Learning team strategy in a complex POMDP

Fedde Burgers
fedde@science.uva.nl

Morris Franken
morrisef@science.uva.nl

Jeroen Kools
jkools@science.uva.nl

***Abstract*—Designing a multi-agent system that can successfully learn and plan in a partially observable environment is a challenging task. It demands a sufficient solution to many subproblems for which no optimal solution is known, or such methods do exist but would take an infeasible amount of time for large stochastic environments. We therefore present a system with a hybrid architecture, using several learning components in addition to a considerable chunk of stationary policy. For evaluation, our team of agents is pitted against competing teams and older versions of itself in a game of Domination. The final test consisted of a round-robin tournament between 7 teams, in which our system managed to become the overall champion.**

I. Introduction

In this paper, we examine a multi-agent system designed for playing *Domination* on one particular map. *Domination* [1] is a finite horizon strategic game where two teams of agents compete for possession of a number of control points. A control point is captured when an agent moves over it. Points counting towards victory are awarded to the owners of each control point on every time step. Agents are able to fully share their knowledge of the partially observable environment with their teammates, and they can collect ammunition to engage in combat with the enemy. The game is further complicated by the existence of walls, which are obstacles to movement and shooting (but not to observation). The game state changes mostly deterministically, with the exception of ammo packs, which appear according to a stochastic distribution, which is fixed but unknown. In every time step, each agent can rotate and then move forward, both within a certain range. If they have ammo, they have the option to fire straight ahead. Within the range of the weapon, any agent that is hit is killed. Dead agents subsequently reappear at their team's spawnpoint.

In this paper a description is given of an approach for a brain that is able to get very good results in the *Domination* game. In our approach we attempted to keep the brain as simple but effective as possible. In our opinion a team can dominate his opponent by selecting the correct role, learn what the weakness or strength is of his opponent and of hisself and exploit these strength and weakness. In order to accomplish this we came up with some hard-coded roles and strategies, which can be switched and called on different moments. Our belief in adjusting to the opponent and exploiting the brain's own strength and his oppenent weakness, online learning is prefered. Offline learning is done minimally, as opponent brains were not available for training and we did not want our brain to only learn to conquer hisself.

The rest of this paper is organized as followes. In section II we will briefly review the literature for multi-agent POMPDs and other fields relevant to our application. Next, section III will deal with the finer details of the *Domination* simulation, as well as the adaptive and handcrafted strategies employed by our agents. Our system is put to the test in section IV with experiments evaluating several versions and components, and the results are reported in section V. Section V will summarize our findings and outline our ideas for further work in this field.

II. Background and Related Work

By formulating any discrete time stochastic decision problem as a Markov Decision Process, it is possible to find a optimal solutions with a number of very general algorithms, only requiring the usually reasonable Markov assumption. Since the earliest research into MDPs in the 1950s, a number of algorithms has been developed to solve MDPs. For single-agent problems with full knowledge of the current state and transitions, solutions include dynamic programming, value iteration, policy iteration as well as many variants thereof [2].

In problems with partial observability, where the current state might not be fully known, optimal solutions are harder to obtain. By adding beliefs concerning the true current state, POMDPs extend the MDP framework to deal with this kind of uncertainty. However, finding an optimal policy for a POMDP comes at a considerably higher price. Whereas normal MDPs are solvable in polynomial time, POMDPs are harder than NP [3]. If we consider multi-agent POMDPs, complexity is further increased.

For this reason, approximative learning techniques are the preferred and most viable approach for POMDPs. Since the late 1990s, there has been extensive research into multi-agent learning techniques [4] [5]. The two main branches of multi-agent learning are supervised learning and reinforcement learning. In the former, expert feedback is given on every action, which can then be generalized to a policy for every situation. Backpropagating neural networks are an example of a method that can be used for supervised multi-agent learning. Reinforcement learning on the other hand gives much sparser feedback; reward is only given for selected states that actually achieve or fail to achieve goals, which leaves the task of evaluating intermediate states completely to the agent. Because this approach does not depend so much on expert knowledge, it can be applied quicker and easier on many complex problems.

In multi-agent systems with multiple learners, the environment is nonstationary, which means that it might not be sufficient to train a lot before commencing the task (offline learning) but that online adaptation is required. Because the optimal strategy might change, it is harder to converge to a

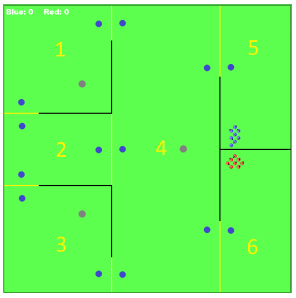


Fig. 1. Map with areas and waypoints

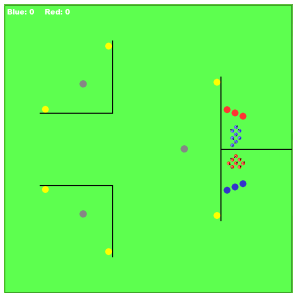


Fig. 2. Map with sniper- and blockpositions

good strategy than it is in single agent systems. Scalability of learning algorithms is also an important issue. This motivates our approach to the *Domination* domain, subdividing the problem in tasks, like navigation and aiming, in which the desired behavior is obvious, and tasks that are more complex and opaque, such as role assignment or choosing between offensive and defensive action. The tasks belonging to the first category were hardwired with a sophisticated handcrafted policy, while the others are optimized online by means of reinforcement learning. This results in a best-of-both-worlds system: the robust reliability of expert knowledge and reinforcement learning to find hidden advantages and adapt to new enemies.

III. Methodology

In this section we introduce our methodology. As has been mentioned in section I we intended to keep the brain as simple, but effective as possible. In the following subsections this simple but effective brain is described.

A. Path Planning

As the map is fixed, the brain did not have to learn it. To let the agents move around in this world we used a simple method; we divided the field in 6 different convex areas. An agent can move freely within each area space, and can move to another area through a set of waypoints. Once an agent wants to move to another area, it will first check whether or not it can move to its goal without hitting the walls. if so it will ignore the waypoints. In figure 1 the map is showed with different areas. The yellow stripes represent the boundaries between the areas and the blue dots are the waypoints.

One of the problems we ran into was the collision between 2 agents. This is solved by detecting if an agent is going to collide with another agent. If this still doesn't prevent collision, the agent quickly notice it got stuck, and will try to escape from it's current position.

B. Roles

Our strategy consists of a set of different roles for each agent:

- Blocking (forming a wall of agents right for the enemy spawn point)
- Guarding (sitting on control point)
- Patrolling (circling around a control point)
- Sniping

1) Blocking: In the first few tournaments, the blocking strategy was highly effective, as none of the enemy agents where able to move around them. but in time, the others managed to detect blockers and where not hindered by them (see figure 2). This brings us to our first decision problem, blocking against an enemy which isn't capable of dealing with blockers results in an easy win. However, blocking against an enemy which can easily manoeuvre around them will actually put us in a disadvantage, as the blocking agents are not capable of doing anything else than just sitting still, leaving the other agents outnumbered. Therefore we implemented a system which learns whether or not blocking is successful against the enemy. This is done by keeping a score for the performance of these blockers:

$$score_{block} = seenagents_{enemyteam} - seenagents_{ownteam} \quad (1)$$

After a number of iterations has passed the average score must exceed 0.5 to keep blocking the enemy. When the average score drops below 0.5 the agents will stop blocking immediately and will take another role.

2) Guarding: Guarding is the most simple role there is, its only objective is to stay on the control point. However, it might also decide to start sniping, which is explained later on.

3) Patrolling: Patrolling agents will patrol around a certain control point, searching for ammo and enemies. It will occasionally switch role with the current guard of a nearby control point if it has more ammo, or when the current guard has been shot dead. Agents which are currently patrolling can also start sniping if they have sufficient ammo.

4) Sniping: Whenever a guard- or patrolling agent has enough ammo, it will start sniping, or camping behind a wall. This means it will go to one of the pre-set sniper positions, and shoot all enemies which come along. It will remain in this position until it either runs out of ammo, or when the nearest control point is taken (or when the agent is shot, of course). The beauty of this strategy is, that these snipers were very hard to kill because they use the wall as a cover, and the enemy is not able to turn quickly enough to shoot back. This results in highly efficient ammo usage.

In order to beat these snipers we implemented an anti-sniper procedure. This was done by checking for snipers, if an agent

has found a sniper, it will try to move around it, in such a way that it becomes out of sight for the sniper. At the end it will appear behind the sniper, and is able to shoot him without the sniper shooting back.

As can be seen in figure 2, at each of the two control points at the left, there are 2 sniper positions. If only one sniper is available, this agent should learn which of the positions is the best. Therefore a score is updated each agent's turn as given by algorithm 1 . This score is communicated between the agents and the sniper position with the highest normalized score is chosen first.

Algorithm 1 Update the score for each of the 4 left sniperpositions

```
1:  procedure UPDATESNIPERPOSSCORES
2:      for all sniperpositions do
3:          if agent is near sniperposition (< 60) then
4:              Normalizationfactor = Normalizationfactor + 1
5:              Score = Score + observed enemy agents
6:              if agent shoots and on sniperposition then
7:                  Score = Score + 10
8:              else if Agent dies then
9:                  if Agent is on sniperposition then
10:                     Score = Score - 10
11:                 else
12:                     Score = Score - 5
13:                 end if
14:             end if
15:             NormScore = Score / Normalizationfactor
16:         end if
17:     end for
18: end procedure
```

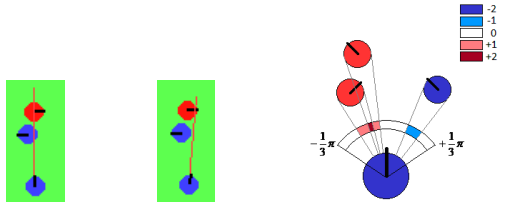
C. Priority list

While each agent had a certain role to fulfill, they all have to respond to events which are more important than performing there role-oriented action. We therefore created a priority list which each agent has to obey. The priority is given by algorithm 2.

Algorithm 2 Priority

```
1:  if agent has ammo and enemy is near then
2:      kill_enemy
3:  else if got stuck then
4:      escape
5:  else if our score is 20% higher than opponent score then
6:      initialize special strategy 'deFuik'
7:  else if agent has ammo and enemy control point is near then
8:      takePointWithAmmo
9:  else if ammo is near then
10:     takeAmmo
11: else if agent has no ammo and enemy point is near then
12:     takePointNoAmmo
13: else
14:     doDefault
15: end if
```

Kill_enemy: In this function, the agent will move towards the enemy, and will shoot when the distance is smaller than the maximum shot distance. There are some conditions where this is not as simple as it seems. For instance, this system will shoot his team mate without a problem if it gets in the way (see figure 3a. We therefore implemented a function which computes the best angle to fire upon an enemy. This is done by creating a table, where each element represents the score for shooting in a particular direction, (see figure 3c



(a) Friendly fire

(b) Non-friendly fire

(c) Score the different angles

Fig. 3. Smart chosing shooting angle

This system will prevent friendly fire as much as possible and will also try to shoot multiple enemies if possible. To improve our shooting even more, we gave different rewards to each agent. For example, we gave a higher reward for enemies which are faced towards us (possible danger), or agents which are on a control point. We also increased the rewards for enemies which we suspect to have ammo. And we increase the penalty for shooting friendly agents which have ammo. Predicting if an enemy agent has ammo is done by keeping a list of possible dangerous enemies, each time an agent dies, it will backtrack the last observation, and subtract the possible (subject, killer, dader).

Another decision problem is when there are multiple enemies around, but none of them are within a shooting range. Which enemy should be followed? We solved this by ranking the enemies according the amount of steps it would take to shoot this enemy and the direction is was facing. Agents facing away from us will receive a lower ranking than agents which are faced towards us.

escape: The escape function will guide the agent out of its current position when it got stuck.

deFuik: This function is part of a special tactic where all agents work together to push the enemy back into there base. It is executed in 2 steps:

1) send out 3 blockers to slow down the enemy. These blockers will keep updating their score
2) when the blockers maintain an average score of 0.8 or more, 50 iterations after they where initialized, they will notify the other agents to come and position themselves right in front of there spawn point, to kill any enemy agents which manage to get pas the blockers (however, there is always 1 agent who keeps collecting ammo, he will switch roles if one of the other agents is out of ammo).

doSpecialAction: This function is used for when agents have planned out some path they should follow instead of the default role-action. For example, the anti-sniper do use this function to get behind the sniper.

takeAmmo: One of the simplest functions: The agent takes the ammo, which can be taken in the least steps.

takePointWithAmmo: We created two functions for taking an enemy control point. One for which the agent has ammo,

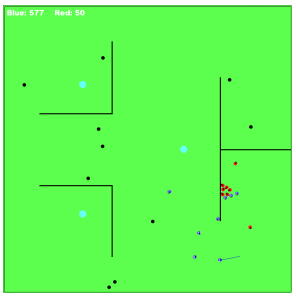


Fig. 4. deFuik in action

and the other for when he does not have ammo. They both have different priorities. The main reason for this is, when the agent already has ammo, taking a control point is more important than collecting ammo. And for when the agent does not have ammo, the collect ammo function is more important. Whenever this function is triggered, the agent will switch role with the current agent who supposed to guard the control point.

takeAmmo: This function will take care of collecting the ammo packs. When an agents sees a new ammo pack it will update the shared ammo observation. The ammo is distributed according the first come first serve method. This might seem a little nave method, but it actually works very well. The reason for this is, when an agent already has ammo, it is highly likely it has an other tasks which is of a higher priority than collecting ammo. This results in a fairly shared ammo between the agents.

doDefault: The very last function which holds the role-depended actions for the agents. This function will also decide when agents suppose to start sniping, or when to switch roles with each other. It's also possible for an agent to take a temporary role. This is useful for when a guard finds out that that the control point he supposed to take is already taken by the enemy, and he does not have ammo to shoot the enemy. In this case he should either switch role, or temporarily take a different role, and try again later.

D. Communication

In order to learn which action or strategy is the best, our agents have to communicate. Communication is done by updating scores every turn of each agent. Besides scores, the agents also keep a shared list with ammo location and enemy locations. Each of these locations keeps a score, which represents the probability of an ammo pack or enemy to be still on that location. With the shared ammo list it is possible to communicate where ammo is, which is unreachable for the agent that had observed it. Another agent can take it then. The shared enemy list can help with deciding which enemy to shoot, but as most enemy agents are moving every turn the location of enemy agents is in most cases unsure. Enemy snipers on the other hand can be communicated in this way.

E. Ammo distribution

Besides the behaviour of the enemy agents there is only one uncertain aspect in the configuration of the *Domination* game for which our brain is designed: the ammo distribution. Learning this distribution online during a game gives our agents the opportunity to find more ammunition. The learning of the ammo distribution is done by algorithm 3.

Algorithm 3 Update ammo scores for each area

```
1:  procedure UPDATEAMMODISTRIBUTION
2:      for all Agents do
3:          for all new found ammo in area a do
4:              nr_of_ammo(a) = nr_of_ammo(a)
5:          end for
6:          if Agent is in area a then
7:              score(a) = 0
8:          end if
9:      end for
10:     for all Areas a do
11:         score(a) = score(a) + 0.2 * nr_of_ammo(a)
12:         for all Neigbour_Areas n do
13:             score(a) =score(a) + 0.1 * nr_of_ammo(n)
14:         end for
15:     end for
16: end procedure
```

The world can be divided by a number of areas and by giving each area an ammo score agents learn which area is most likely to have ammo. If an area just is visited, it is not likely that there is ammo.

IV. Experimental Setup

We performed 5 separate experiments to evaluate the performance gain when using a number of components:

Experiment 1: Learn waypoints

This experiment was an attempt to find better waypoints between map areas than the hand-selected locations. After all, the speed and rotation constraints (up to 60 degrees per time step) complexify navigation and cause an optimal set of waypoints to be far from apparent. In this experiment, a genetic algorithm is used to optimize total travel time between a string of random locations on the map. In a population of size n agents, with genomes including 36 waypoints, each agent receives the same string of d destinations. The number of turns required to visit all destinations in order is used as a fitness score. The next generation of agents consists of the two best agents from the previous round (elitism), up to 25% recombinations of high-scoring agents (crossover) and 75% mutants. This is repeated for g generations. Besides the waypoints, the genomes include two genes specifying mutation chance and mutation magnitude. This was an attempt to allow for convergence and counter genetic drift.

Experiment 2: Learn sniper position

As said in section III our brain should learn which of two sniperpositions is the best to fight the opponent. In this small experiment we test if the brain does learn which is best and evaluate on a pathmap of the game if this seems right.

Experiment 3: Learn de Fuik

In this experiment we want to test the performance of our special strategy 'deFuik'. Our agent runs 200 games against 2 opponenet agents in 3 different configurations of

deFuik. As baseline we take our agent without ever using deFuik. 2 other configurations are tested upon performance improvement: conditional deFuik and permanent deFuik. The first only starts deFuik when our brain has 20% more points than the opponent brain and all control points are captured. The second always starts deFuik when all control points are captured. In the first place the performance is given by win percentage. In the second place we also take the difference in points as a performance measure. In the *Domination* game this is irrelevant, but with this measure we expect to show that deFuik will lead to an increase in this measure, if the opponent is weak. For that purpose one of the opponents chosen is weak and one is strong[1].

Experiment 4: Learn avoid/attack control point

By keeping track of points and deaths per control point, it becomes possible to adapt online to the tactical focus of the opponent. In this experiment we use the scores and death tallies per control point to rank the control points from safe to dangerous. If the enemy seems to be concentrating most of his forces on one particular control point, maybe it would be beneficial to attack a control point that is not as hotly contested. There, a little more manpower could tip the balance instead of just being cannon fodder. This is what we call an avoidant strategy. Alternatively, one could take a more confrontational, aggressive approach and reallocate agents from the most safely held control point to the one where the enemy is the strongest. In both strategies, up to 2 agents are permanently reassigned. Both will be compared with each other and with the default strategy.

Experiment 5: Learn ammo distribution

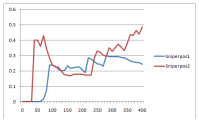
In our agent brain there is not a smart way implemented to adjust to the ammo distribution. Because of the time pressure and the results encountered without the learning of the distribution, ammo distribution was not our priority. The final tournament of the AMAAS Spring 2011 competition revealed that this was our biggest weakness. With the ammo distribution in the earlier rounds, our agent was almost always superior, but with the changed ammo distribution we had to admit ammo distribution should have more of our concern. In this experiment we test if our agents learn the ammo distribution by giving scores to different areas. Because of lack of time we were not able to implement how to deal with the learned ammo distribution, but we expect to show that our scoring algorithm works well. In this experiment the score is not set to zero when an agent passes the area.

V. Results

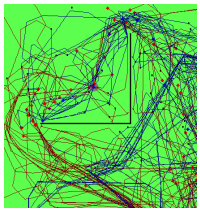
Experiment 1: Waypoints

The results in I were obtained with varying population sizes and numbers of destinations and generations. Improvement of the best evolved agent is measured as the decrease in required number of steps to visit 5000 destinations, compared to the original waypoints. Negative performance indicates an increase in the number of required turns.

[1]g7 and g1 in the AAMAS Spring 2011 tournament respectivily



(a) Scores of sniperpositions



(b) Pathmap of corresponding game

Fig. 5. Results of Experiment 2

TABLE I
Results of Experiment 1

Parameters	Improvement	P-value
Pop 40 Destinations 50 Generations 100	-0.53%	0.608
Pop 100 Destinations 100 Generations 100	1.89%	0.161
Pop 100 Destinations 50 Generations 200	5.21%	0.002
Pop 200 Destinations 200 Generations 200	1.88%	0.162
Pop 100 Destinations 50 Generations 800	3.70%	0.025

In the best runs, a statistically significant improvement in travel time was realized. But because the evaluation task for the genetic algorithm, navigation between random points only, was quite a simplification of the full *Domination* task, and the improvement wasn't very large, we decided not to use the obtained values in our actual agent.

Experiment 2: Learn sniper position

In this experiment it is shown that our brain has the ability to learn which of two sniperpositions is the best to deal with the opponent. In figure 5a the scores of 2 sniperpositions are given. The sniperposition with the highest score is the first sniperposition an agent picks. In the graph can be seen that sniperposition 2 is the first to pick for most of the game. As this is a game of our brain against an opponent brain with sniperdetection, the score of sniperposition 2 decreases for a moment. But after that both sniperpositions are taken sniperposition 2 again proves to be the best spot. In figure 5b the pathmap of this game around these sniperpositions shows that at sniperposition 2 (bottom left) the most opponent agents pass along that way and also are killed more at that point.

We also tested our agent with and without learning the sniperpositions. Some improvement was measured, but not significantly.

Experiment 3: Learn deFuik

The results in table II show that most of our expectations about the performance of deFuik are supported by the outcomes of experiment 3. Improvements in win percentage are not significant for both configurations of deFuik. Against the

weak opponent the conditional version shows improvement in the difference in points. So when the opponent is weak deFuik can help to win sovereign. Against the strong opponent deFuik does not have significant influence. The mean difference for the conditional is somewhat lower, the win percentage is higher though. A conditional configuration for deFuik seems the best.

TABLE II
RESULTS OF EXPERIMENT 3: DEFUIK

Opponent	Strategy	Win %	p-value win	Mean difference in points	p-value difference
weak (g7)	baseline	100		742.7	
	conditional	99.5	0.16	795.3	0.008
	permanent	100	-	774.9	0.125
strong (g1)	baseline	69		140.6	
	conditional	71.5	0.29	126.0	0.56
	permanent	72.0	0.26	130.5	0.69

Experiment 4: Learning role distribution

In this experiment we again ran 100 matches with different agents, in this case all three combinations of the unmodified agent, an agent with aggressively adaptive role reassignment and one with avoidant adaptive role reassignment were tested againt each other.

TABLE III
RESULTS OF EXPERIMENT 4

Strategy	Opponent	Win %
Aggressive	Default	39%
Avoidant	Default	40%
Aggressive	Avoidant	56%

It turns out that neither of the adaptive strategies manage to gain an advantage over the default team with fixed role distributions.Repeating the experiment with only one reassignable agent, instead of two, produced similar results.

Experiment 5: Learn ammo distribution

In figure 6 the scores for a game at time step 100, 200, 300 and 400 are given for 2 ammo distributions. The distributions are the used distributions for the AAMAS competition [2]. Our agents started in the top spawn point [3]. The scores are logically high where the agents come a lot. In our implemented algorithm this is fixed by setting the score to zero when an area is visited. In the figures it is clearly visible that as the standard deviation of the right distribution is smaller, the difference between better and worse areas is bigger and sooner recognizable.

VI. DISCUSSION AND CONCLUSION

With the current techniques, it is probably most feasible to create a strong solution to large stochastic multi-agent problems using a combination of learning and stationary policy, with the latter based on human expert knowledge and insight. Learning full policies remains a challenging goal

[2]Ammo distribution 1 (top): horizontal Gauss(0.5,0.2), vertical Gauss(0.5,0.35) ; Ammo distribution 2 (bottom): horizontal Gauss(0.5,0.3), vertical Gauss(0.3,0.1)

[3]the blue team

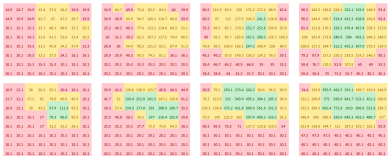


Fig. 6. Scores of ammo areas in 4 steps of the game for 2 distributions

because the inherent problem complexity quickly makes the problem intractable.

The first experiment showed that genetic algorithms can be applied to optimize waypoint-based navigation. However, the training setting was perhaps too dissimilar to the full task to be of real use. It would be interesting to explore the learning of more behaviors in the *Domination* domain.

The online learning algorithms in our system, seem to learn in an appreciated manner. The performance of our agent depends higly on other (hand crafted) strategies and the improvements in performance are not significant in most cases.

The strategies we tried in experiment 4 consistently performed worse than without reallocation. Something might be wrong with the reallocation logic, or agents might be reassigned too soon. Another possibility is that deviating from (near) equal distribution of roles might be detrimental in any situation, because the strength gain in one location is negated by a weakness in another, and teammates in an overcrowded area might lack ammo and hinder each other.

Learning the ammo distribution would be the first thing to improve in our agent brain. Experiment 5 showed that our agent is able to learn the ammo distribution, but we have not implemented how to use this information. It is certain that this could be a major improvement.

There is much more that can be explored and probably improved. Ideally an agent brain should be able to handle different configurations of the *Domination* game. It is likely that a team of agents under full control of a human player would far outperform all the agents we have seen thus far. This does not take away from the fact, that the tournaments in which our system participated have shown that it is a strong, robust and all-round *Domination* player among its peers.

REFERENCES

[1] Domination game site. [Online]. Available: http://code.google.com/p/mas-domination/

[2] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, vol. 11, pp. 1–94, 1999.

[3] T. Forschungsberichte, F. Iv, M. Mundhenk, and M. Mundhenk, "The complexity of optimal small policies," 1999.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, vol. 4, pp. 237–285, 1996.

[5] L. Buşoniu, R. Babuška, and B. De Schutter, "A comprehensive survey of multi-agent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 2, pp. 156–172, Mar. 2008.