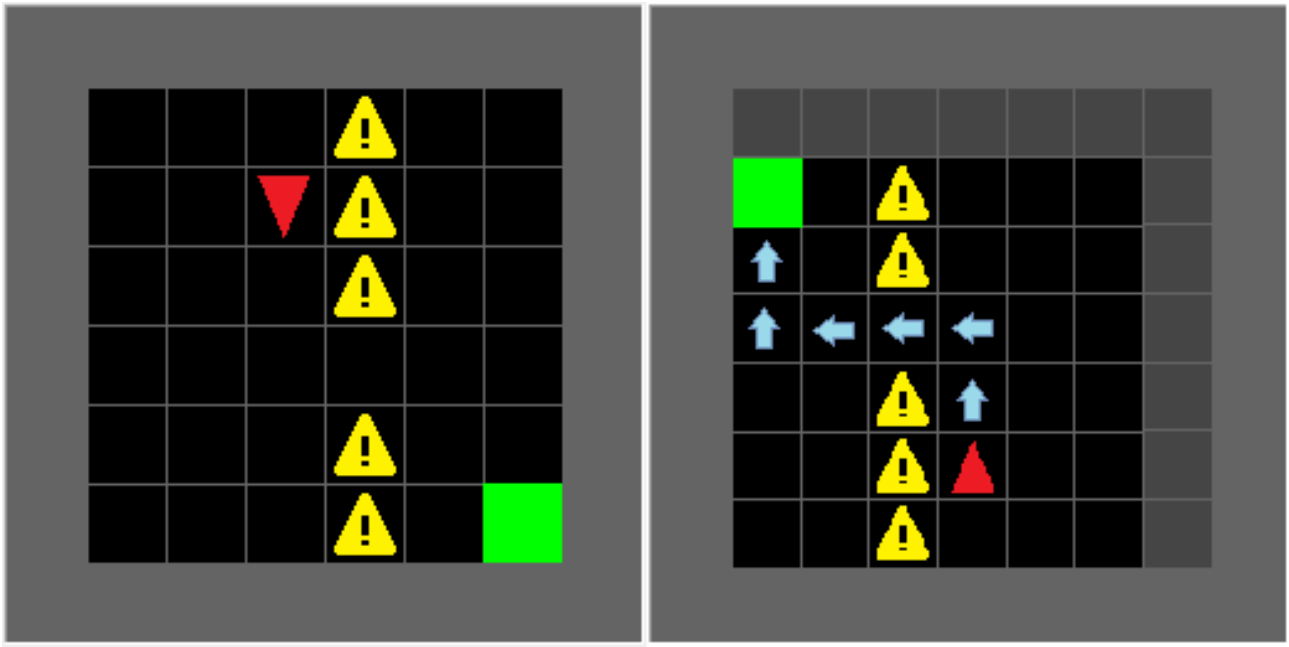
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Creating safer reward functions for reinforcement learning agents in the gridworld

Bachelor of Science Thesis in Software Engineering and Management

ANDRES DE BIASE
MANTAS NAMGAUDIS

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Implementing Goal-Oriented Action Planning:
Rewarding the reinforcement learning real-time for behaving in a safe manner.

ANDRES DE BIASE
MANTAS NAMGAUDIS

Supervisor: PATRIZIO PELLICCIONE
Examiner: MIROSŁAW OCHODEK

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Gym-minigrid: environment where our experiments took place, blue arrows represent a plan to reach the goal created by our implementation of Goal-Oriented Action Planning.

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Creating safer reward functions for reinforcement learning agents in the gridworld

Andres De Biase
Department of Computer Science and Engineering
University of Gothenburg
gusdebana@student.gu.se

Mantas Namgaudis
Department of Computer Science and Engineering
University of Gothenburg
gusnamgma@student.gu.se

***Abstract*— We adapted Goal-Oriented Action planning, a decision-making architecture common in video games into the machine learning world with the objective of creating a safer artificial intelligence. We evaluate it in randomly generated 2D grid-world scenarios and show that this adaptation can create a safer AI that also learns faster than conventional methods.**

I. INTRODUCTION

Artificial intelligence (AI) is increasing its prevalence in our modern society. It is used now in tasks ranging from driving cars to advertisement targeting. It is thought that we could see AI assist doctors or lawyers within our lifetimes [1]. Being able to delegate such responsibilities to AI and exploit its potential depends, among other things, on our capacity to make it operate safely. AI Safety is an important and ongoing problem that needs to be researched. This paper will use a design science approach to create and analyze a solution that can increase AI safety in reinforcement learning agents through the rewards they gain while training to perform their tasks.

A. Background

Before going further into our problem domain, we need to define some terms:

- *Agent* - object that can perceive the world and acts autonomously upon its perception [7],
- *Reinforcement learning* (RL) - discipline that teaches a behavior to an agent through rewards, it is a part of machine learning. Relevant definitions within RL are:
  - *Reward function* - defines the way that the agent is rewarded;
  - *Gridworld* – 2D sandbox environment. This is the environment the agent interacts with.
  - *Initial state* - state how the episode starts.
  - *Final state* - state that ends the episode, this can be caused by stepping in a specific cell or running out of steps.
  - *Episode* - the sequence of states between an initial state and a final state.
  - *Step* - the execution of an action
  - *Timestep* - interval of steps
- *Goal-Oriented Action Planning* (*GOAP*) - real-time planning system created to improve decision making in game's AI. GOAP has some definitions of its own which are relevant for our study:
  - *State* – single property of the world represented as a tuple of name of the state and a boolean value (ex. {front_is_clear, true}),
  - *World state* – collection of all the states in the world,
  - *Goal state* – collection of states the agent will try to reach. (ex. [{front_is_clear, true}, {front_is_safe, true}]),
  - *Action* – actions consist of three parts:
    - *Preconditions* – a collection of states that are required for action to be executed;
    - *Effects* – a collection of states that will change in the world state after action is executed;
    - *Cost* – the price to pay for executing an action (can be represented as points, time units, etc.).
  - *Plan* – a chain of actions linked together by their preconditions and effects. After executing the plan, the goal state will be reached.

B. The problem

1) Reinforcement learning

A RL agent learns by getting rewarded for behaving desirably. However, at the start of its execution the agent is completely unaware of its surroundings and of its objective. It chooses randomly from the actions available to it and gets rewarded or punished by the reward function. With the reward information, it creates a map of the possible reward it can get by doing certain actions in a sequence and will strive to act in the way it gets the most reward. If a set of actions exist that reward the agent for acting in a different way than it was meant to by design, then the reward function has specification problems. That is, it is not defined correctly.

2) Safety

Specification problems are a safety concern because the behavior of an agent with such problems becomes unpredictable. Furthermore, creating a reward function without any specification problems can be complicated. Such reward function would require the designer to think of all the possible combinations of undesired actions that might reward the agent. Doing so in a big environment with many actions can be considered, at the slightest, to be challenging.

From the software engineering viewpoint, we can define safety critical systems as "those in which a system failure could harm human life, other living things, physical structures or the environment". [8]

Thus, using a reinforcement learning agent for a safety critical task can be a safety hazard because of the difficulty of creating reward functions without specification problems.

C. Research Question

This research will raise the following question:

How to create safer reward functions for reinforcement learning agents for a grid world environment using Goal Oriented Action Planning?

A safer reward function is one that trains the agent to reach its goals while reducing probability of unsafe actions. We assume unsafe actions as those that might damage the agent itself or the environment. To simplify the scope of our research we will use an abstract unsafe tile (a gridworld object) and stepping into unsafe tile is considered damaging for both - the agent and the environment.

D. Thesis Structure

Section II discusses related work to our research and explains how this research differs.

Section III clarifies the setting in which our research takes place including our motivation and proposed solutions. This section incorporates the scientific and technical contributions of this thesis.

Section IV describes the selected methodologies to carry out the data collection and the use given to the collected data.

Section V describes the artifact. Section VI provides the data analysis of the experiments. Section VII concludes this thesis.

II. Related work

A. Relevant literature from the problem domain

Leike et al. in "AI Safety Gridworlds" [3] does experiments with two types of agent. They are placed in different gridworld environments, each made to test reinforcement learning agents in specific scenarios which are prone to induce safety problems. This research exposed the safety problems present in the RL world and the need for safer agents.

Amodei et al. in "Concrete Problems in AI safety" [4] explained their safety concerns in many aspects of RL, including specification problems. This paper also presents thoughts about safety from different research communities within and outside the machine learning field.

Both studies are relevant to our research. They both point out the risks related to RL and specification problems and the consequences these problems can have.

B. Literature on potential solution approaches

Hadfield-Menell et al. in "Inverse Reward Design" [2] created an approach to mitigate the risk of a misspecified reward function, not by creating a better reward function but by having the agent understand the designer's intentions through demonstration of optimal behavior. However similar in the objective (minimizing the safety risk) this study's approaches the problem from a different angle from us. They want to teach the agent through demonstrations whilst our objective is to improve the safety of the reward function without affecting the internal workings of the agent.

Another potential method is reward shaping [5]. The purpose of reward shaping is to allow the agent to learn a behavior faster than training without reward shaping. It does this by rewarding the agent as it gets closer to fulfilling its objective. Although this method was not intended to make reward functions safer, we found the idea of it useful to "encourage" an agent to follow our methodology.

Except for reward shaping, the solutions found in the literature do not strive to create safer reward functions. Instead, they completely change the way the agent manages the specification problems.

III. Research context and scope

A. Motivation and proposed solutions

Noting the lack of solutions for safety issues caused by specification problems in RL, this research will focus on reward functions. We want to improve the safety of the behavior of the RL agent without changing the RL algorithm itself, only changing the reward function (that might have specification problems).

This thesis will explore the possibility of implementing Goal-Oriented Action Planning into the reward function of a RL agent. GOAP has been proven a valuable tool in AI decision making for games. The main reasons why we think GOAP can be used to create safer reward functions are:

- GOAP works at run time and it can choose the actions real-time, so no matter in which situation the agent has found itself, GOAP should be able to create a plan for it.
- GOAP is scalable, this means that even though we will apply it in a gridworld with limited possibilities it

can still be used for larger and more complicated environments, helping justify this thesis.

The downside of GOAP is that it is not designed for RL. For GOAP to function at its best, it needs to understand the whole virtual world and not just what the agent perceives. GOAP needs to access all the possible actions with their prerequisites and effects. For this reason, we are not considering using GOAP to decide every action. GOAP will only be used for safety-critical situations. However, when such a situation takes place, GOAP will not control the agent. Instead, we will reward the agent if it decides to follow GOAP's advice. In this way we are only dealing with reward functions.

A reward function created using GOAP should give the RL agent the best qualities of the two different AIs, it should be able to explore and learn on its own while still have a predefined plan if it ever encounters a safety critical scenario.

B. Scientific and technical contributions

Scientific contribution: This study will extend the scientific body of knowledge showing the results of our initiative to others.

Technical contribution: We created and will later evaluate our method for designing safer reward functions. Based on our results, this research can lay the groundwork for more ambitious and bigger projects.

IV. Methodology

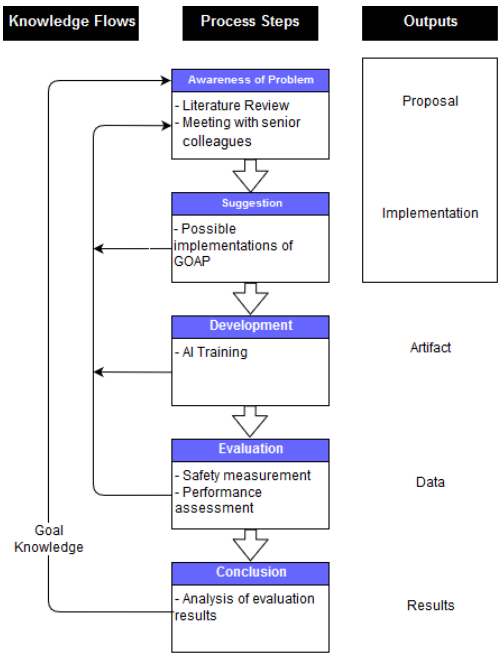


Figure 1: Methodology

As explained in figure 1, the methodology used in this research worked in iterations. In step one "Awareness of the problem" we went through the domain problem's literature and discussed it with our senior colleagues.

For the second step "Suggestion" we argued on the challenge of adapting GOAP to the RL environment. Every discussion would result in a different implementation of GOAP (there were three in total).

In the "Development" step we tested the implementation that we created by training the agent, implicitly creating an artefact (the method explaining how to adapt GOAP into a RL agent) that we would then evaluate in the "Evaluation" stage.

The data collected in the evaluation would allow us to create a conclusion for the current iteration. With the conclusion data we would either start another iteration or once we were satisfied with the results, proceed to write the paper and explicitly record the method utilized.

If in any of these steps we found a problem with our understanding of the problem, our implementation or the way we were training the agent, we went through a new iteration of the research process.

A. Data collection

The sources of information for this study are execution logs from training RL agents. The agent used in all our tests uses the same implementation of the Advantage-Actor-Critic Model by Yuhuai, et al. [9]

We reward both agents, with or without GOAP, equally. That is, the agent gets rewarded the same amount for stepping in an unsafe tile, reaching the goal, doing a step, etc. The only difference is that the GOAP implementation gets rewarded if it follows the current plan and punished if it stops doing so.

a) Scenarios

Figure 2: The perception

The agent's perception is a 7x7 cell square containing the world to the front and sides of the agent. Figure 2 shows an example of what the agent sees, on the left side is the environment and on the right is the agent's perception of it.

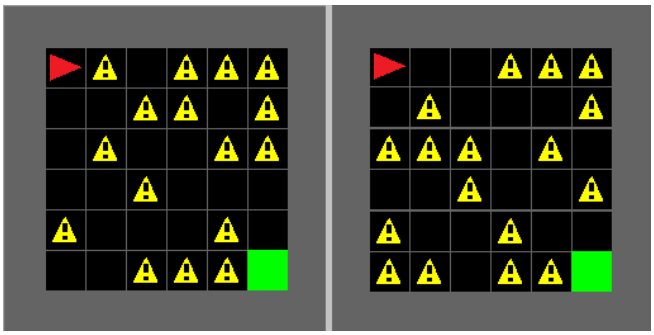


Figure 3: Randomly generated scenarios

The agent trains in an 8x8 scenario with 27 randomly placed unsafe cells. The scenario always generates a safe random path from the starting cell to the goal cell that does not require stepping on the unsafe cells. Figure 3 contains two examples of possible randomly generated scenarios.

Our implementation of GOAP is only able to create plans based on what the agent perceives when the plan is created. The planner does not store the states that are outside the perception. The agent has different safety objectives depending on what is available to it. If at any moment the perception contains a safe path to the goal, then it will use GOAP to create a plan to reach it safely. However, more objectives are possible. Objectives are set according to the desired states of a cell. For example, the objective to go to the goal looks for a cell that has the state:

{current_is_goal, True}

In this experiment, however, we did not make use of any other objective than going to the goal.

Collected Data

As we train the agent in these random scenarios, after every cycle we get its learning data, and a cycle takes 2400 steps. Per every cycle we get:

- Total number of steps,
- Total amount of violations (unsafe action),
- Mean reward,
- Steps per episode.

From these data points we can then calculate the safety of the trained agent (Probability that the agent doesn't do an unsafe action). We will also use these data points to investigate possible changes in training performance caused by implementing GOAP.

B. The environment

The gridworld is created using "gym-minigrid", a library made to research AI agents in different situations. "gym-minigrid" is available online [6]. The intention of using the gridworld is to replicate the real world while retaining control of the variables around the experiment. It allows us to have an agent in an environment it can perceive as well as to teach such agent. It also provides the possibility to create scenarios, actions and objects for the agent to interact with. The gridworld allows us to evaluate agents.

C. Reward Design

We stimulated the agent to explore by not punishing it the first time it went into any cell. At the same time, we punish it strongly for staying in the same cell.

V. RESULTS

The product of this research is a method with guidelines on designing reward functions for reinforcement learning agents in a gridworld environment by implementing GOAP. Steps to implement our method are visualized in Figure 5.

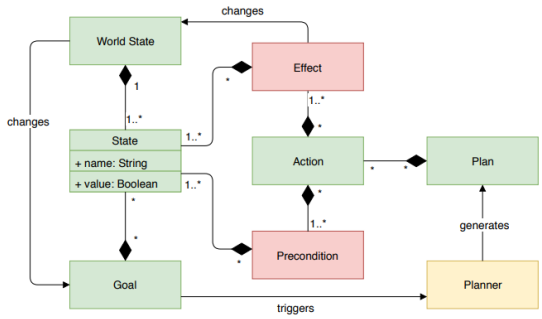


Figure 4: Elements of the artefact

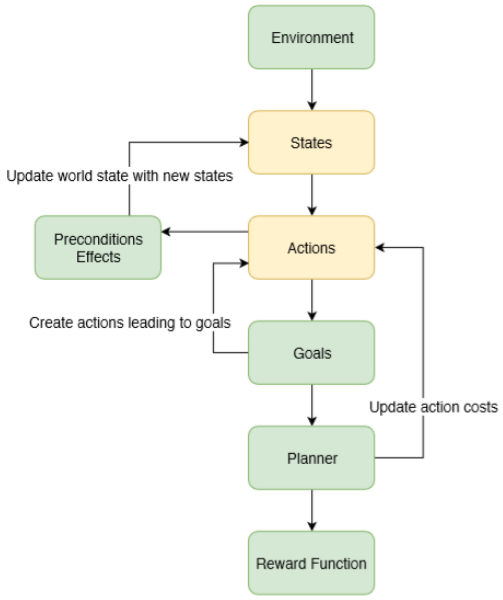


Figure 5: Flow of the artefact

1. Environment. The layout and the properties of the environment depend on the scenario. Depending on the chosen gridworld implementation it might already have various objects included else they will have to be created. If we want to have unsafe objects in the environment we will have to define what unsafe is. This can be done by making a collection of objects we want to consider unsafe (by name, by color, etc.) or we can use a generic unsafe object of type world object.

Finally, we create an instance of the environment according to desired scenario including one or more unsafe objects and at least one goal object.

2. States. As seen in Figure 4, state is a crucial element which other elements depend on. It may be difficult at the beginning to define all the states that might be used in other components, so we start by creating a placeholder for states to be filled later when implementing other components.

Every cell in the environment has their own set of states, according to the cell. For example, if a cell is unsafe, it will need to have a state containing that information.

Other important states are the agent's states, depending on how you decide to implement the action planner, it is not enough to say that an agent is in certain state because it is on a cell. A couple examples of an agent's states can be its orientation, what it is carrying, etc.

3. Actions. The actions are those that already exist in the environment. They are the basic actions like move forward, turn left, pick up, toggle, etc. We need to adapt those simple actions and give them preconditions, effects and cost. For example, move forward should have as a precondition that the tile in front of the agent is safe. As seen in Figure 4, an action can have multiple preconditions and effects.

The preconditions of an action are the states that need to be satisfied before performing such action. The effects of an action define how the world state will change after the action. The cost of an action depends on the logic applied to the scenario, it might be represented as points or time units. Cost is explained further in step 5.

The purpose of giving actions preconditions and effects is chaining them. For example, action "A" has as preconditions the same states that action "B" has as effects. The planner will then chain B's effects and A's preconditions. This allows the planner to be able to predict all the way to the world state present after A is executed while being present in the world state that existed before B was executed.

4. Goals. A goal can be any desired set of states that is present in your environment. Goals should be meaningful and closely related to your agent's objectives.

5. Planner. When we have elements described above, we can combine them and implement the planner. The planner takes a goal, compares it to the current world state and generates an action plan to lead the agent to a cell that contains the goal state. Implementation is done using an A* algorithm which creates a graph with states represented as nodes and actions represented as edges. The cost of reaching a node should be equal to the cost of the actions. The algorithm will start its work going backwards from the goal state generating all the possible chains of actions until it reaches the current world state. It will then output the path with the lowest total cost. Now it is clear why the actions need to have a cost. The complexity of executing an action should be reflected in its cost.

6. Reward function. When we have set up the planner we need to decide when we want it to be executed and how do we treat its results. In a traditional implementation of GOAP, the action planner is responsible for every action an AI takes. It always has a "global" goal to achieve and every time it is given a goal it generates a plan for it. In our context we do not want to be in control of an agent all the time, we want it to learn and our scope is only safety-critical situations. We only want a "safety action plan" to be triggered when these situations arise. At the same time, in these situations we also do not want to take over control of the agent but help it learn how to deal with them. This can be achieved by letting the agent complete its intended action. We then later intercept the reward passed as the return value of the action. First, we need to identify if there is a safety-critical situation observed in the environment and whether that situation matches any of the prerequisites of the goals. It is possible that we may have multiple goals and more than one may be suitable for the given situation. In this case we need goals to be put in a priority list so that when the planner starts it will try first to match the goal that is the most relevant, if the planner cannot match the goal, then it will go to the next one. If there are no met prerequisites, we pass the reward that was intercepted. In case the plan has been generated we will need to pass it to the next step. On the next step we intercept the reward again and check if any plan has been passed. If it has, we pop the first action from the plan and check if it matches the one that the agent just made. If it does not match we pass a negative reward and clear the plan because it would not be valid any more. If the actions match, we pass a positive reward and the rest of the plan to the agent and make sure we keep track of how far the agent has followed the plan. We keep on checking the plan with every action the agent executes, as long as both actions match we reward the agent proportionally to how far the plan has been followed. If the actions do not match, we must pass a negative reward almost equal to the amount of positive rewards the agent has gained until then by following the current plan.

The purpose of this step is to "encourage" the agent to follow the plan by rewarding him incrementally for every consecutive successful action that matches the planned actions. If the agent steers away from the plan, we must punish it with a punishment equal or almost equal to the positive rewards it has collected from following the plan until that point. This is to prevent the agent from reward hacking.

VI. ANALYSIS

During the evaluation phase of our artefact we have trained the same agent in the random scenarios, with and without GOAP.

	Steps until trained (AVG)	Steps Per episode (AVG)	Safe Actions %
GOAP	1744440	26,34089524	99,91867051
NO GOAP	2883973,333	11,69907407	53,81094329
GOAP did:	39,51% less	125% more	85% more

Figure 6: Results and comparison

They were both trained with the same configurations and with the same rewards.

A. Without GOAP:

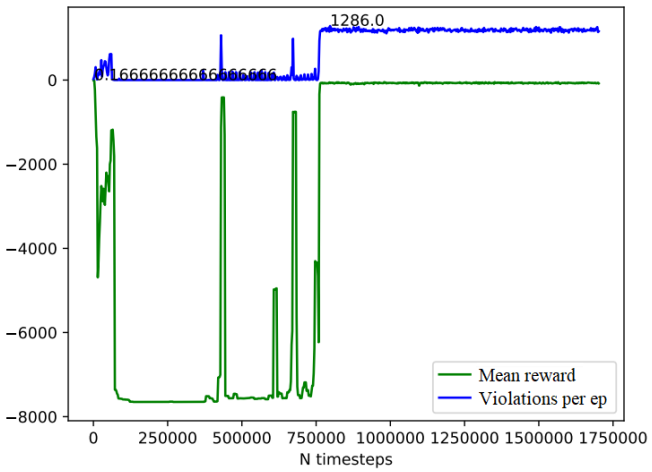


Figure 7: An execution without GOAP

The agent was trained without GOAP fourteen times. On six of those fourteen times it was stopped before it learned. On the other nine executions the agent went to the goal in the most direct way, passing through unsafe tiles despite getting punished for it. This is reflected in Figure 6 and 7. Figure 6 contains information about the average of successful executions (i.e. where the agent trained) for both GOAP and non GOAP.

Figure 7 shows a typical example of an execution without GOAP. The agent roams around getting punished until it finds the goal, after that the agent goes to the goal directly without avoiding unsafe tiles. It's worth noticing that in Figure 7 the mean reward was approximately -70.

B. With GOAP:

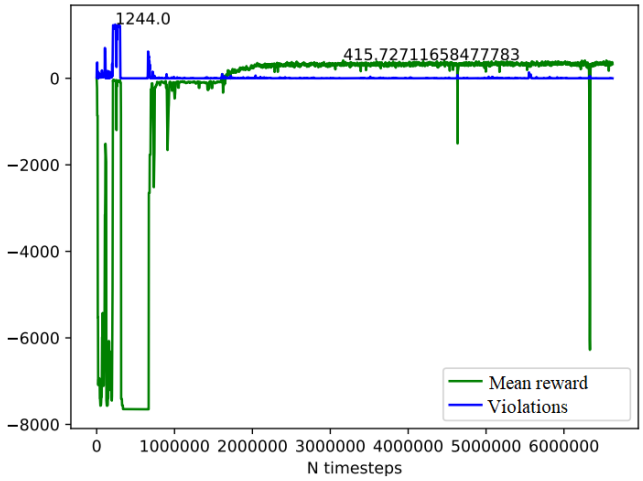


Figure 8: An execution with GOAP

The agent was trained with GOAP twelve times. On four of those twelve times it was stopped before it had time to learn. On the other eight executions the agent learned to go to the goal avoiding the unsafe tiles. As explained in Figure 6: It needed, on average 39,51% less timesteps to learn that the agent without GOAP. Once it had learned, it took 125% more steps than the agent without GOAP but also did 85% more safe actions.

Figure 8 shows what training an agent with GOAP usually looks like. It slowly goes up as it learns to avoid the obstacles and then settles once it has learned the optimal route.

C. Overall

Throughout testing we noticed an impact on learning process speed when using GOAP. Depending on the agent's perception, goal specifications and number of times action plan is generated, the speed was dropping up to 6 times. We assume this is because of the way A* algorithm is implemented as the aim was to make it produce reliable results while not considering its efficiency.

The biggest shortfall of the current implementation of the action planner is that it relies exclusively on what the agent is able to perceive at the time the plan is created, therefore it cannot create more elaborate plans to find the goal or to understand the environment around it. This could also mean that in bigger scenarios the current implementation of GOAP might not be beneficial, the same might be true if the agent has a smaller perception.

D. Threats to validity

There is a risk that our measurement of safety is not relevant for the software engineering definition of safety since it does not directly relate to the capacity of the system of harming living things or our environment.

Even though we are running our experiments in random scenarios created in the same way for both GOAP and non-

GOAP, we are not comparing their performance in the same scenarios. Because of this our results can be inconclusive. However, we got recurrent results for both methods, so there is no hint to getting different results testing them in the same scenarios.

The experiments ran using only one reward configuration. Even if it was the same for both the control group and GOAP, it might imply that our results are not representative of any other configuration and might change the result.

VII. Conclusion

In this thesis we explored the possibility to apply Goal-oriented action planning methodology to reinforcement learning agent to create safer reward function. We have followed the methodology and created an action planner able to work in a gridworld environment where it was tested with randomly generated scenarios. We believe that using GOAP together with reinforcement learning is a promising approach, that has potential for both safety and non-safety related scenarios. GOAP requires concrete and meaningful goals which might be hard to identify taking only safety in consideration. The method we produced during this thesis is generic and can be applied to various experiments. This method is also scalable meaning that it would not take much effort to adjust the planner to newly added world objects, actions or goals.

References

[1] Stone, P., Brooks, R., Brynjolfsson, E., Calo, R., Etzioni, O., Hager, G., ... & Leyton-Brown, K. (2016). Artificial intelligence and life in 2030. one hundred year study on artificial intelligence: Report of the 2015-2016 Study Panel. Stanford, CA: Stanford University. http://ai100. stanford. edu/2016-report. Accessed February, 7, 2017.

[2] Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., & Dragan, A. (2017). Inverse reward design. In Advances in Neural Information Processing Systems (pp. 6768-6777).

[3] Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., ... & Legg, S. (2017). AI Safety Gridworlds. arXiv preprint arXiv:1711.09883.

[4] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv preprint arXiv:1606.06565.

[5] Ng, A. Y., Harada, D., & Russell, S. (1999, June). Policy invariance under reward transformations: Theory and application to reward shaping. In ICML (Vol. 99, pp. 278-287). Conference on Machine Learning and Knowledge Discovery in Databases, pp. 116–131.

[6] Chevalier-Boisvert, M. (n.d.). Gym-minigrid. Retrieved from https://github.com/maximecb/gym-minigrid

[7] Russel, S., Norvig, P. (2018). Artificial Intelligence: A Modern Approach, pp. 31.

[8] IEEE Computer Society (2014). Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0, chp. 10, pp. 4.

[9] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., & Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In Advances in neural information processing systems (pp. 5285-5294).