# Exploring the combination of reward shaping and shielding techniques in a single-agent grid world environment using Q-learning

**Bart Claessens (r0578986) & Thomas Claessens (r0659644)**
**5 April 2020**

## Abstract

This report discusses implementation details for the project submitted for Contemporary AI Topics. We chose to use the taxi environment from the OpenAI Gym library. On this taxi environment we want to apply the techniques introduced in both papers. We started with a vanilla Q-learning agent and modified it to choose a random action when multiple actions are equally good (e.g. no ranking between actions exists). Then we extended this learning algorithm with both the shielding technique introduced in the first paper and the reward shaping technique from the second paper. We show and discuss the results of using both techniques separately and simultaneously.

## 1 Introduction

Reinforcement learning is a promising technique which has already shown its usefulness in the domain of machine learning. Think about agents that learn to play games, autonomous robots, self-driving cars, the possibilities are endless. Apart from characteristics like the algorithm's efficiency and convergence speed, there are also still a lot of unanswered questions and (mostly context-related) practicalities which are not really necessary for the algorithm to work well, but therefore not unimportant! An example is unsafe behaviour which becomes really important when placing the agent in a real, physical environment. A possible solution to this problem is introduced in *Safe Reinforcement Learning via Shielding* [4].

When the algorithm is guaranteed to be safe, increasing its performance gets a higher priority. Sparse rewards can cause a learning agent to have myopic behavior. This can cause convergence to a more short-term optimal policy or require a long time to converge to the global optimal policy. A possible solution to this problem (as introduced in *Decision-Making with Non-Markovian Rewards: From LTL to automata-based reward shaping* [1]) is guiding the learner in its search for a solution. This however requires a conversion from the problem to a Markovian structure, on which then reward shaping can be used.

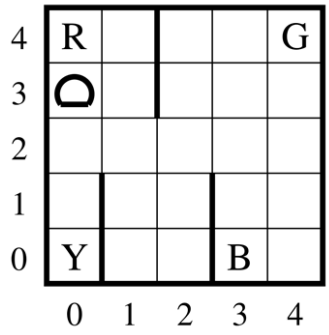


Figure 1: The 5-by-5 taxi grid

## 2 Description of the experiment

The goal of this experiment is to analyse what happens to a reinforcement learning problem when (a combination of) shielding and reward shaping techniques are applied. We will use the Q-learning algorithm as a basis to apply the techniques. Our hypothesis is that the amount of episodes required to find a (best) solution will be reduced when applying reward shaping. It will also be a guarantee that unsafe actions will not be taken when using shielding.

### 2.1 The Environment

The taxi environment we chose to use was originally introduced in 1999 [2]. The environment has a 5-by-5 grid world (displayed in figure 1) inhabited by a taxi agent (shown by the half-circle). There are four specially-designated locations in this world, called hubs, marked as R(ed), B(lue), G(reen), and Y(ellow). The taxi problem is episodic. In each episode, the taxi starts in a randomly-chosen square. There is a passenger at one of the four hubs (chosen randomly), and that passenger wishes to be transported to one of the three other hubs (also chosen randomly). The taxi must go to the passenger's location (the "source"), pick up the passenger, go to the destination hub (the "destination"), and drop-off the passenger there. The episode ends when the passenger is deposited at the destination hub. The 5-by-5 grid is bounded by walls so the taxi can not drive outside of this grid. These are the bold lines in figure 1. Walls also appear inside the grid to make the world a little bit more interesting. When a wall is between two tiles, the taxi is blocked to drive from one tile

to the other and stays on the same tile after performing this action. Therefore, the taxi cannot drive through walls.

## 2.2 Actions and states

There are six primitive actions in the domain: (a) four navigation actions that move the taxi one square North, South, East, or West, (b) a Pickup action, and (c) a Drop-off action. There is a reward of -1 for each action (time cost) and an additional reward of +20 for successfully delivering the passenger. There is also a reward of -10 if the taxi attempts to execute the Drop-off or Pickup actions illegally. We call these illegal actions. This means a Drop-off action when there is no passenger in the taxi, or on a wrong tile or a Pickup action on a tile where there is no passenger, or when the passenger is already in the taxi. If a navigation action would cause the taxi to hit a wall, the action is a no-op, and there is only the usual reward of -1.

We consider 2 kinds of states: a state can either be Non-Markovian and be represented by a 2-tuple (row_number, column_number) plus the drop-off/pick up memory or it can be Markovian and be represented by a 3-tuple (row_number, column_number, passenger_location) for a single game. This tuple can also be represented by a number as is explained in the next section.

## 2.3 Non-Markovian to Markovian

The environment is inherently Non-Markovian (like most environments), but is made Markovian in the program by embedding the history of each Non-Markovian state into the state itself. This way, the state does not have to memorize the actions that happened in the past (agent history) and therefore becomes Markovian. The history of a state can be for example where a passenger has been dropped-off or picked-up.

While the Non-Markovian environment only has 25 states (the taxi can be on 25 distinct tiles), the Markovian variant has $5 * 5 * 26 = 650$ states (5 rows and columns and 26 passenger positions) for a specific episode. Notice that the destination location does not change in an episode, so must not be added in the equation. When this is extended to learning over multiple episodes, the state becomes a 4-tuple (row_number, column_number, passenger_location, destination_location) and the agent has to traverse in a $650 * 4 = 2600$ dimensional state-space. This explosion of states by adding even a single history/memory variable is not desirable in Q-learning and can be drastically reduced with the introduction of a shield.[1]

## 2.4 Shielding

One of the most important, if not the most important qualities of an autonomous agent is safety. When deploying an agent, we have to make sure it doesn't choose actions that can lead to an unsafe state. These unsafe actions thus must be avoided at all times, even when an unsafe state will occur in the distant future. In other words, an unsafe action is an action that leads

[1]The state-space doesn't really get reduced, but the accessible state-space does. So when you expand to a more complex environment with too much states for Q-learning to handle, shielding is not a solution. Other kinds of algorithms like deep Q-learning are necessary to work with a very big/continuous state-space.

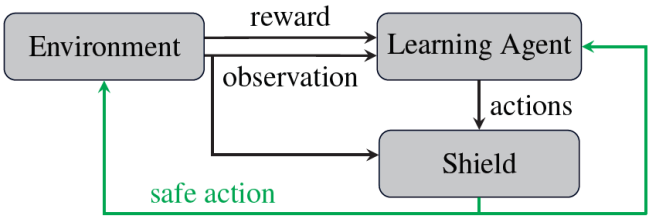


Figure 2: Post-posed shielded reinforcement learning

to a path where an unsafe state is unavoidable. These actions will be replaced with safe ones by the shield. To determine what the unsafe states are in our environment, we need safety specifications. These are rules (translated to LTL in [4]) that describe the safe state-space. From this safe state-space, the unsafe states can be derived.

**The safety specification**

To demonstrate the impact of the shield while preserving simplicity, we only choose one safety specification. Of course, more can be added. The safety specification is: "The passenger cannot be dropped-off on street tiles". We picked this specification because it filters out the states where the passenger is standing on a street tile which is (in our opinion) an unsafe state. The passenger can only be dropped-off at one of the 4 different, safe hubs (R,G,Y or B). Notice that this does not eliminate all illegal actions so the taxi can still get a penalty.

**The shield**

The shield ensures the correctness of the safety specification. It maps a set of actions onto a set of safe actions for every state. Post-posed shielding is used (as described in [3]), which means an unsafe action chosen by the agent is replaced with a safe action. This new action is then sent to the environment and fed back into the agent (figure 2) to update its Q-table for the right action.

To translate the safety specification from English to a set of unsafe states, you could use LTL and from there construct a safety automaton, which only contains safe states. Because our environment is relatively simple, we directly constructed a safety automaton. Figure 3 shows this automaton (ignore the green solution for now). This automaton contains the $5 * 5 * (26 - 21) = 125$ states (21 street tiles) for a specific episode. The state where the passenger is at the destination hub is the only accepting state. As is visible, the taxi can transition from the middle 5-by-5 grid (where the passenger is in the taxi) to one of the other 4 grids (where the passenger is in one of the hubs). This can only be done on the hubs (R,G,Y or B) which is exactly our safety specification.

The automaton would be a lot bigger if we had not implemented the safety specification (650 states instead of 125 as calculated in section 2.3). In our environment, the chosen unsafe states are also states that are never in the best solution trace (the chain of states that ends in the highest possible reward), because dropping-off and picking up the passenger on a street tile is not necessary to end the game and gives a penalty of -2. Therefore, the shielded algorithm will converge

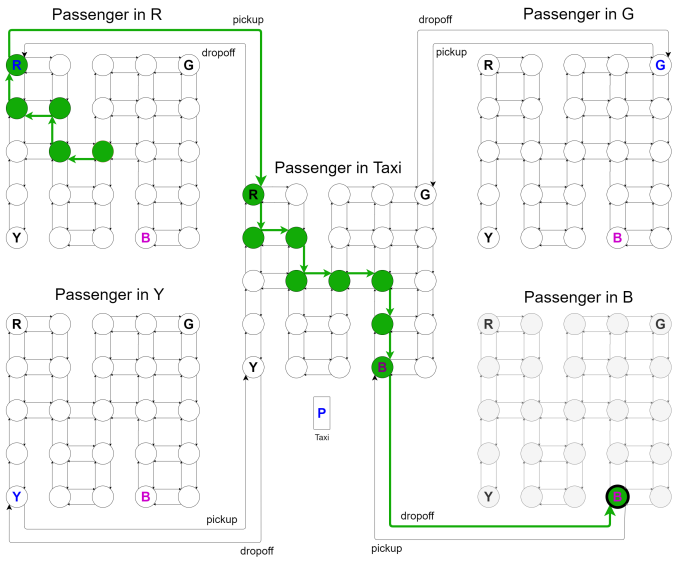


Figure 3: An example of a solution for the Taxi environment with the destination being location B. The taxi starts in the center of the grid and the passenger in location R.

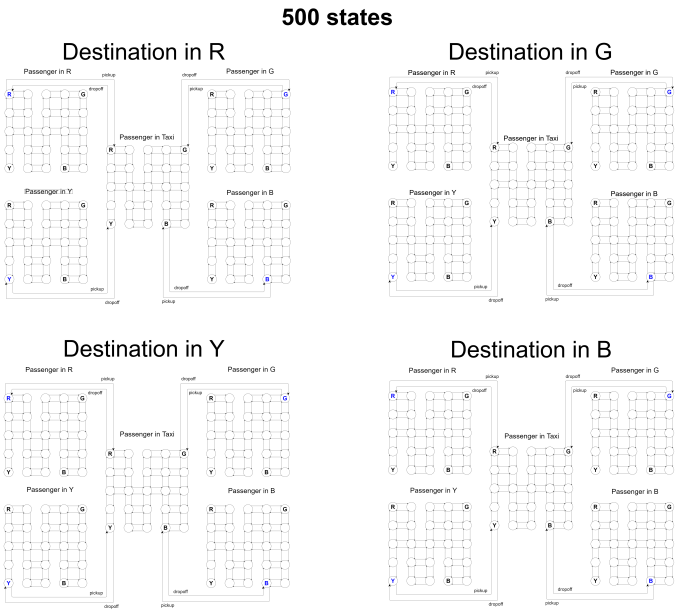


Figure 4: The 500 different states for all possible games.

quicker on average.

We have to be careful, because in general this is not always the case. The best solution trace could include an unsafe state and thus be unsafe as a whole. The shielded algorithm will then converge to a suboptimal, but safe trace (which will also be the best safe trace) which is more important.

### Implementing the shield

We did not use LTL to translate the safety specifications to a shield, but the unsafe states are determined from the safety automaton. In the shield, the unsafe state numbers are calculated from a set that contains the unsafe state 4-tuples. This set is created by taking all tuples and filtering according to the safety specification. The unsafe states are thus determined before the agent starts learning.

When the agent chooses an unsafe action while learning, the shield detects this, blocks the attempt and asks the agent to choose another action. This happens until the agent chooses a safe action.

## 2.5 Reward Shaping

There is only one positive reward that can be earned in the taxi environment. When this reward is earned, the game is solved and the episode ends. This type of problem is also called a search problem e.g. all rewards are negative and a positive reward is given when the solution is found. This sparse reward behavior will result in the agent behaving randomly until it stumbles upon the reward, after which the whole path to the reward will be enforced. By using reward shaping we can spread the reward out to improve search. Such reward transformations have the form R'(s,a,s') = R(s,a,s') + F(s,a,s'), where R is the original reward function and F is a shaping reward function. Reward shaping becomes more effective when the sequence of states required to achieve the reward is larger or when the number of possible actions in each state is larger.

A simple example can be used to illustrate this. Let's assume we have a locked door and two buttons X and Y. The problem is to find the correct sequence of 10 button presses to open the door. There are $2^{10}$ possible sequences. So if we solve this by using random sequences it would take $2^{10}/2$ tries on average to get the reward. If you get a hint that you found the first 4 buttons when you press them correctly you would need on average $2^4/2$ tries to get the hint. Then you would need on average another $2^6/2$ to find the second part of the solution. The number of tries needed with the hint is significantly smaller: $2^4/2 + 2^6/2 < 2^{10}/2$. Using more buttons (actions) or requiring a longer sequence, would make the difference even larger. The general formula for this example is

$$a^{s_1}/2 + ... + a^{s_n}/2 < a^{s_{tot}}/2$$

where a is the number of buttons (actions), $s_{tot}$ the length of the required sequence and $s_1 + ... + s_n = s_{tot}$. Here we can see that dividing the total problem in subproblems of equal size gives the best reduction. This formula does not hold exactly for more complex problems but the general rule for dividing the overall problem in smaller subproblems of equal size to apply reward shaping is a good heuristic.

In this case we divided the problem in two subproblems: (a) pick up the passenger (b) drop-off the passenger at the destination hub. When the passenger is picked up, the problem is halfway solved so half of the reward, $20/2 = 10$, can be given. To maintain optimality, this partial reward also needs to be subtracted again when the agent enters a path that can no longer reach the final solution or when the agent 'undoes' the first subproblem after gaining its partial reward. As said in the paper [1], if F(s,a,s') is chosen from a restricted class of potential-based shaping functions defined as $F(s, a, s') = \gamma\phi(s') - \phi(s)$ (for some real-valued function $\phi$), then this guarantees preservation of optimal and near-optimal policies with respect to the original unshaped MDP.

We do this by defining a potential function over every state. All starting states have potential zero. States with the passenger in the taxi get potential 10. States where the passenger is at the destination get potential 20.

$$\phi(s) = x, \text{ if passenger is at destination}$$
$$\phi(s) = x/2, \text{ if passenger is in taxi}$$
$$\phi(s) = 0, \text{ else}$$

with x = total reward (=20)

In figure 3 we see a trace of a correct solution. The states in the bottom-left, top-left an top-right are the states where the game can start, these all have $\phi(s) = 0$. In the states in the center, the passenger is in the taxi, these have $\phi(s) = 10$. In the states in bottom-left, the game is solved. Only one of these states can ever be reached because the game ends as soon as it's solved. The solved states have $\phi(s) = 20$. When the agent picks up the passenger in R, transitioning to the center, he gains half the reward $\phi(s') - \phi(s) = 10 - 0$. Then he moves to location B. When the passenger is dropped, transitioning to bottom-left, the other half of the reward is gained.

What is not mentioned in the paper is a requirement for the potential of starting states. We argue that any state that is in the initial state distribution must have $\phi(s) = 0$. This is because if the initial state has a potential higher than zero, then a part of the reward cannot be achieved anymore. We have made sure all starting states have $\phi(s) = 0$ in our solution.
Also important to note is than when a reward is fully achieved, it should not be lost again. The second remark is not an issue for episodic problems where the game finishes as soon as the reward is achieved. For environments where a reward can be achieved multiple times or the game continues after the reward is achieved, this will have to be taken in consideration.

# 3 Experiments

After implementing the shield and reward shaping we ran several experiments to compare both techniques and see the combined result. Several parameters of the learning algorithms and the environment were varied to see the effects. We also thought about introducing an exploratory factor *epsilon*, but we think this will not make a big difference in this simple environment (only slow down the convergence). Changing the final reward has no effect on the shape of the curve because it is given only once and at the end of each episode. It only shifts the curve up or down, depending on the new value.

To check our hypothesis, a comparison between the 3 different algorithms (regular Q-learning, shielded Q-learning and Q-learning with reward shaping) is necessary so we plotted the accumulated reward in function of the episodes of both shielding and reward shaping. We also want to see what happens when we combine both techniques, which we also included in our experiments. After plotting some graphs with different amounts of episodes and iterations, we saw that after about 300 episodes, all techniques found a solution so making a graph with more than this amount is unnecessary.

We also wanted to see the effect when making the penalty for illegal actions larger to compare it with a normal penalty.

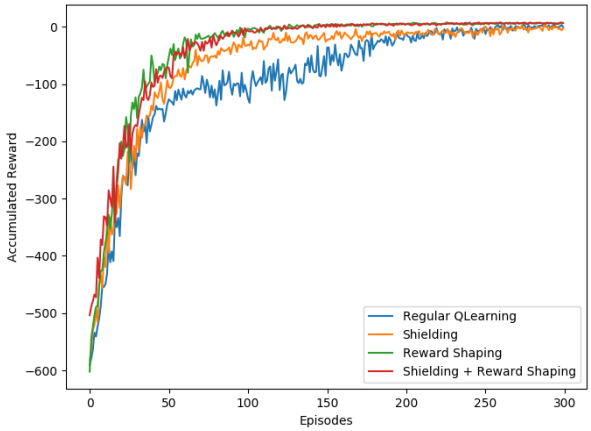


Figure 5: Comparison of all implemented techniques averaged over 50 iterations

## 3.1 Training

While training, the agent has a timeout of 200 episodes. So if the taxi keeps driving around or is stuck driving into a wall, it gets a total reward of -200 (time cost).

The time it took to run (for 50 iterations over 300 episodes each):

- Regular Q-learning: 66,41s
- Shielded Q-learning: 61,31s
- Q-learning with reward shaping: 38,19s
- Shielded Q-learning with reward shaping: 43,25s

Here you can clearly see that when adding the techniques, the time it takes to find the solution is reduced.

We timed this using `time.process_time()` (in Python). The used computer specifications are: Intel Core I3-2120 @ 3.3GHz (dual-core), 8GB RAM and 64-bit Windows 10 Pro (using the Windows Subsystem for Linux).

# 4 Results

In figure 4 the result is shown of plotting all four chosen combinations. We plot the average accumulated reward for each episode over 50 iterations.
In figure 6 we try a different negative reward for illegal actions, comparing how this affects both regular Q-learning and a shielded agent.

# 5 Discussion

In our experiments we used the following variables:

- *alpha*: learning rate of the Q-learning algorithm
- *n*: number of episodes to train on
- *t*: number of iterations to average out

**Shielding versus Reward Shaping**

In the comparison plot (figure 5), we can see that both shielding and reward shaping give a faster convergence then regular Q-learning. This is what was expected to happen. Shielded agents (Sa) will start off with a higher reward (episode 0-25). We think this is because some illegal actions already get blocked in the shielding case and this is also the region in which the Reward Shaping agent (RSa) still hasn't gotten to the partial reward yet. So the Sa won't drop-off illegally while the RSa still has a chance to do this.

Most noticeable is the section between 50 and 150 episodes. In this region it is very clear that both shielding and reward shaping outperform regular Q-learning. The regular Q-learning graph has a different shape then the Shielding or Reward Shaping graph here. We think this is because of the illegal actions that give a penalty of -10. While the regular Q-learning agent (Qa) still tries these illegal actions (until it has found a solution at around episode 200), the Sa and RSa have at this point already learned to avoid these actions.

As is visible starting from around episode 75, Reward Shaping converges to a higher reward more quickly in comparison to regular or shielded Q-learning. We think this is because the RSa finds a shorter path (to the partial reward of 10) quicker, so it gets less time penalties. It takes a lot longer for the Qa and Sa to find the final +20 reward (the path is on average double as long).

Also notice that the graphs don't converge to zero, but (eventually) a positive value. This value depends on the starting conditions of the episode (the taxi and passenger spawn randomly). We calculated that the maximum possible accumulated reward is 13, the minimum is 2.

The reward shaping curve is similar to the shielding curve in form, but seems to be shifted up by a constant factor. We think this can be explained by the -10 penalty from dropping-off or picking-up the passenger illegally. In the case of reward shaping, when the agent learns to pickup the passenger on the right location, it already gets a part of the final reward (+10). This way (before picking up the passenger), illegal actions will be eliminated because it has already learned to directly pickup the passenger instead of randomly driving around on the grid. So in contrast to the shielding technique, the reward shaping technique eliminates about half of these locations and will therefore have an improvement of a constant factor on average.

**Combination of Shielding and Reward Shaping**

Shielding by itself gives an improvement over regular Q-learning but when used on top of reward shaping it does not seem to have much impact. Even worse, the convergence speed drops a little on average. This is counter-intuitive because we would expect the two techniques with a positive influence to have at least the same positive influence as the "sum" of the two techniques.

We do however see an improvement compared to reward shaping in the early episodes (this is the shield at work).

**Variation with higher penalty**

In figure 6 we show the effect of changing the penalty for an illegal action to a higher value, -20 vs -10. As expected the higher penalty causes the graph to start lower for both normal

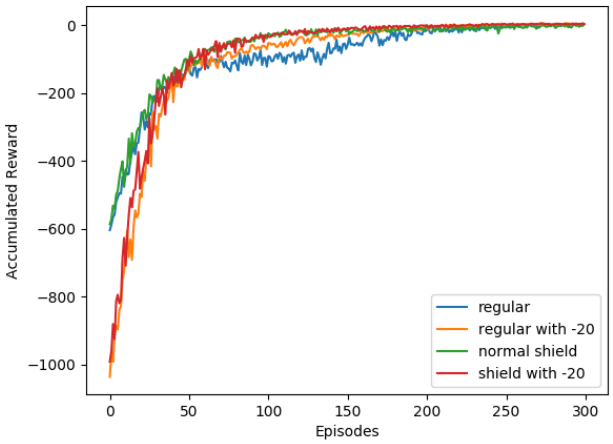


Figure 6: Small negative reward compared to larger negative reward averaged over 50 iterations.

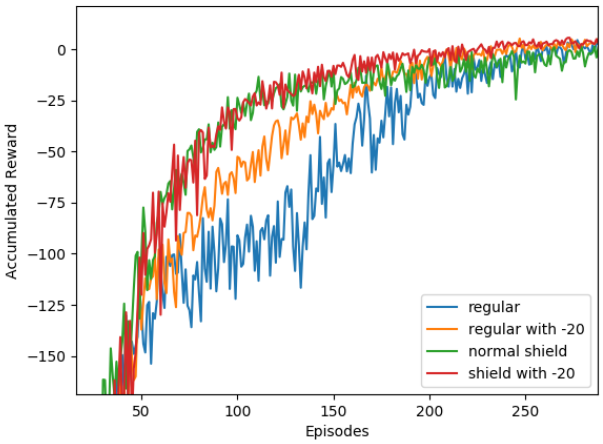


Figure 7: Small negative reward compared to larger negative reward averaged over 50 iterations.

and shielded Q-learning. Interesting is the region between 50 and 200 episodes. This is better visible in figure 7. We can see that a higher penalty causes regular Q-learning to converge notably faster in this region, while shielded Q-learning already performs almost equally good for both penalties.

## 6 Implementation

For the project we made use of the OpenAI Gym toolkit. Gym is a toolkit for developing and comparing reinforcement learning algorithms. The Gym library is a collection of test problems — environments — that can be used to work out reinforcement learning algorithms. [5] The library is written in Python. We have used Python for our part of the implementation as well. From the available environments in Gym, we chose to use the 'Taxi-v3' environment to do our experiments. All the Python scripts were run on Ubuntu 18.04 or Debian. We used the Windows Subsystem for Linux (WSL) and Xming to display graphs in Windows. The Matplotlib package was used to render graphs.

## 7 Conclusion

We implemented the shielding technique [1] and reward shaping [4] to Q-learning in an already existing environment to analyse its separate and combined resulting behaviours. In both separate cases, the learning performance improved. The reward shaping technique did better than the shielding technique in terms of accumulated reward. This is because in contrast to reward shaping, shielding is not focused on improving the learning performance, but on avoiding unsafe behaviour, which it did correctly. In the combined case, a result that lies in between the two techniques was achieved, which we found to be counter-intuitive.

## 8 Time Estimation

We mostly worked together in sessions for this project. We estimate that we spent (together) about 9 hours preparing and discussing (in semester 1), 20 hours implementing the creative part (including scrapped ideas), 15 hours for preparing and giving the presentation and 15 hours for writing this report.

This gives a total working estimation of 59 hours together +- 15 hours individually.

## References


[1] Scott Sanner Sheila A. McIlraith Alberto Camacho, Oscar Chen. Decision-making with non-markovian rewards: From ltl to automata-based reward shaping. http://www.cs.toronto.edu/~sheila/publications/cam-etal-rldm17.pdf, 2017.

[2] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition, 1999.

[3] Ruediger Ehlers Bettina Könighofer Scott Niekum Ufuk Topcu Mohammed Alshiekh, Roderick Bloem. Safe reinforcement learning via shielding. https://arxiv.org/pdf/1708.08611.pdf, 2017.

[4] Ruediger Ehlers Bettina Könighofer Scott Niekum Ufuk Topcu Mohammed Alshiekh, Roderick Bloem. Safe reinforcement learning via shielding. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211, 2018.

[5] OpenAI. Documentation. https://gym.openai.com/docs/, 2020.