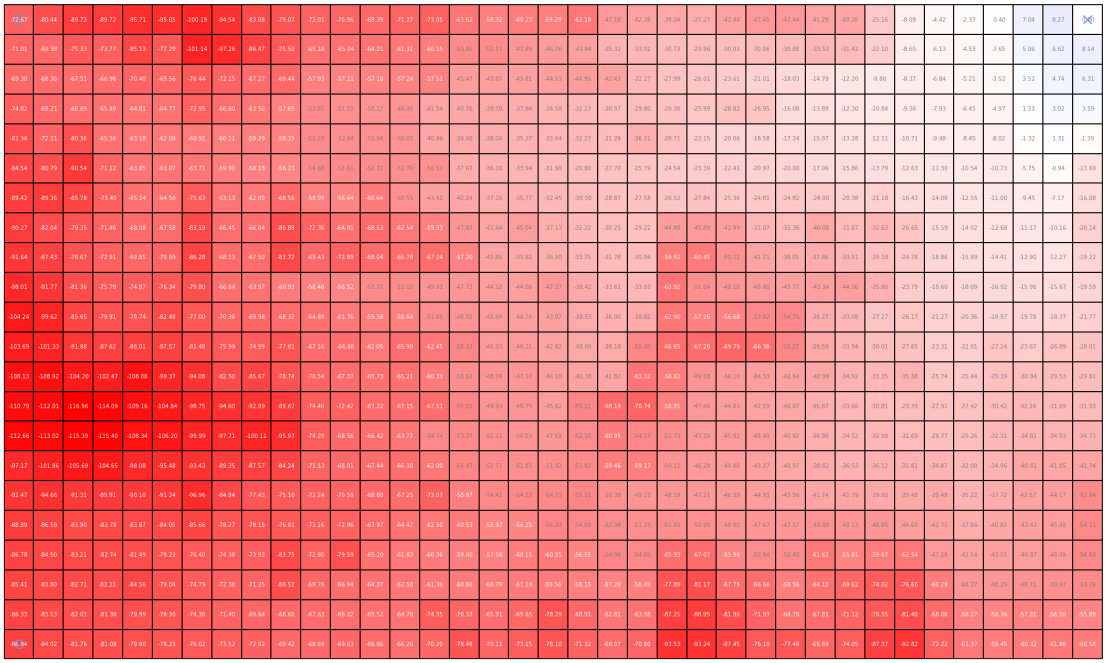

Figure 77: Classifier State Value Map

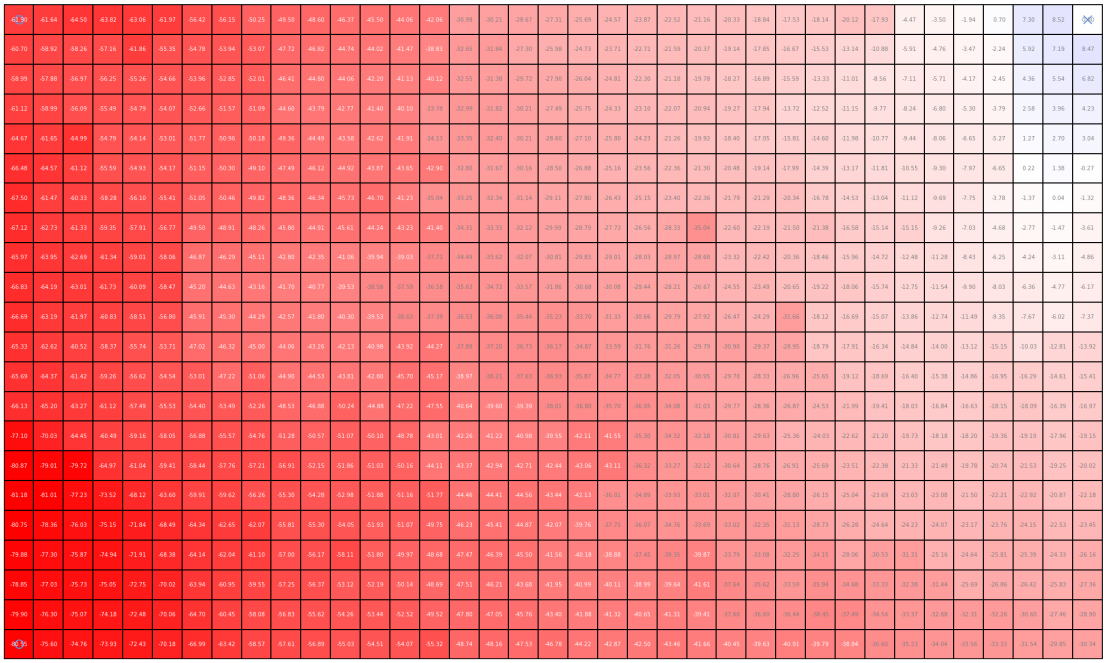

Figure 78: MLIRL State Value Map

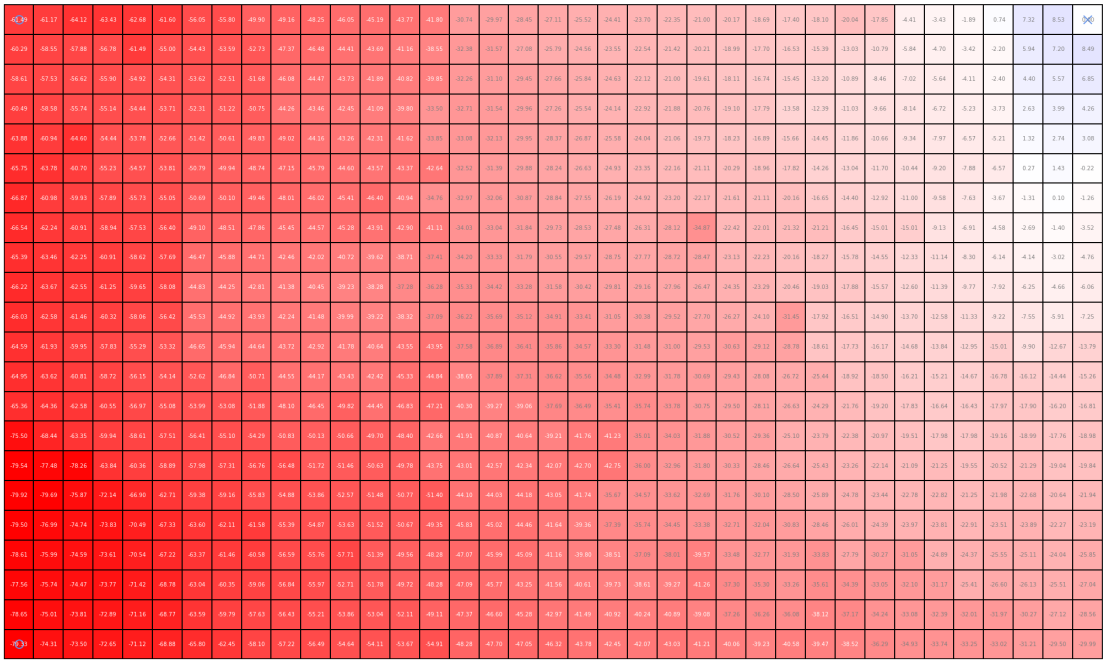

Figure 79: Expert State Value Map

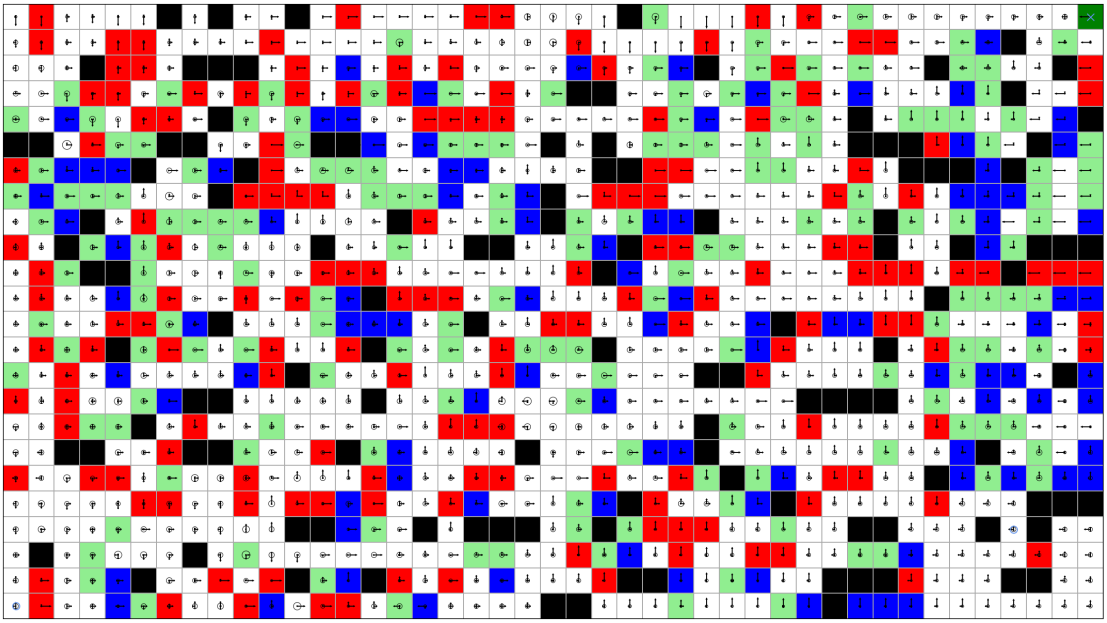

Figure 80: Classifier Policy on Novel Gridworld

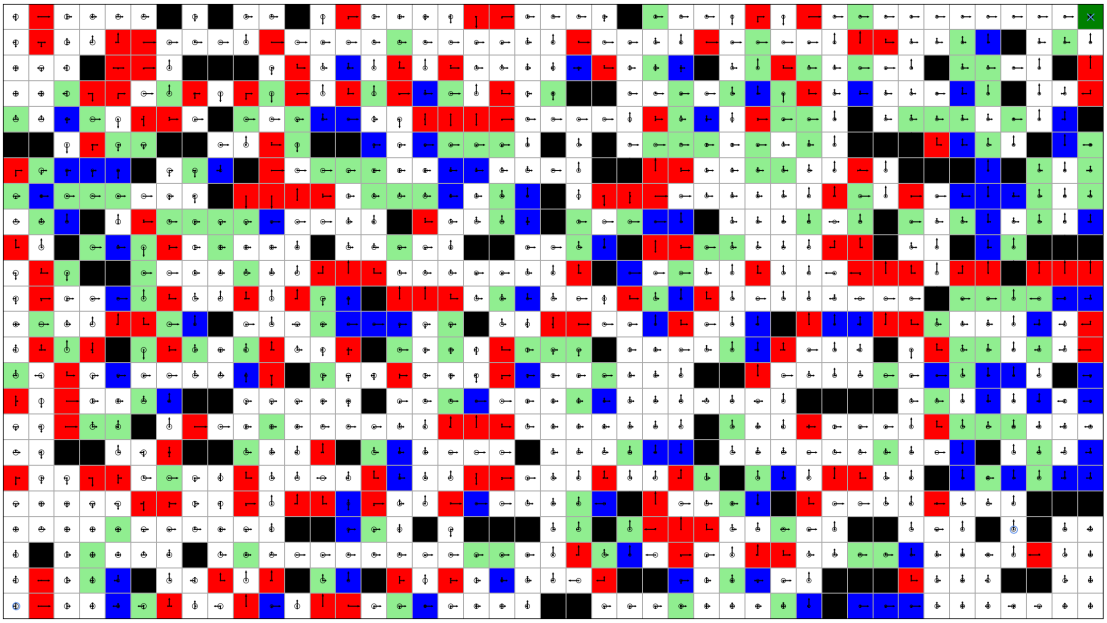

Figure 81: MLIRL Policy on Novel Gridworld

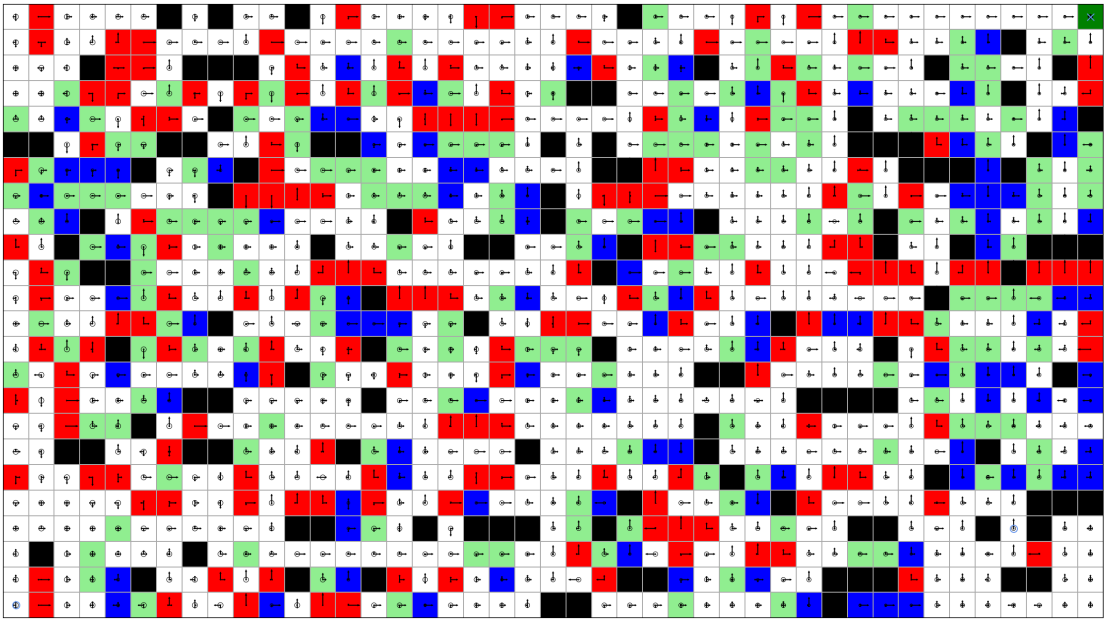

Figure 82: Expert Policy on Novel Gridworld

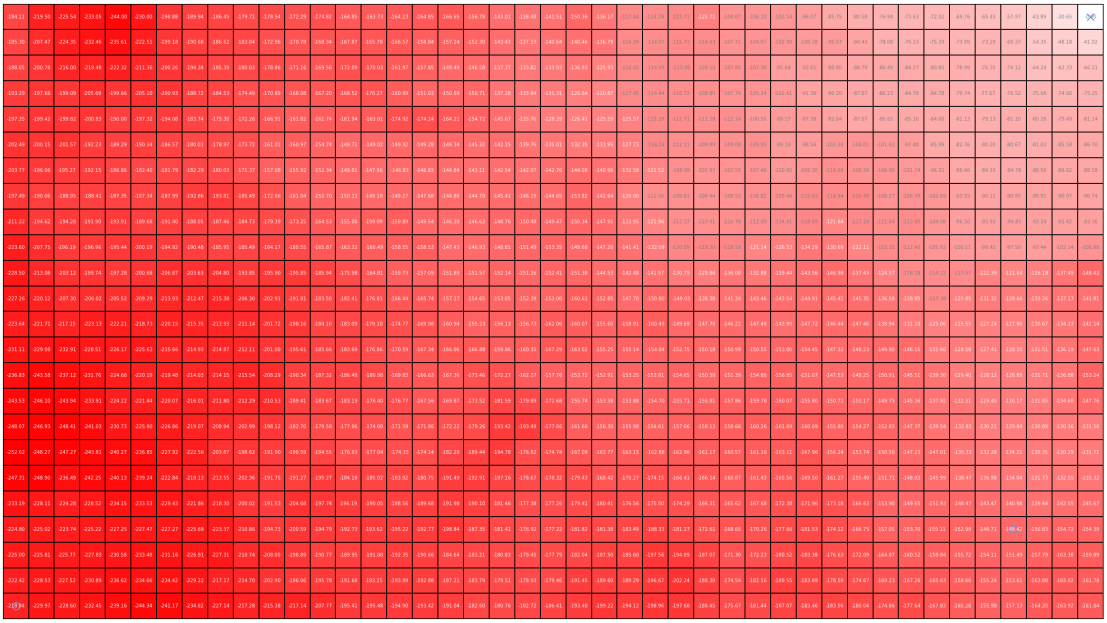

Figure 83: Classifier State Value Map on Novel Gridworld

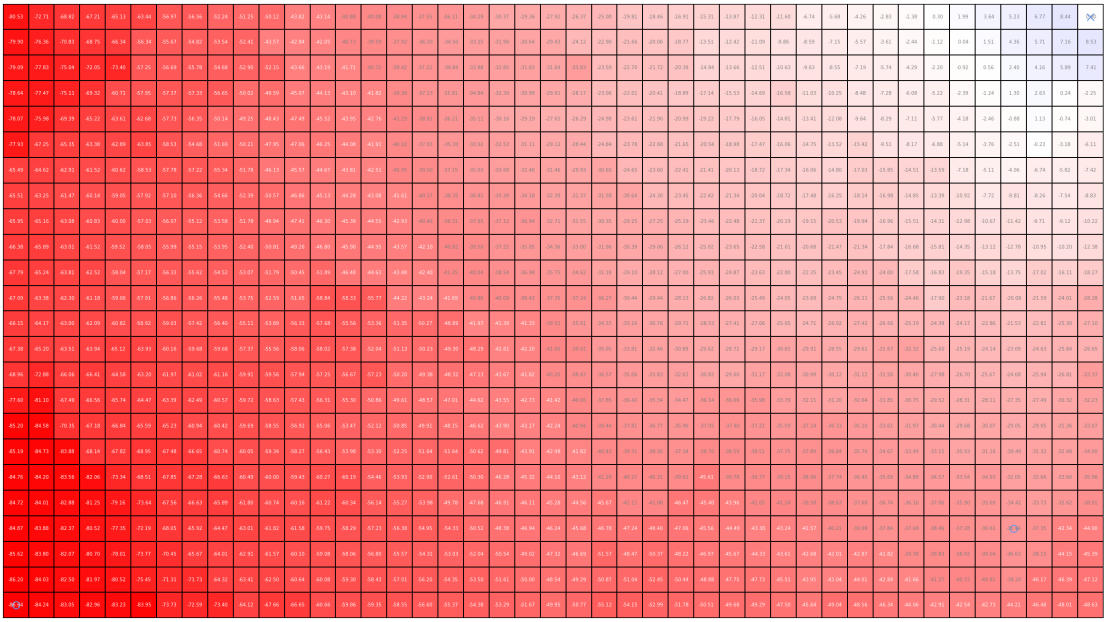

Figure 84: MLIRL State Value Map on Novel Gridworld

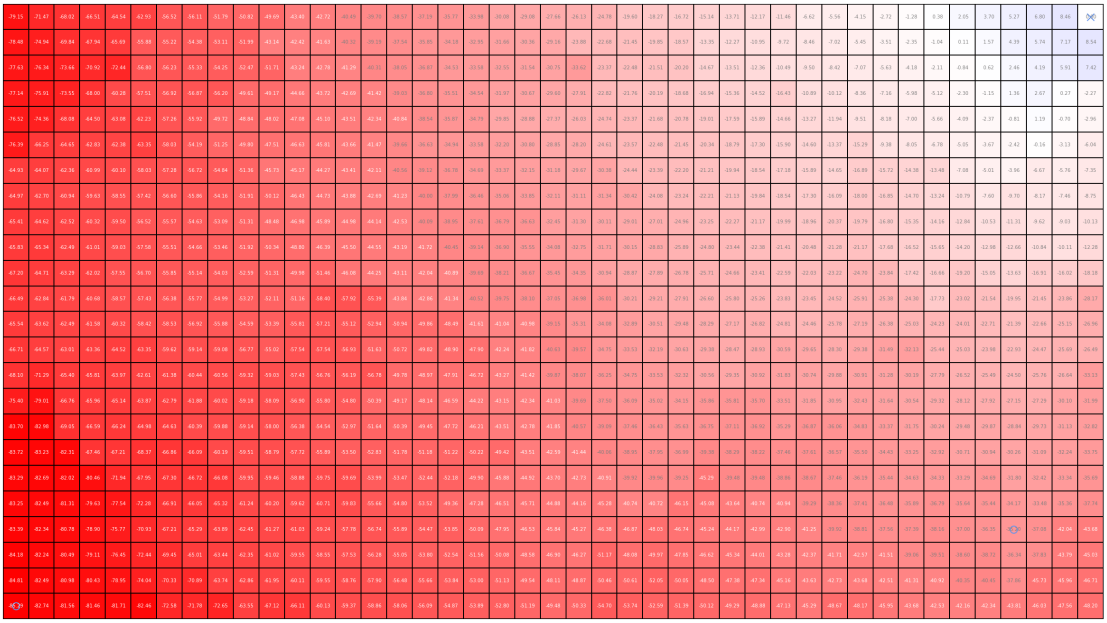

Figure 85: Expert State Value Map on Novel Gridworld

10.5 Classifier and MLIRL Results on Gridworld 4

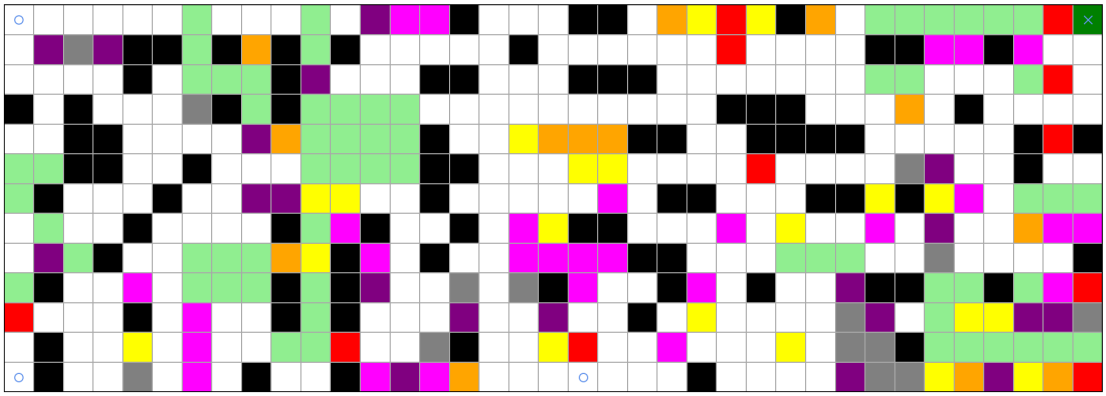

Figure 86: Gridworld 4

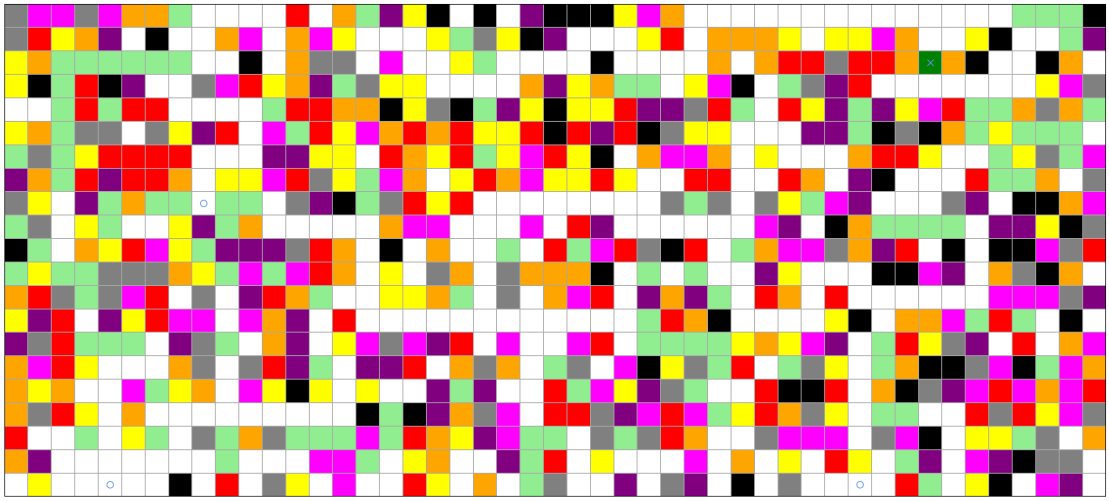

Figure 87: Novel Gridworld for Gridworld 4

Trained on 10000 trajectories with batch size of 128 for 50 epochs.

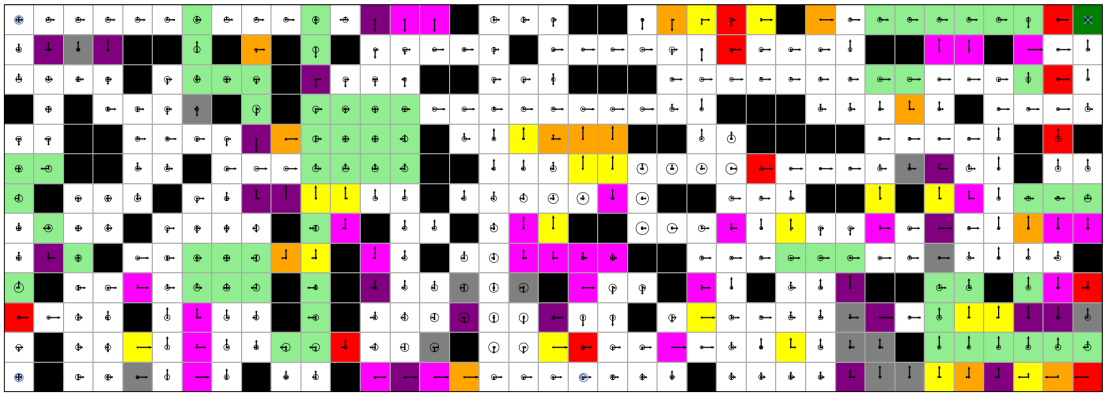

Figure 88: Classifier Policy

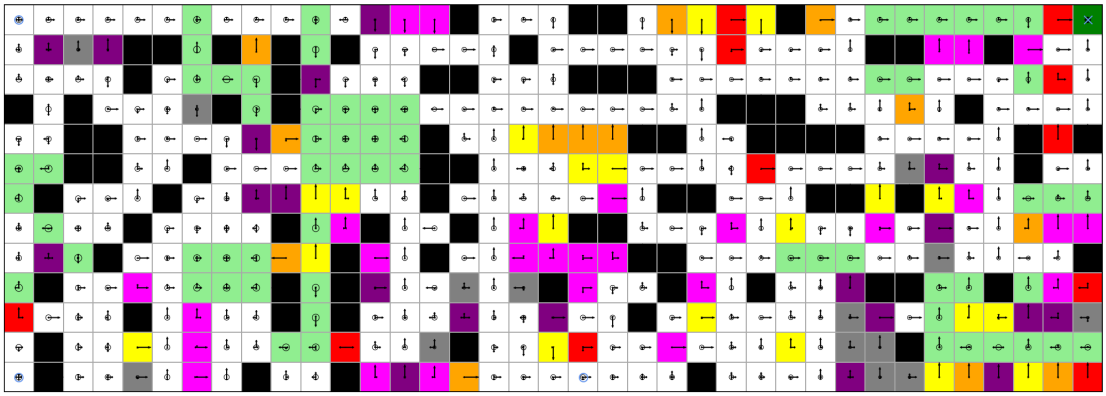

Figure 89: MLIRL Policy

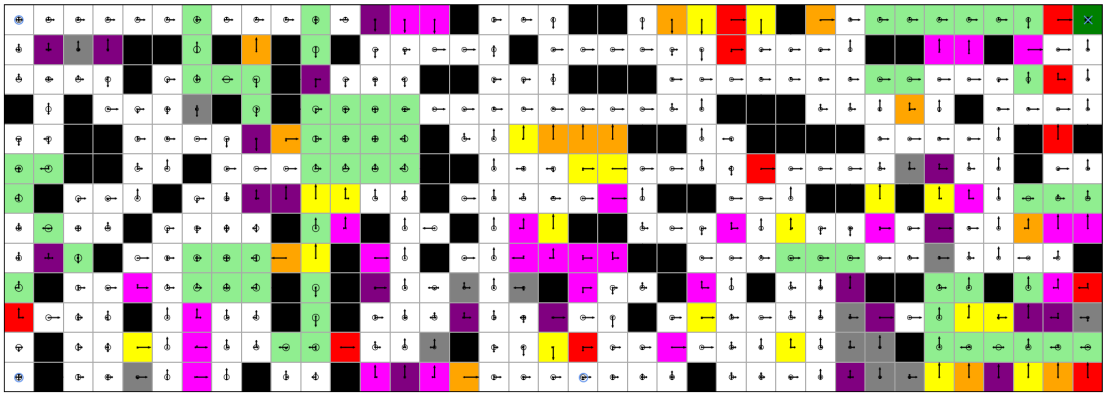

Figure 90: Expert Policy

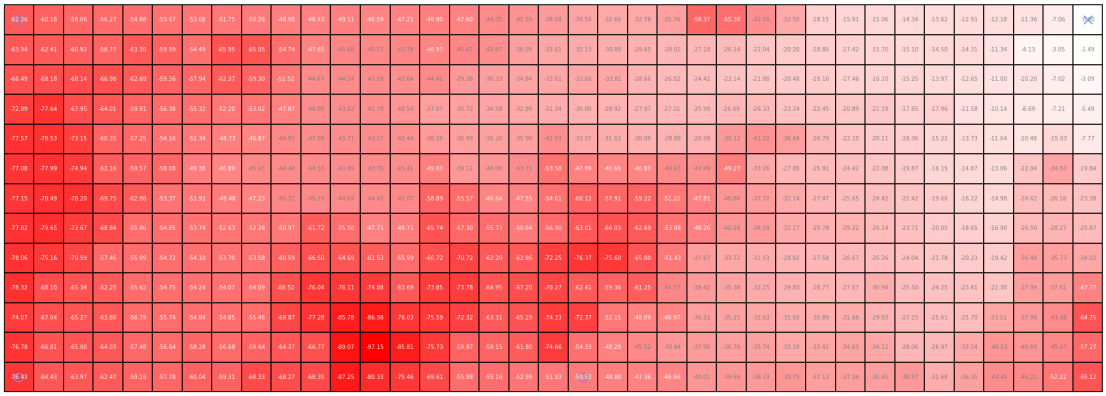

Figure 91: Classifier State Value Map

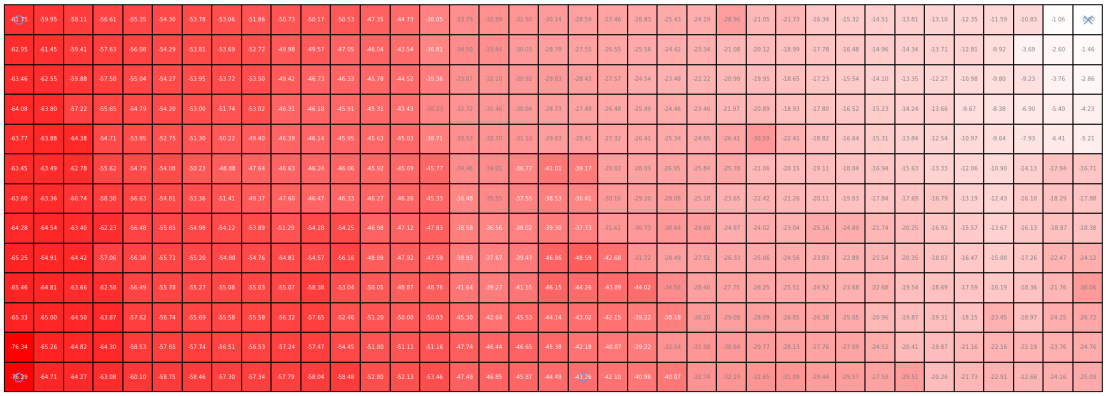

Figure 92: MLIRL State Value Map

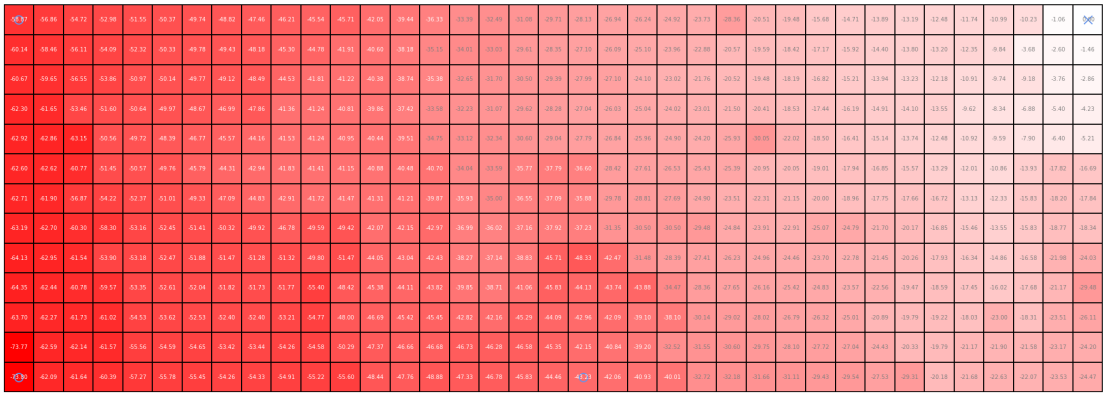

Figure 93: Expert State Value Map

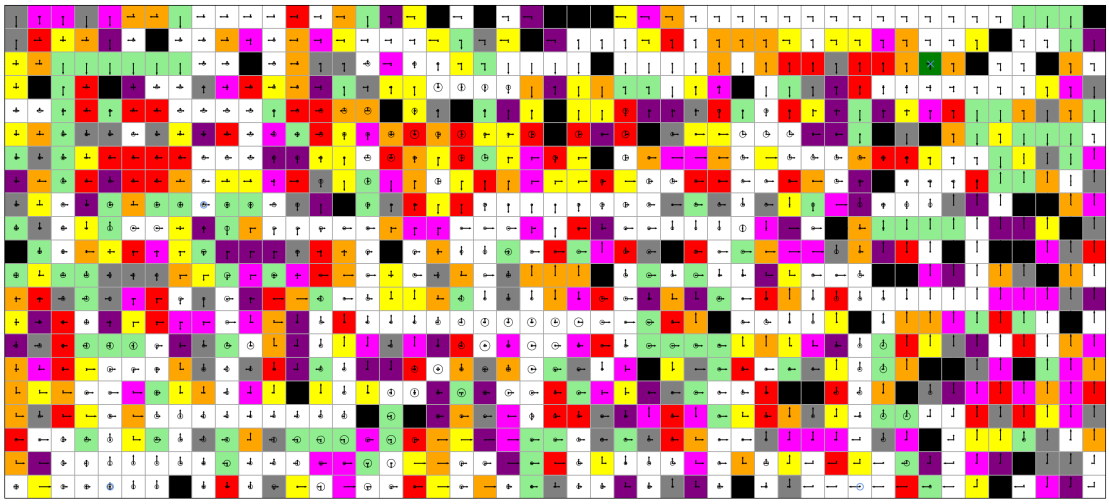

Figure 94: Classifier Policy on Novel Gridworld

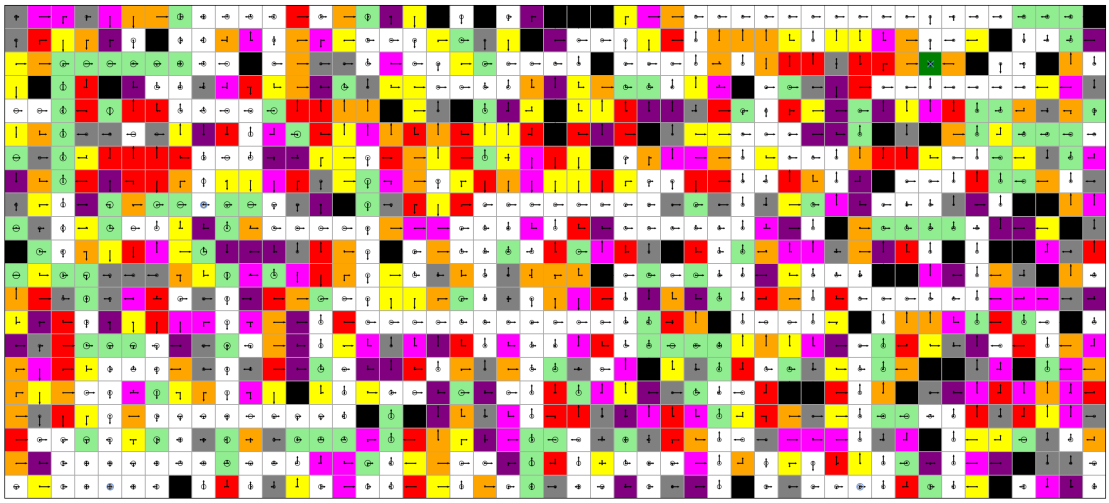

Figure 95: MLIRL Policy on Novel Gridworld

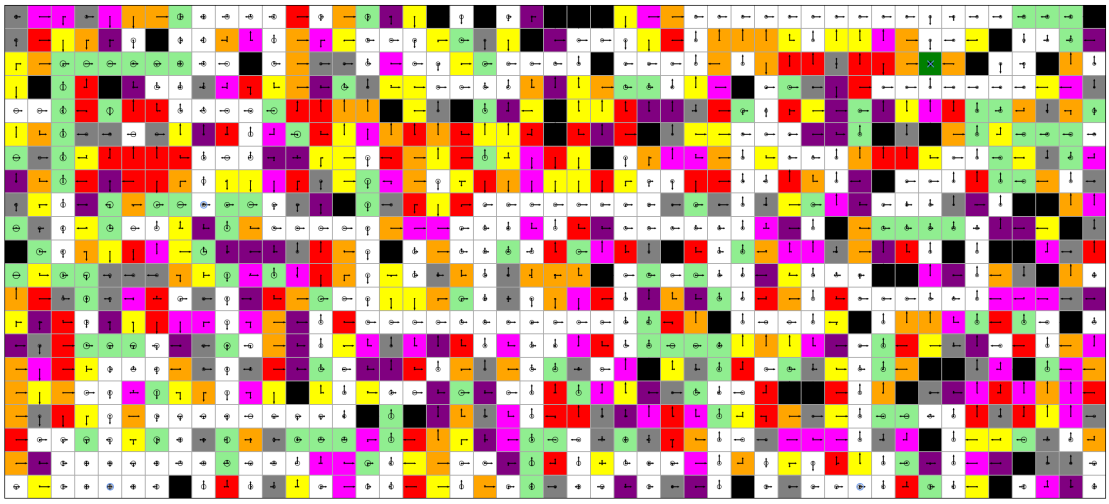

Figure 96: Expert Policy on Novel Gridworld

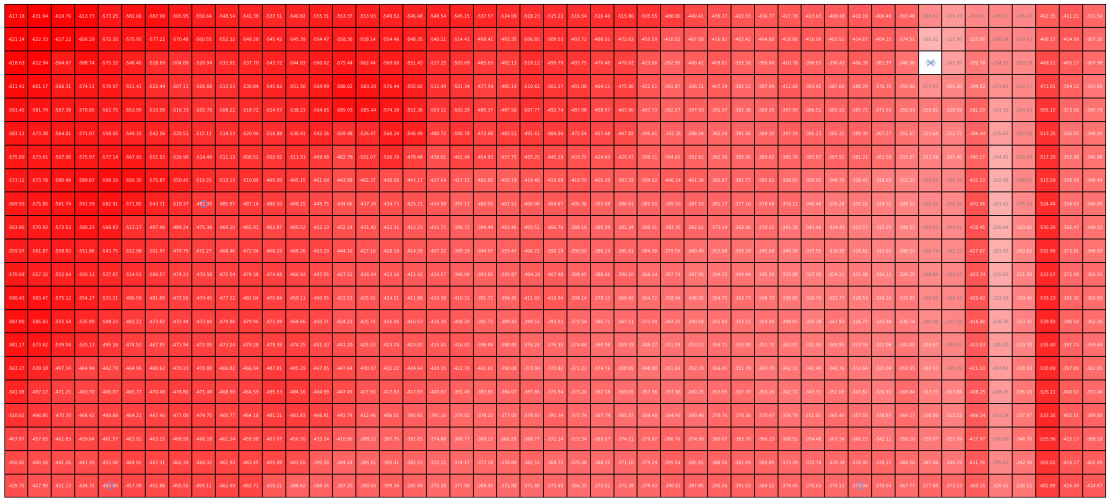

Figure 97: Classifier State Value Map on Novel Gridworld

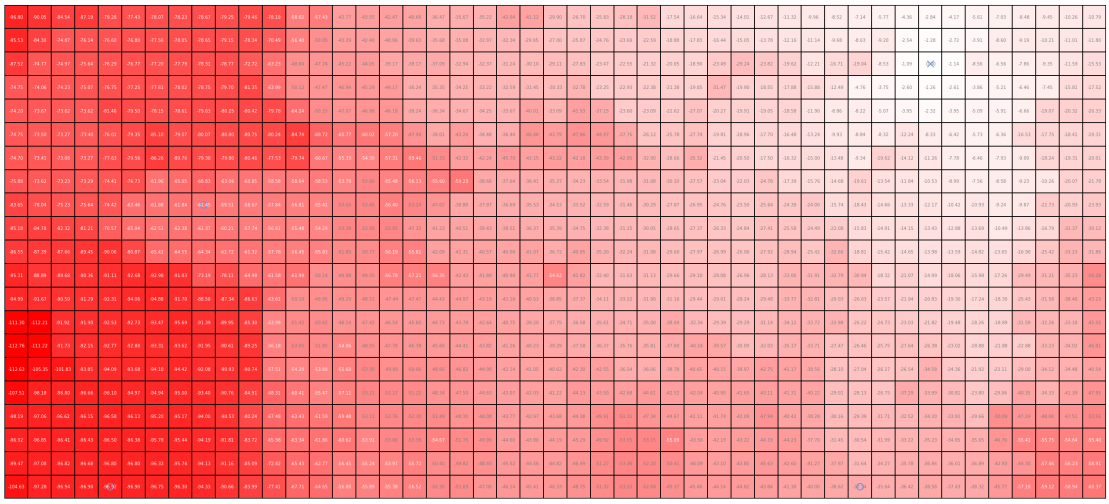

Figure 98: MLIRL State Value Map on Novel Gridworld

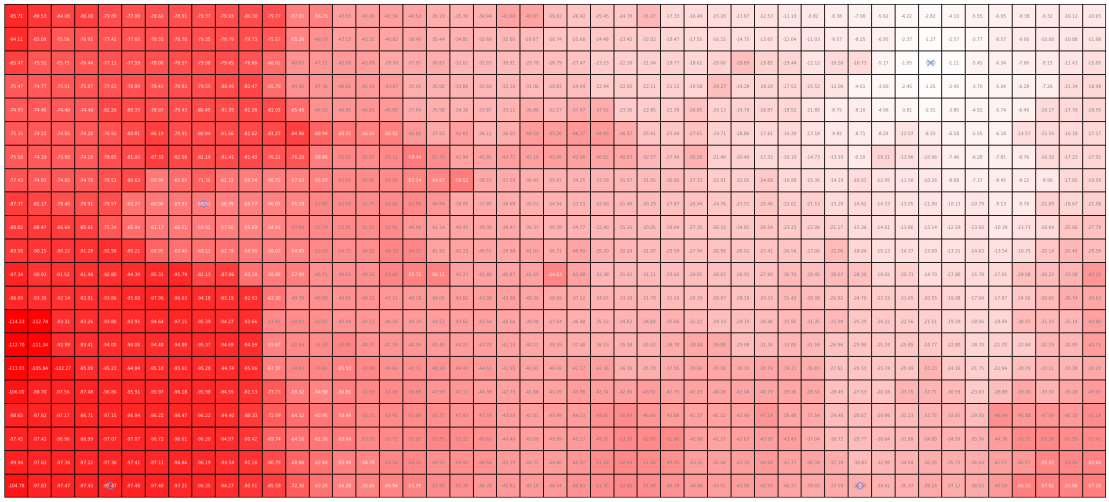

Figure 99: Expert State Value Map on Novel Gridworld

10.6 Classifier Training with 1-hot Vector Representation

Trained on 10000 trajectories with batch size of 128 for 50 epochs on Figure 3. All other hyperparameters remain the same.

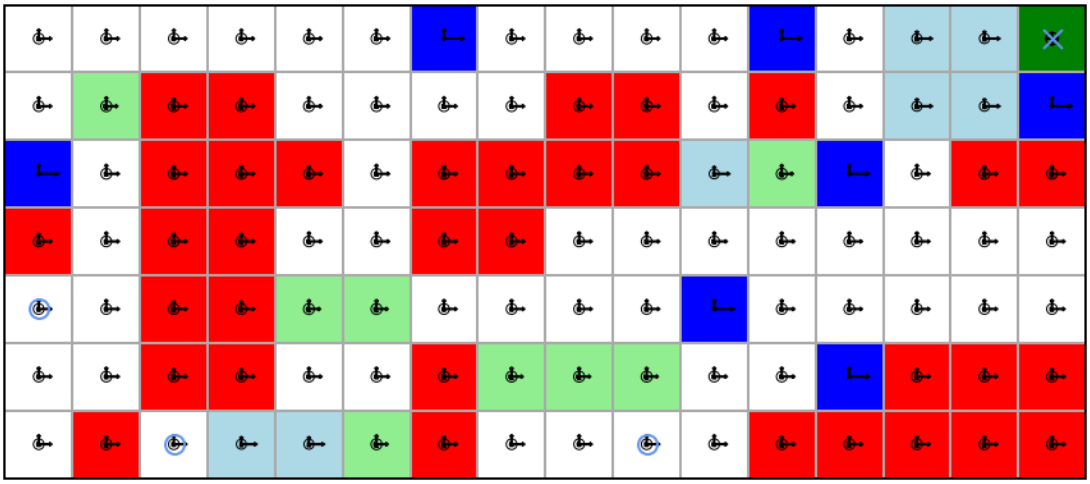

Figure 100: Graph of the classifier's learned policy.

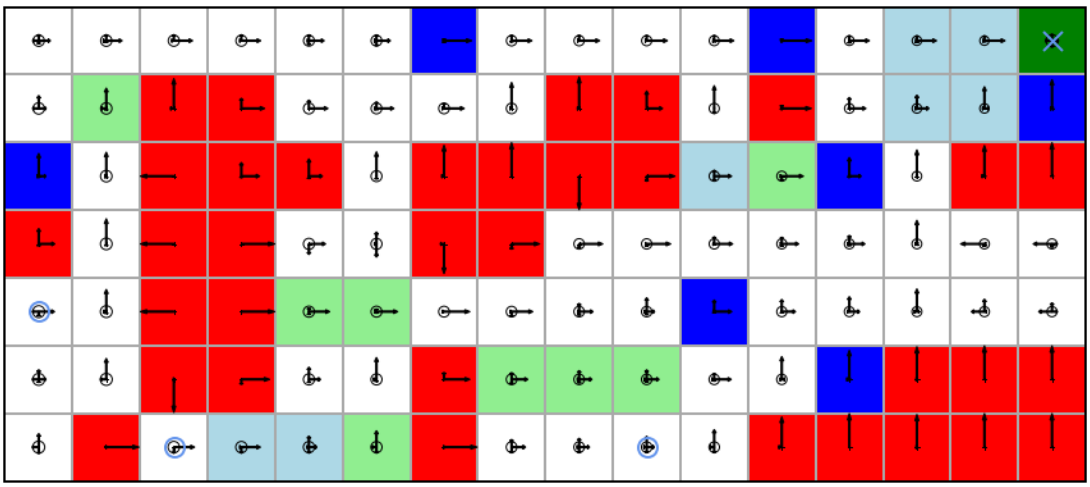

Figure 101: Graph of the expert's policy.

Figure 102: Classifier's state value map.

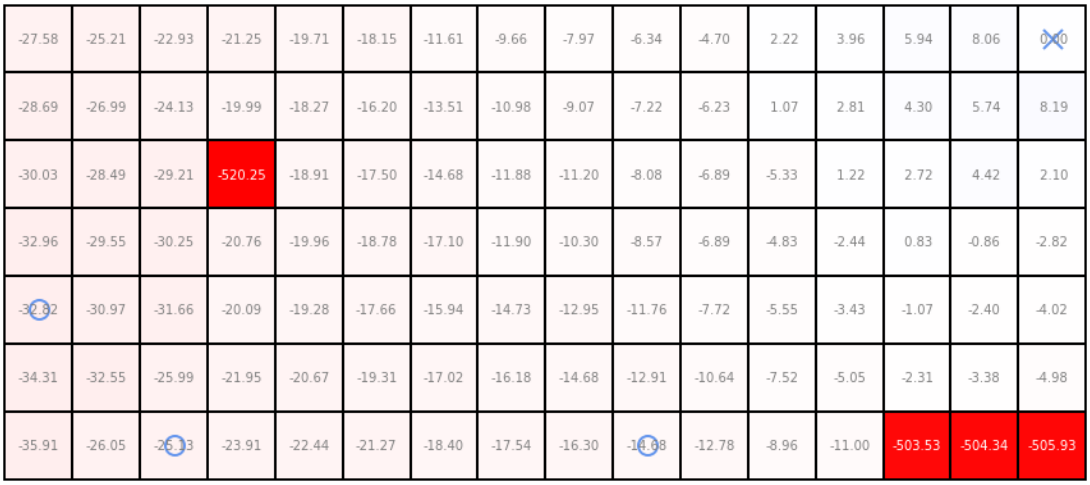

Figure 103: Expert's state value map.

10.7 CMIRL Results on Gridworld 1

Trained on 10000 trajectories with batch size of 128 for 10 epochs.

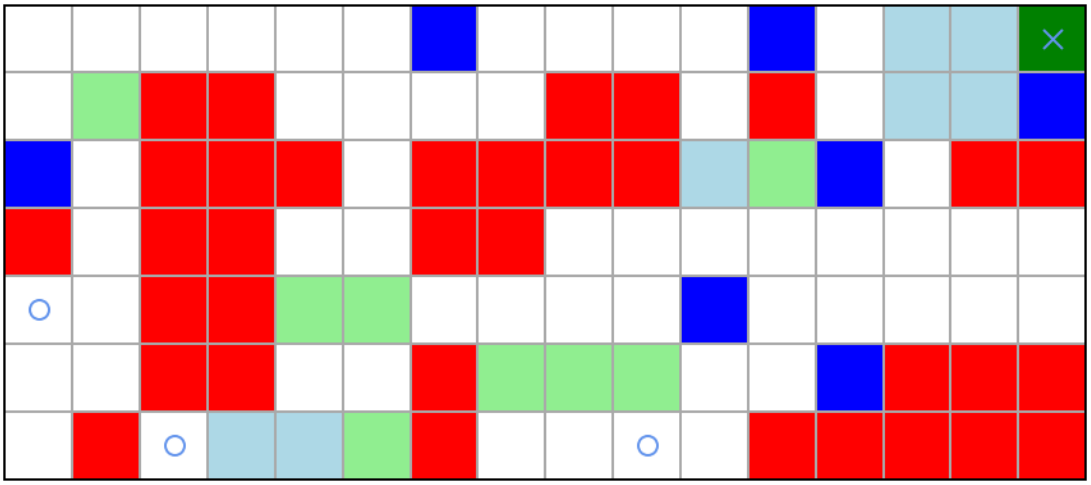

Figure 104: Gridworld 1

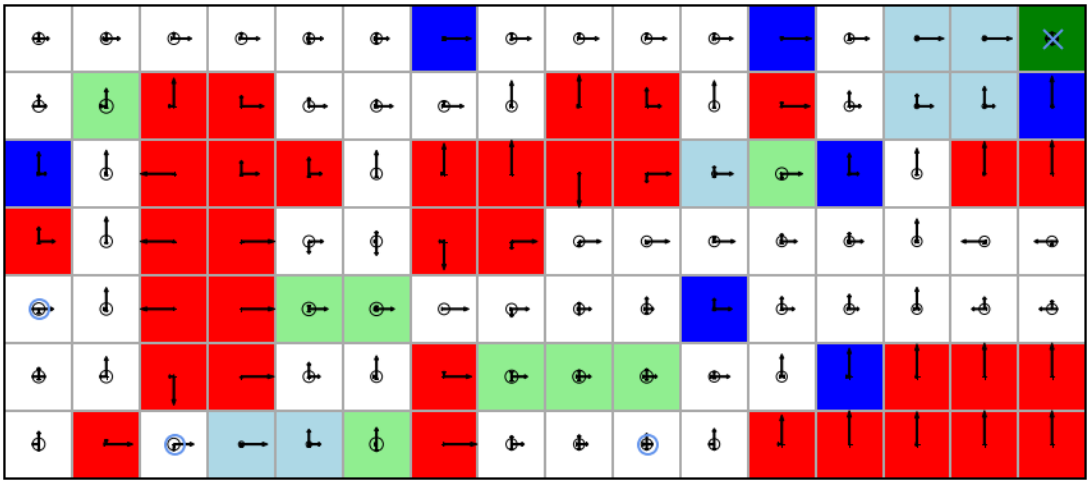

Figure 105: CMIRL Policy

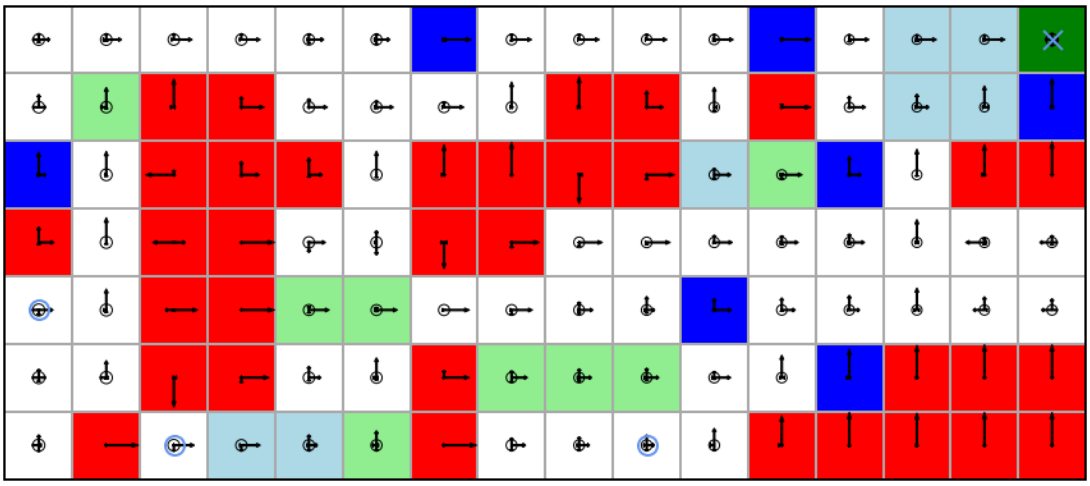

Figure 106: MLIRL Policy

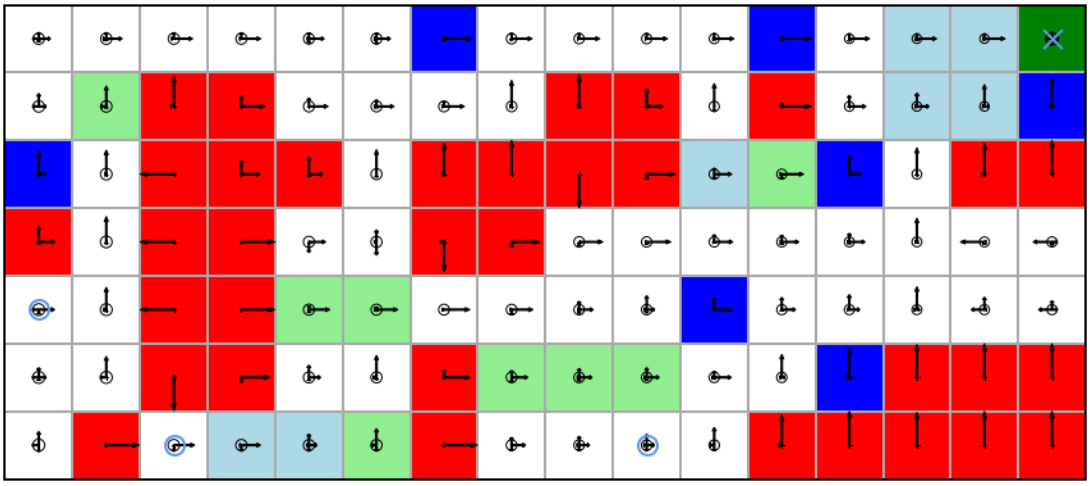

Figure 107: Expert Policy

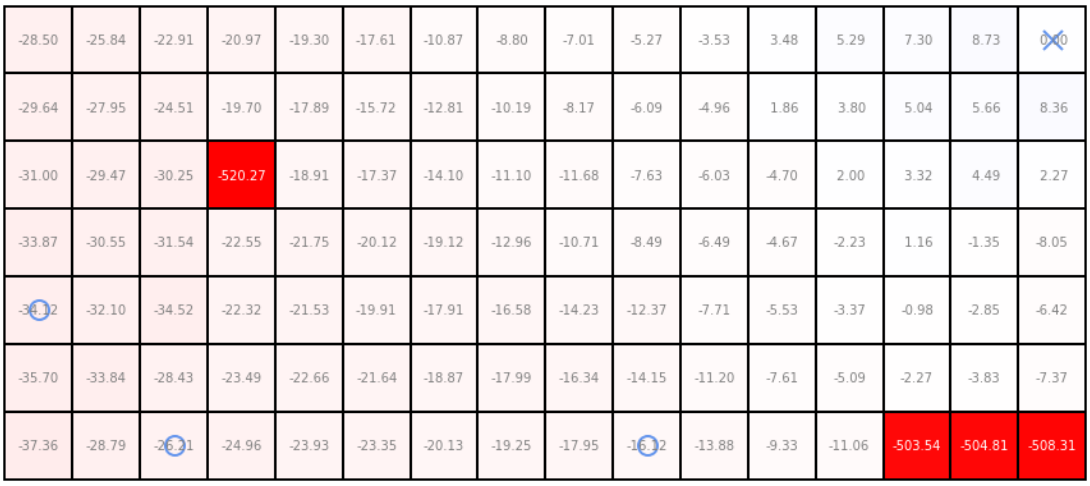

Figure 108: CMIRL State Value Map

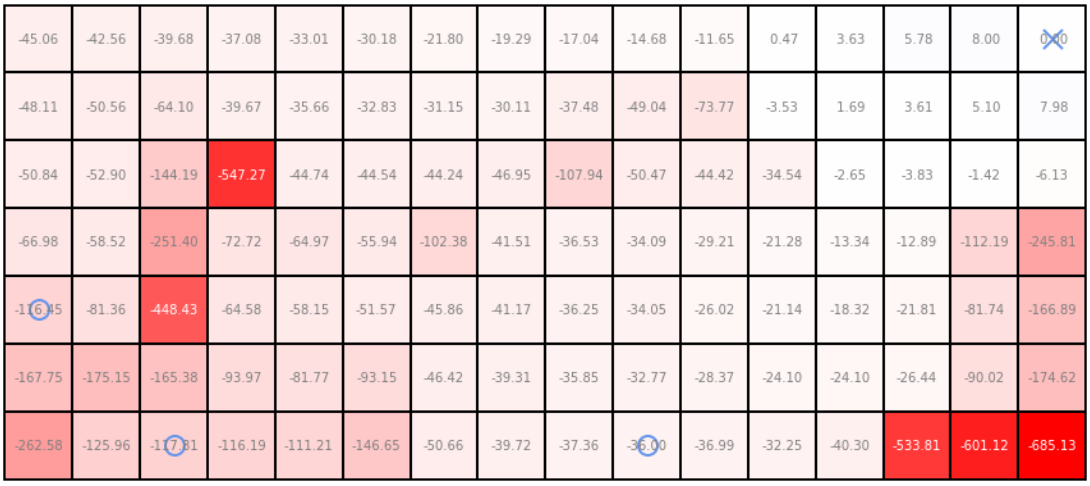

Figure 109: MLIRL State Value Map

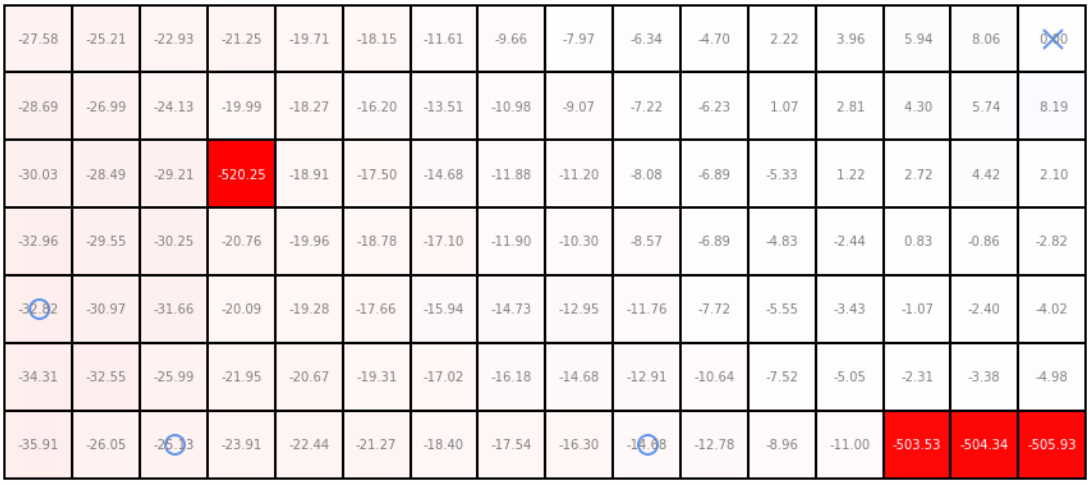

Figure 110: Expert State Value Map

10.8 CMIRL Results on Gridworld 5

Trained on 10000 trajectories with batch size of 128 for 10 epochs.

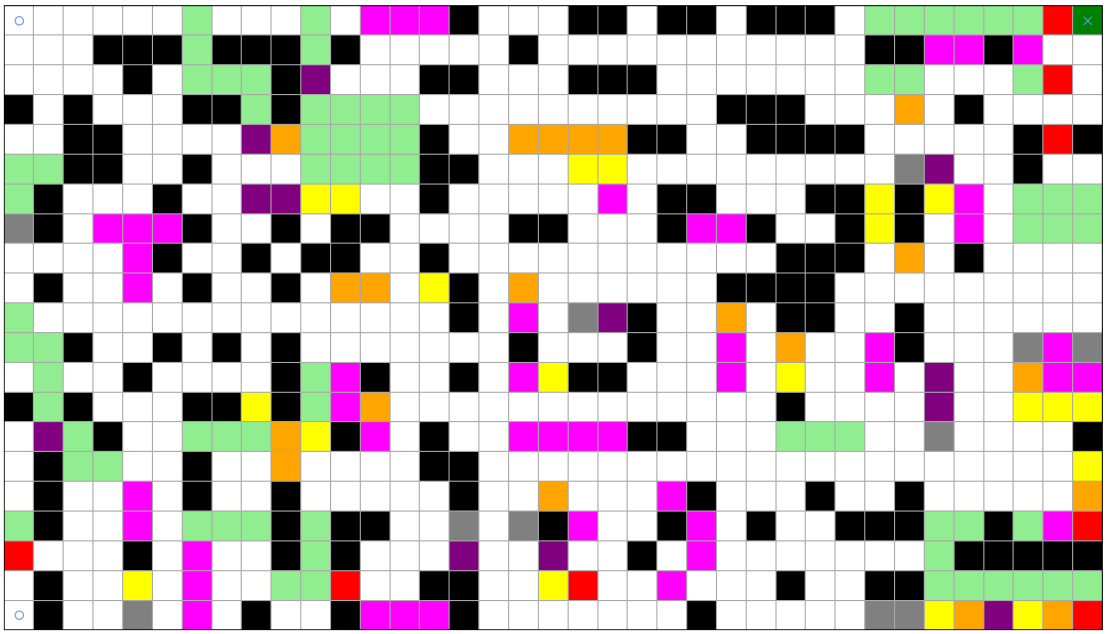

Figure 111: Gridworld 5

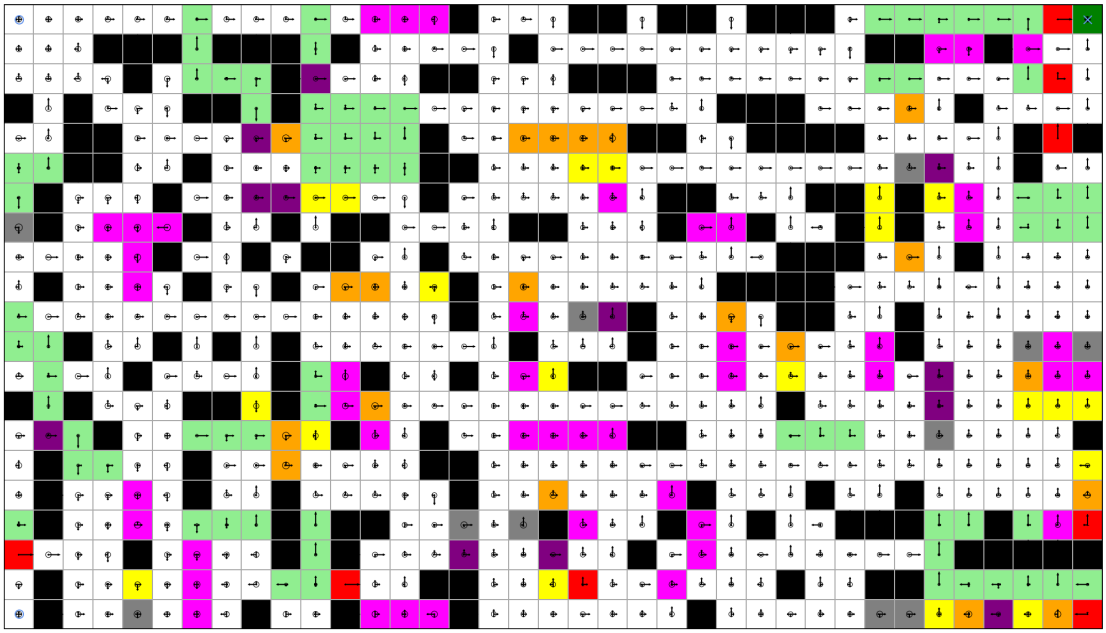

Figure 112: CMIRL Policy

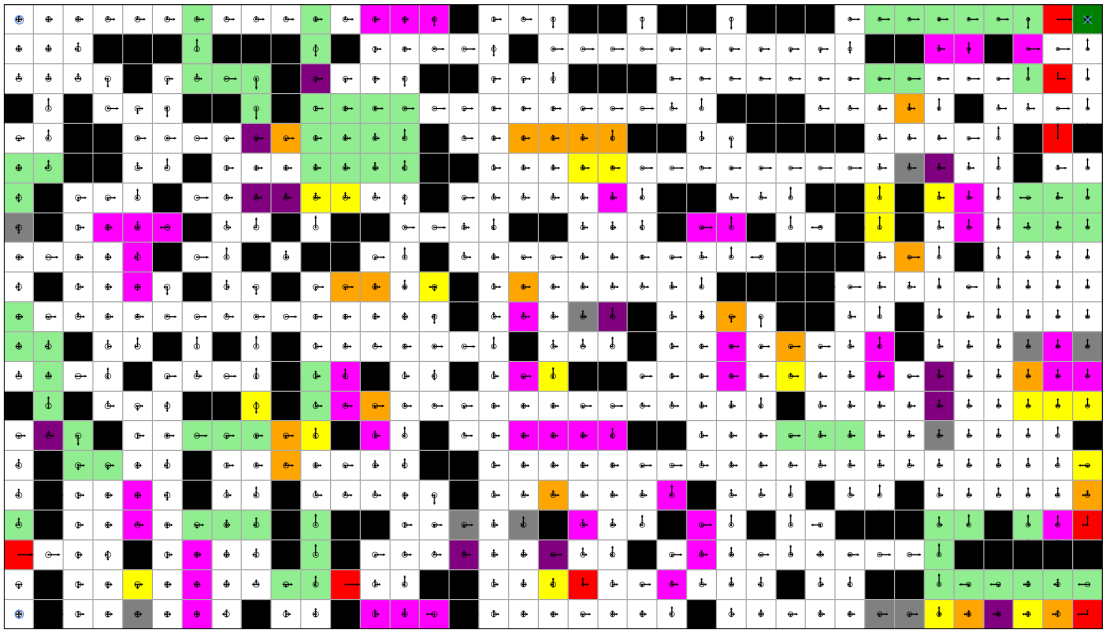

Figure 113: MLIRL Policy

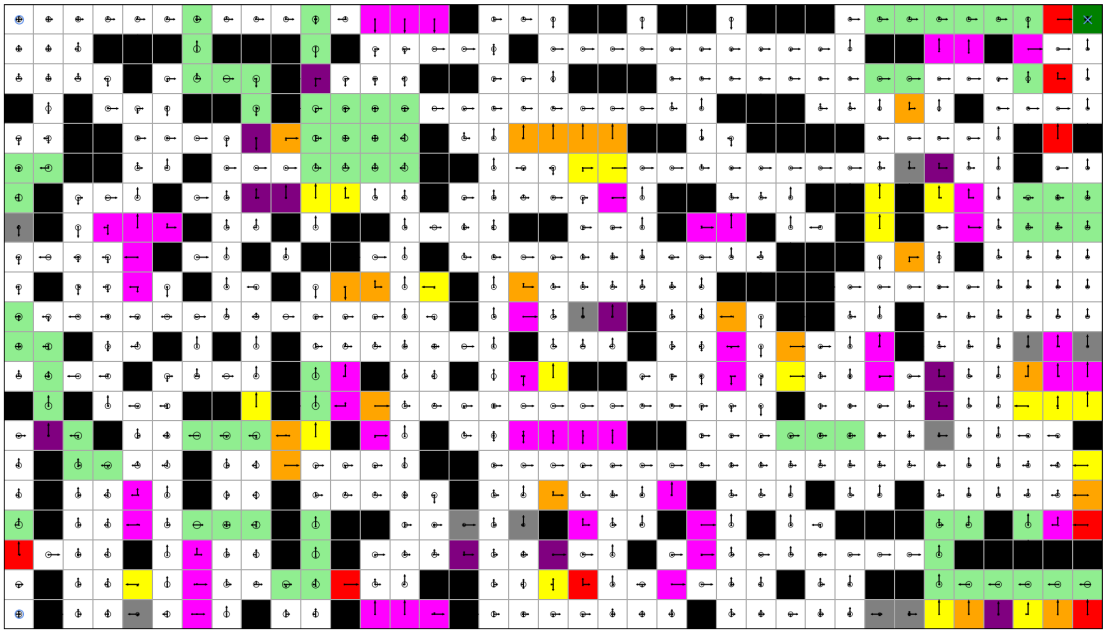

Figure 114: Expert Policy

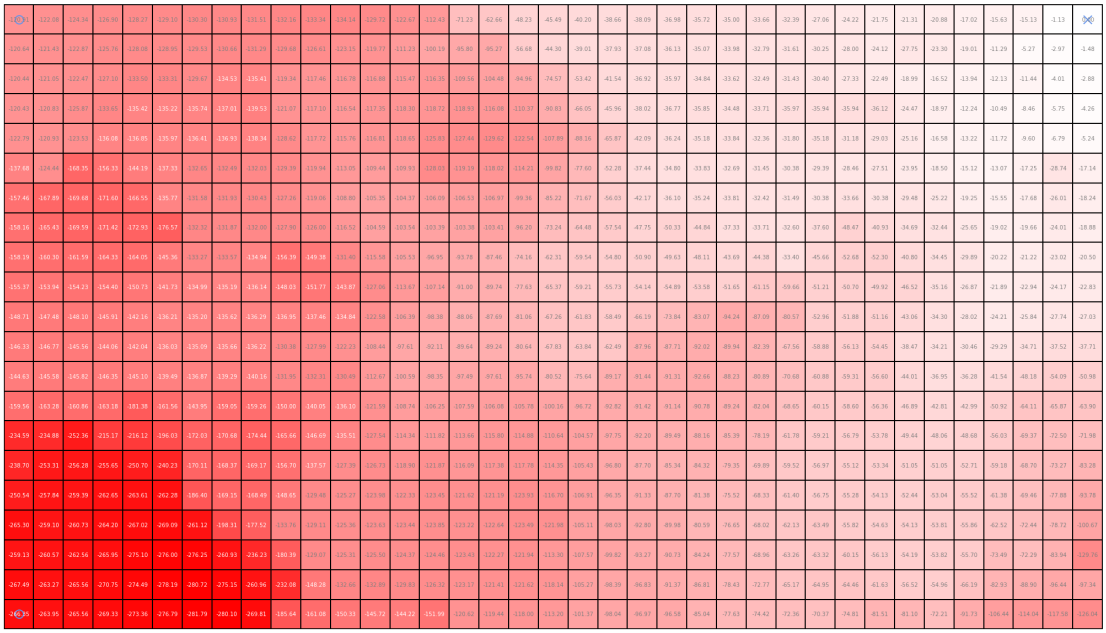

Figure 115: CMIRL State Value Map

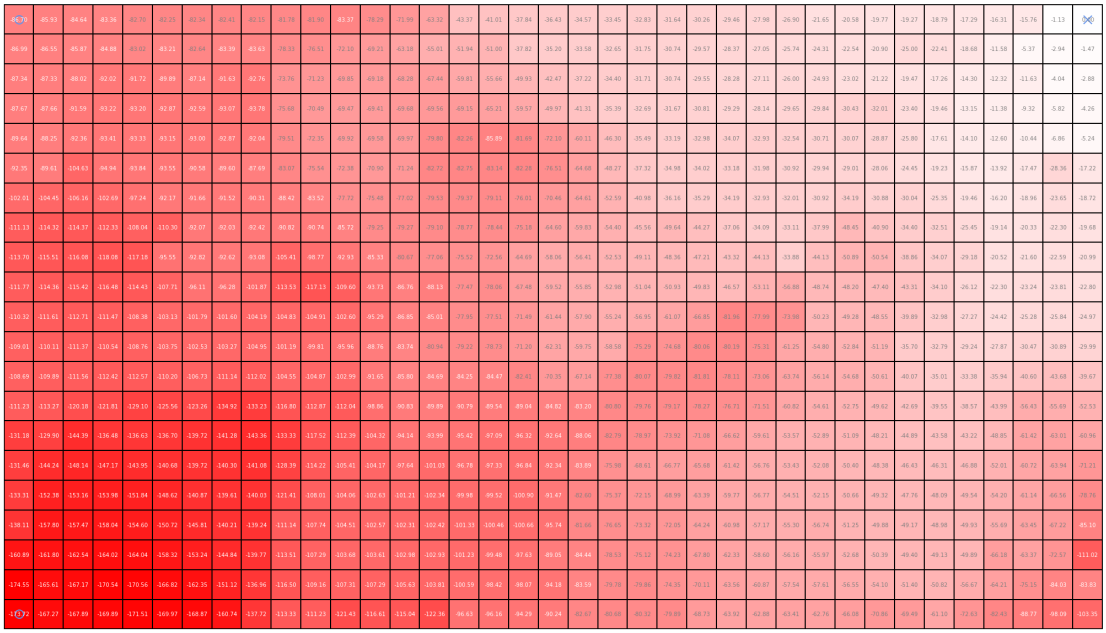

Figure 116: MLIRL State Value Map

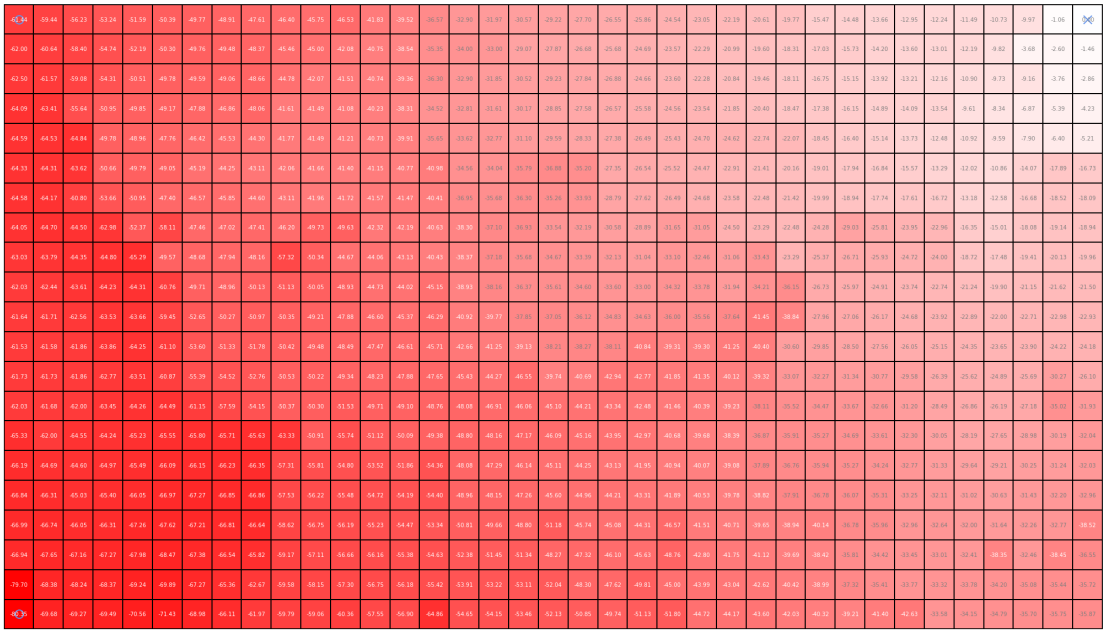

Figure 117: Expert State Value Map

10.9 CMIRL Results on Gridworld 6

Trained on 10000 trajectories with batch size of 128 for 10 epochs.

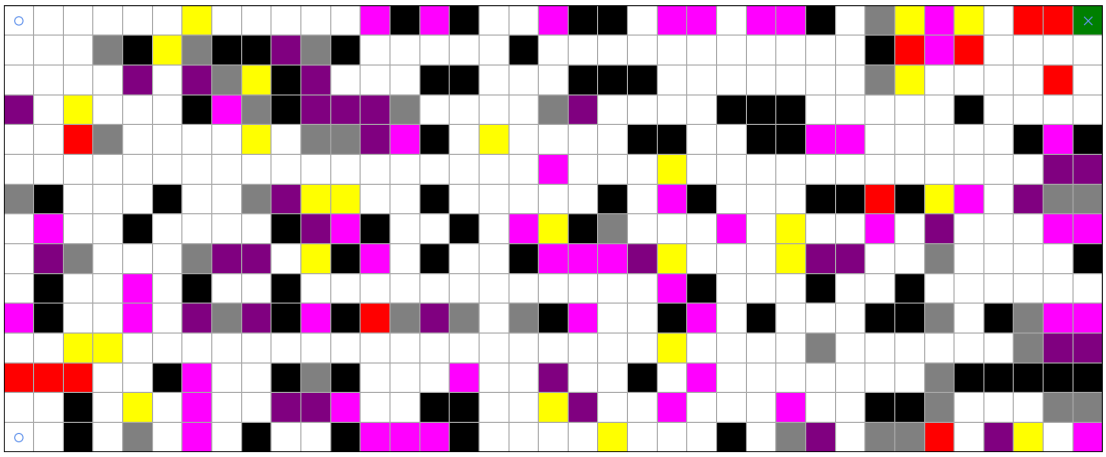

Figure 118: Gridworld 6

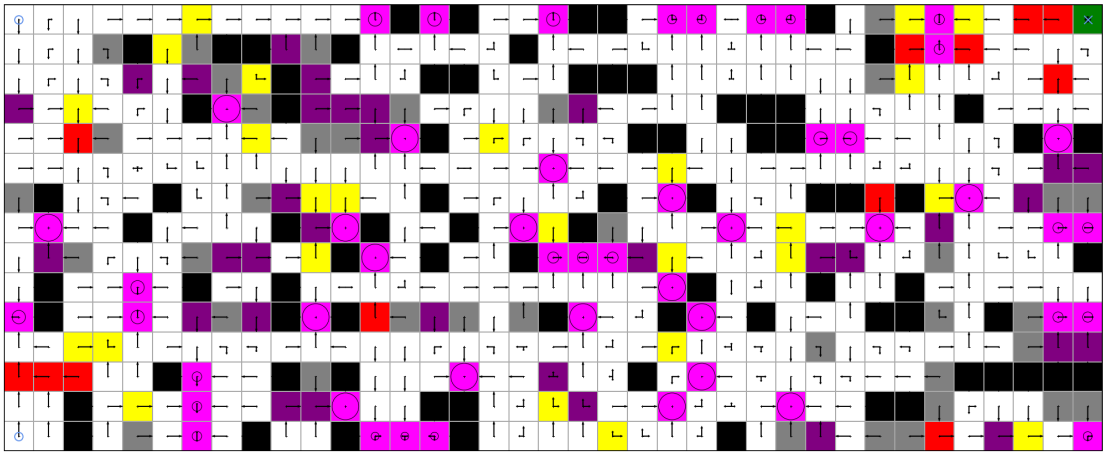

Figure 119: CMIRL Policy

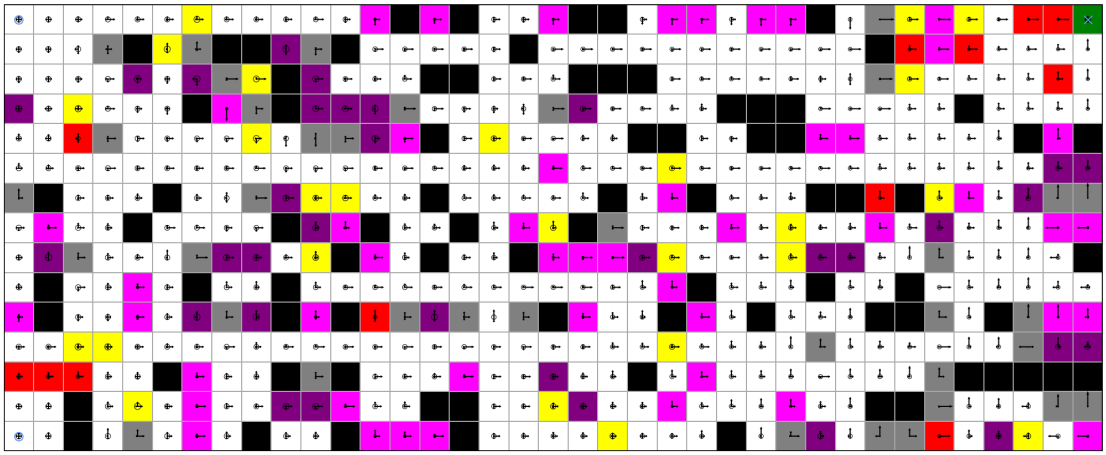

Figure 120: MLIRL Policy

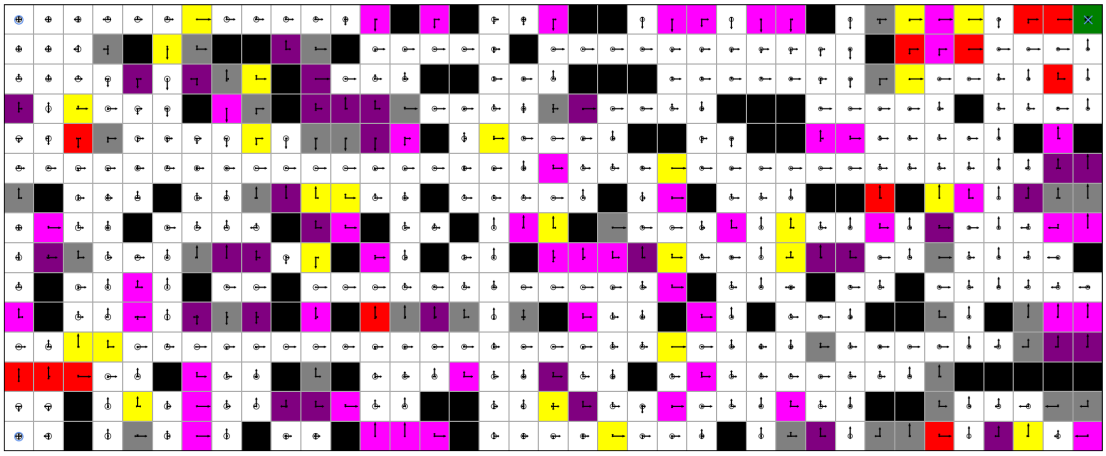

Figure 121: Expert Policy

Figure 122: CMIRL State Value Map

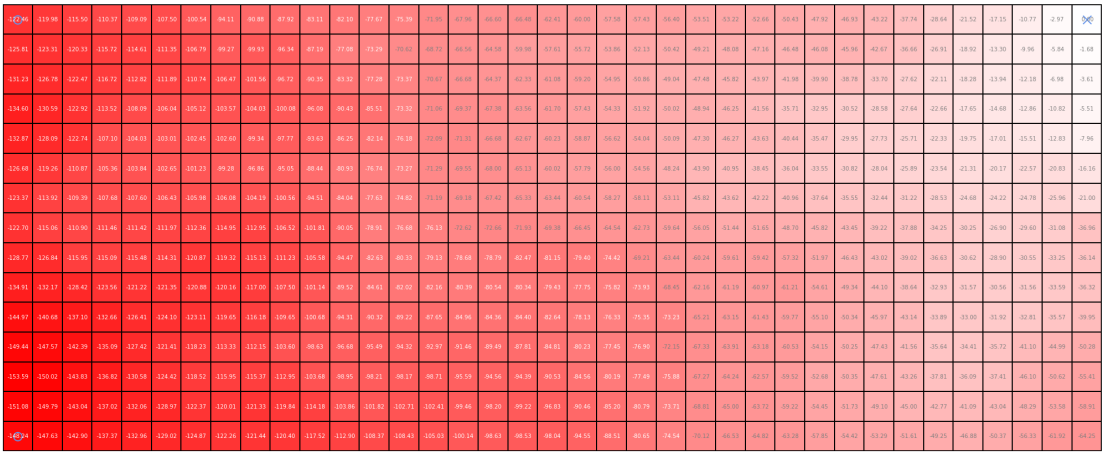

Figure 123: MLIRL State Value Map

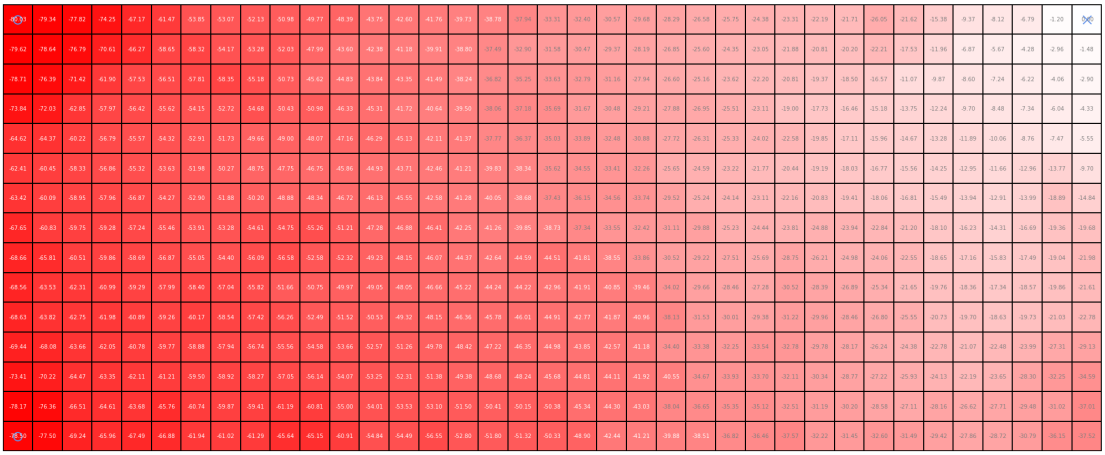

Figure 124: Expert State Value Map

10.10 CMIRL Results on Gridworld 7

Trained on 10000 trajectories with batch size of 128 for 10 epochs.

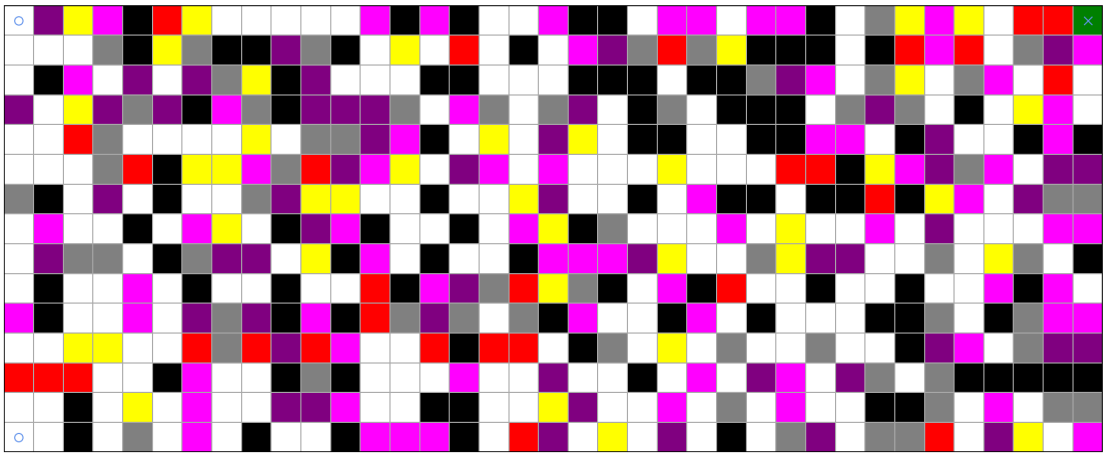

Figure 125: Gridworld 7

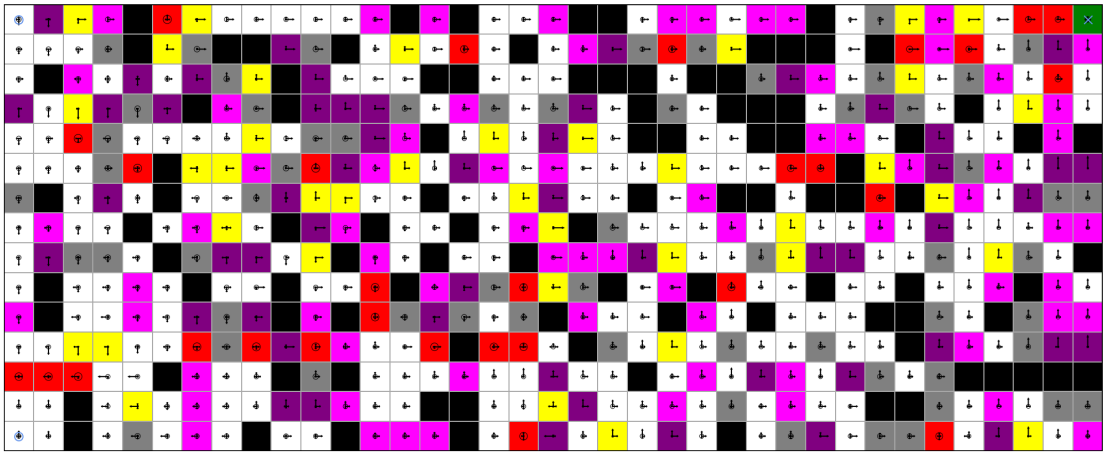

Figure 126: CMIRL Policy

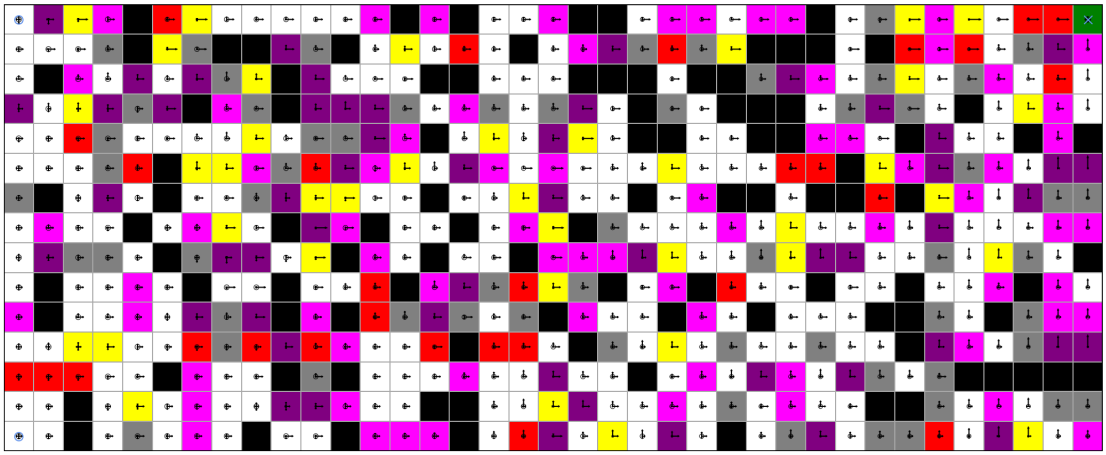

Figure 127: MLIRL Policy

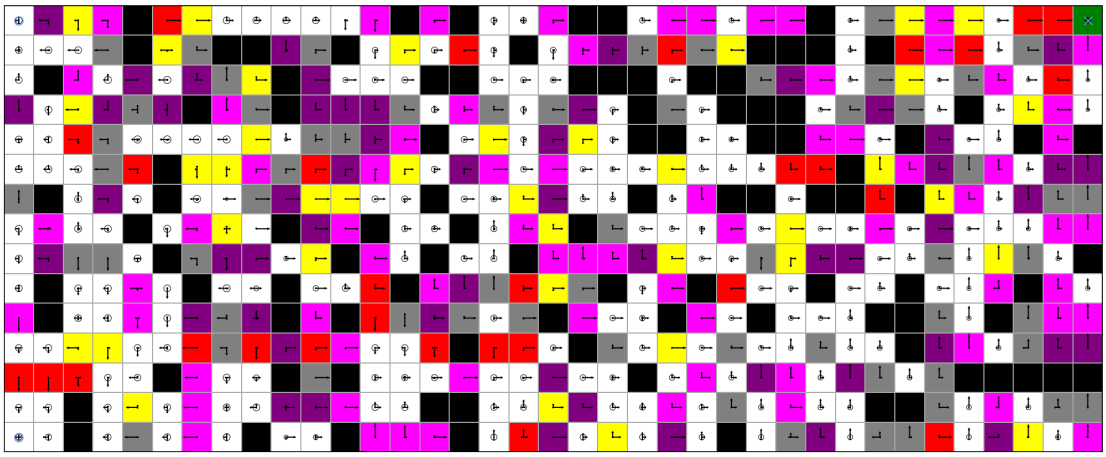

Figure 128: Expert Policy

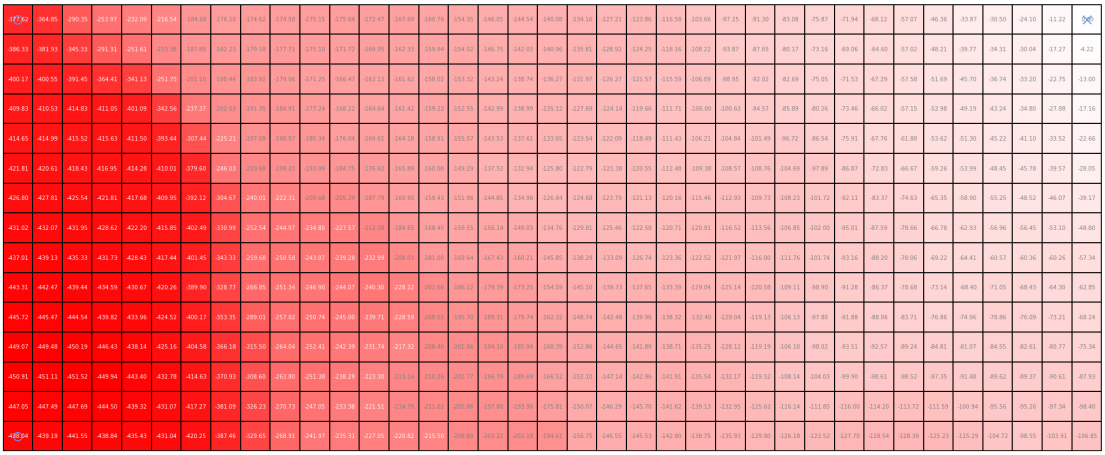

Figure 129: Classifier State Value Map

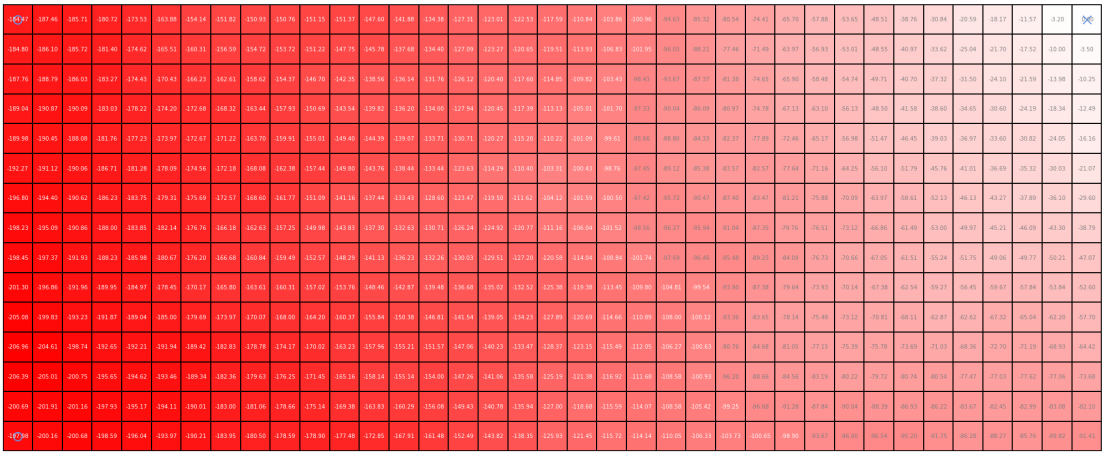

Figure 130: MLIRL State Value Map

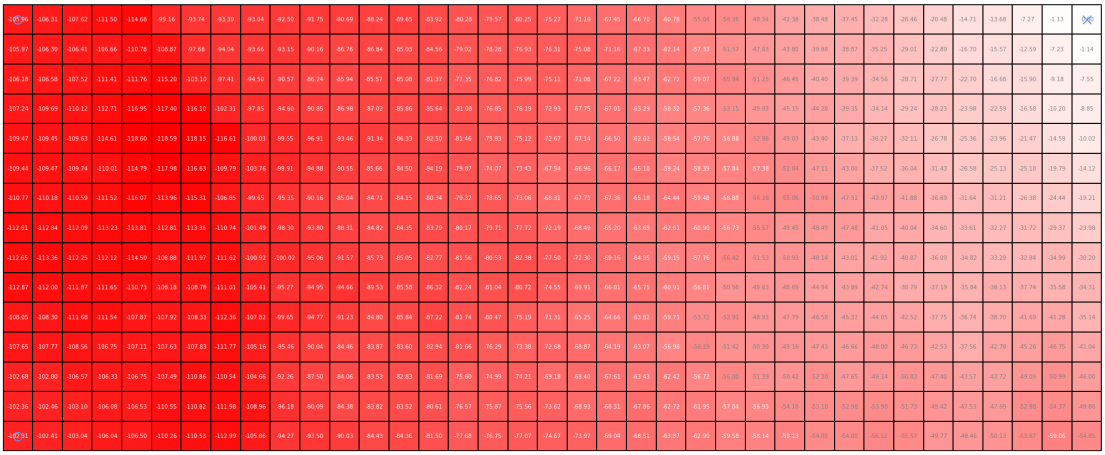

Figure 131: Expert State Value Map