ECSE 526
Assignment 3
Reinforcement Learning

Sean Stappas
260639512

November 7th, 2017

Introduction

1 Description of approach to generalization

Q-learning is the learning algorithm chosen for the agent. Many different distance metrics were used in the program, and close states are grouped based on this distance, as described by Mahadevan and Connell [4]. Here are three simple distance metrics, with the rationale for each choice of components.

Manhattan The MANHATTAN distance represents how close two states are on the Qbert pyramid, i.e., the number of blocks between them. This is a natural representation of distance in this environment, since Qbert can only change state by switching blocks and can only travel one block at a time (with the exception of using disks to jump back to the starting position). A distance of 1 was used for MANHATTAN.

Hamming The HAMMING distance represents the number of different state bits between two given states. The rationale behind this distance metric is that nearby states should have a similar bit pattern, assuming a sane state representation. A distance of 1 was used for HAMMING.

SameResult The SAMERESULT approach groups adjacent states together, such that, whenever a Q-update occurs for a certain Qbert position, all possible states that could have directly led to that state also get updated. The rationale here is that, for most states in the Qbert world, the path Qbert takes to get to that state is usually not important.

Many different state representations were used with varying results. First, a state representation was used keeping track of the color of every block, the position of disks, Qbert's position, the position of enemies (purple agents) and the position of friendlies (green agents). While this state representation completely encapsulates Qbert's world, it is very verbose and intractable if not using any distance metric, as described previously.

With 21 blocks in the pyramid, an agent has 21 possible positions, representing 5 bits of entropy. This applies for Qbert, the enemies and the friendlies. A simple representation of the colors of the blocks can indicate if the block color is the goal color or not. With this binary value for each block, the colors can be represented with 21 bits. Finally, there are 12 possible locations for the disks, with each disk either being present or not, representing 12 bits of entropy. With all these elements in a state representation, we would need a total of $5 * 3 + 21 + 12 = 48$ bits to represent a state completely. This represents a state space with $2^{48} \approx 3 \times 10^{14}$ states. Clearly, this number of states is intractable for learning, showing the necessity for generalization with an appropriate distance metric. This state representation is referred to as VERBOSE in the code.

Another approach to generalization is to use a simpler state representation. A simpler state may not completely represent the game world, but can be enough for an agent to learn how to behave correctly. For example, instead of keeping track of all enemies, blocks and friendlies, one can only keep track of enemies, blocks and friendlies around Qbert, since that is what is important to make an immediate action. One problem with this approach, however, is that it can be hard for the agent to learn to go towards all the blocks in a level to change all the colors and finish the level. More information, such as the number of colored blocks on the board, can then be provided as well. This state representation is called SIMPLE in the code.

Finally, another way to deal with a very large state space is to separate the learning problem into sub-tasks, allowing components of the state to add in complexity instead of multiply. Indeed, if we have separate tasks of enemy avoidance, friendly catching and block finding, we can have a simple specialized state for each task. This is the SUBSUMPTION model described by Mahadevan and Connell [4], and is explored further in Section 5, where the FRIENDLYCATCHER, BLOCKFINDER and ENEMYAVOIDER learners are described in detail.

To simplify the learning process and avoid recurring sudden death, Qbert's motion is restricted to within the pyramid (with the exception of reaching discs). This allows Qbert to focus on learning to find blocks, avoid enemies and find friendlies.

2 Results of generalization

For all the plots in this report, scores were reported for 100 episodes of playing Qbert. For each episode, Qbert keeps playing until he has no more lives yet, and the total score obtained is recorded. Also, for the following plots showing various generalization results, no exploration is being used.

The results of the Qbert agent with no generalization can be seen in Figure 1. We can see that learning takes place relatively slowly because of the massive state space.

The results of the Qbert agent using the MANHATTAN distance metric can be seen in Figure 2. It

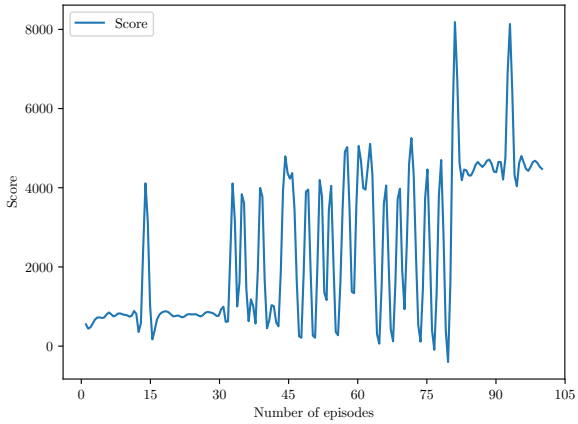

Figure 1: Score achieved by the agent with no generalization and no exploration.

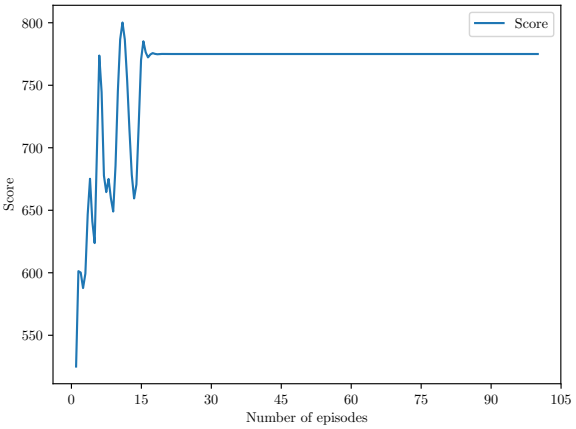

Figure 3: Score achieved by the agent using the HAMMING *distance metric.*

seems that learning is actually hindered with this metric, with overall lower scores being obtained. This may be explained by the fact that assigning the same Q-value to adjacent states may not always be wise. For instance, if Qbert dies to an enemy at a specific block, it probably doesn't make to assign the same penalty to all nearby blocks.

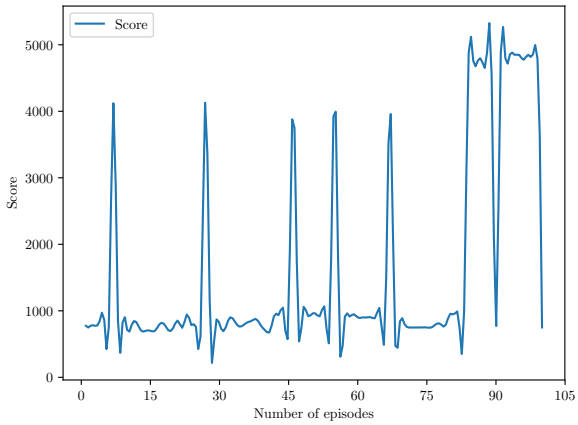

Figure 2: Score achieved by the agent using the MANHATTAN *distance metric.*

The results of the Qbert agent using the HAMMING distance metric can be seen in Figure 3. While the learning converges relatively quickly to a score, it clearly fails to converge to a very high one. Once again, the distance metric may have caused unintended states to have the same reward, causing repetitive destructive behaviour.

The results of the Qbert agent using the SAMERESULT distance metric can be seen in Figure 4. Here, the distance metric clearly failed to provide learning, with the score converging to a lower value than the starting one. By assigning the same utility to states that lead to the same other states, the agent has degenerated to repetitive behaviour, leading to easy prey for the purple enemies.

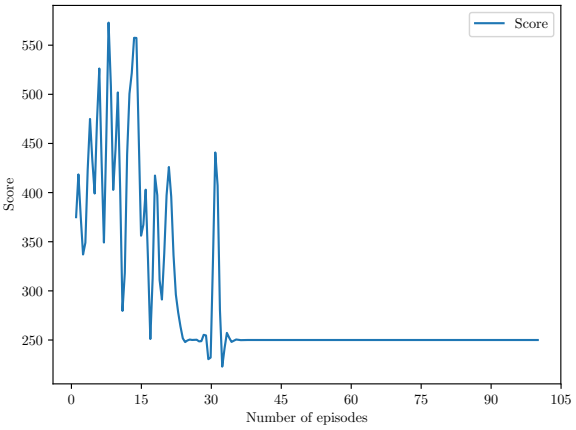

Figure 4: Score achieved by the agent using the SAMERESULT *distance metric.*

It is clear that all the described distance metrics have not been successful. This can be explained by the simple fact that, even with distance metrics, the VERBOSE state space is much too big. To address this issue, we have created a SUBSUMPTION agent using the SIMPLE state representation. The details of this model will be described in Section 5, but its performance as a generalization technique can be seen in Figure 5. Here, we see a clear upward learning trend despite the spiky data. Note that the spikiness in the data is due to the great danger in the Qbert world, where death is very easy near purple enemies.

Note that exploration has been completely ignored here, since it will be discussed in the next

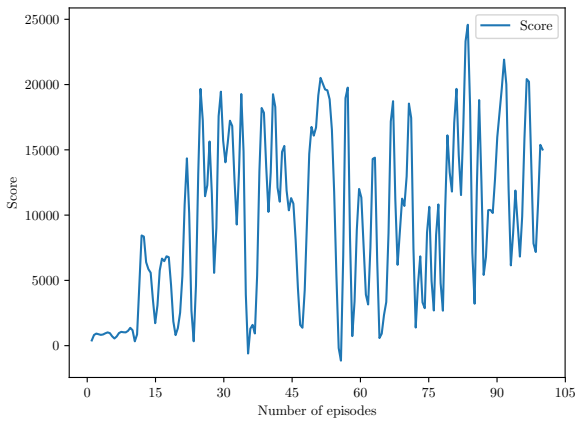

Figure 5: Score achieved by the SUBSUMPTION *agent using a* SIMPLE *state representation.*

section. This explains the relatively mediocre score results. It should be noted, however, that with the subsumption model, the agent learns with the ENEMYAVOIDER quickly enough to avoid purple enemies, and this acts in a way as a form of exploration, since the enemies will force Qbert to go around the pyramid to avoid death. Indeed, this will force Qbert to visit states that he would normally not visit.

3 Description of approach to exploration

First, the ϵ-greedy approach to exploration was implemented, where a random action is chosen with probability ϵ and the greedy action is chosen otherwise. An ϵ value of 20 % was chosen, since exploration is very important to finding all the blocks to complete a level. The ϵ-greedy approach is the one chosen by many researchers using reinforcement learning in the ALE [1, 2, 3]. This method is attractive for many reasons. First, it is very simple to implement with any standard random number generator. Second, despite its simplicity, it can lead to very powerful results, since it can allow an agent to explore states it would never normally explore. However, convergence to an optimal policy can be slow with ϵ-greedy. Also, this method can be dangerous in the Qbert game, where a bad random action can lead to sudden death. For this reason, the random action is only chosen for the FRIENDLYCATCHER and BLOCKFINDER, since a bad random action from the ENEMYAVOIDER can lead to sudden death. This approach will be referred to as RANDOM exploration.

Next, weighting with optimistic priors was implemented, using $N(s, a)$, i.e., the number of times action a has been attempted in state s. The simple exploration function used can be seen in Equation (1). An optimistic score of 100 was chosen, since it is higher than the score obtained by jumping on a block without a goal color (25). This method can often lead to meaningful convergence much faster than ϵ-greedy, since it will directly explore unvisited states, instead of choosing randomly. This method is theoretically very attractive in the Qbert world, since exploring unexplored states usually means going on blocks without the goal color, turning them to the correct color. This approach will be referred to as OPTIMISTIC exploration.

$$f(u, n) = \begin{cases} 100 & n < 1 \\ u & n \geq 1 \end{cases} \tag{1}$$

Finally, a combination of ϵ-greedy and optimistic prior was also experimented with. With this approach, the agent chooses a random action with probability ϵ and uses the exploration function given in Equation (1). This can theoretically offer the benefits of both ϵ-greedy and optimistic prior, i.e., the benefits of random exploration and optimistic unexplored states. This approach will be referred to as COMBINED exploration.

4 Results of exploration

The results of the SUBSUMPTION agent using RANDOM exploration can be seen in Figure 6. The results are relatively good, although it is hard to tell if the agent is correctly learning. This is most probably due to the fact that 100 is a relatively low number of episodes.

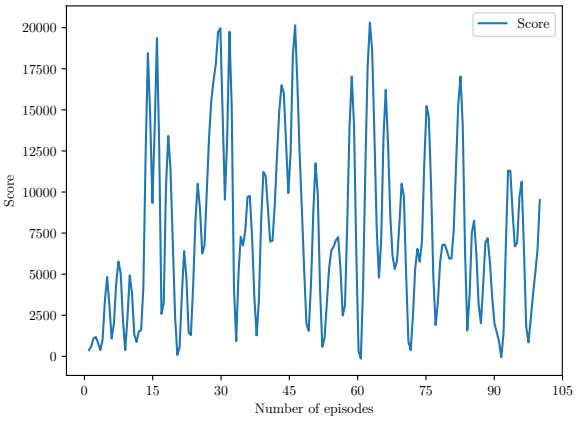

Figure 6: Score achieved by the SUBSUMPTION *agent with* RANDOM *exploration.*

One problem that was clear with ϵ-greedy exploration is that it can be very slow to learn and waste a great deal of training time in the same states. For

example, with ϵ-greedy, Qbert would often oscillate between two adjacent blocks, not seeing the need to explore further until the it randomly chose (with probability ϵ) to explore further. This can be seen in Figure 7, where Qbert is oscillating between the bottom right block and the block to the top right of it. This can explain the slow learning seen in Figure 6.

Figure 7: Screen capture during Qbert's oscillation between two states.

The results of the SUBSUMPTION agent using OPTIMISTIC exploration can be seen in Figure 8. These results are not as encouraging, with the score oscillating around a relatively low value. This can most likely be explained by the fact that we used a relatively simplistic exploration function. With a more sophisticated one, we can most likely obtain better results.

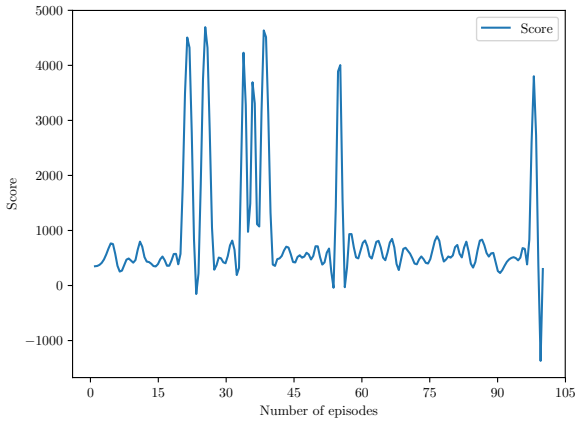

Figure 8: Score achieved by the SUBSUMPTION *agent with* OPTIMISTIC *exploration.*

The results of the SUBSUMPTION agent using COMBINED exploration can be seen in Figure 9. These results are much more encouraging. We can see the same initial high score values as for RANDOM exploration, with a clear increasing trend.

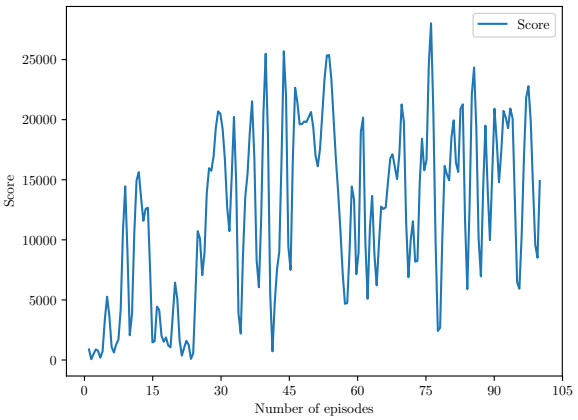

Figure 9: Score achieved by the SUBSUMPTION *agent with* COMBINED *exploration (ϵ-greedy and optimistic-prior exploration).*

5 Agent Performance

Based on the results from the previous sections, it is clear that the SUBSUMPTION approach described by Mahadevan and Connell [4] gives the best results. The model for this approach will now be described in detail.

The agent is separated into three learners: ENEMYAVOIDER, FRIENDLYCATCHER and BLOCKFINDER. The structure is shown in Figure 10, where the "S" suppressor nodes denote priority, i.e., avoiding enemies takes priority over catching friendlies and going on blocks. This approach allows the agent to learn various aspects of the Qbert game independently, with a reduced state space for each task. This separation also makes intuitive sense, since learning how to avoid enemies should have little to do with finding blocks. Combined with the SIMPLE state representation described in Section 1, the SUBSUMPTION model can lead to a very small state space which can be learned very quickly.

The ENEMYAVOIDER is active whenever an enemy (COILY and PURPLEBALL) is close to Qbert, i.e. in any of the blocks directly adjacent to Qbert or 2 blocks away. An important strategy is to kill the purple snake enemy (COILY) by jumping on to the floating disks on the left and right of the board. If COILY is nearby, he will jump off the board, giving Qbert 500 points. This maneuver can be seen in Figure 11. This score is fed as a reward to the enemy avoider learner. As a penalty for dying to COILY, a negative reward of -100 is given to the

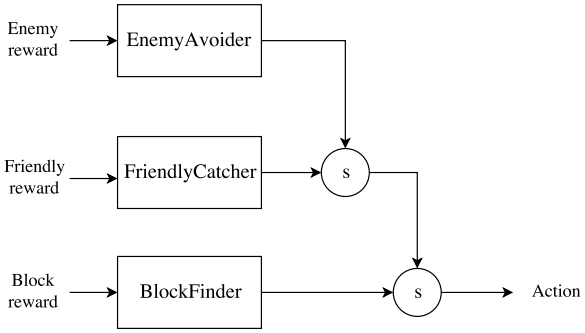

Figure 10: Subsumption network used by the SUBSUMPTION *agent.*

learner. This combination of positive and negative reinforcement allows the agent to learn how to avoid and kill COILY.

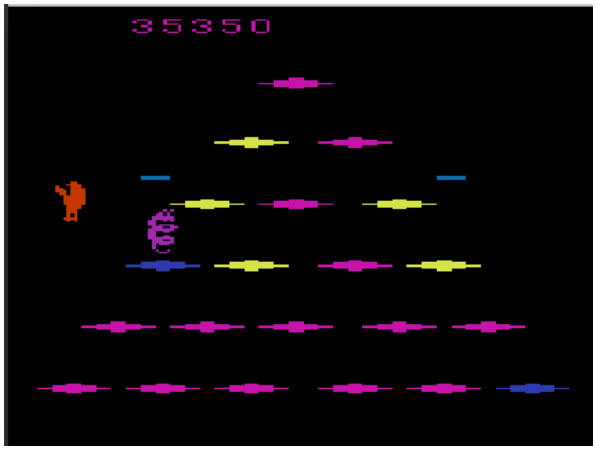

Figure 11: Screen capture during Qbert's jump on a disk to kill Coily.

Indeed, avoiding COILY is probably the most important task to learn in the game. Not only does it prevent Qbert from dying, but it inadvertently makes him explore the entire map while avoiding COILY. This avoidance is essential, and after many runs, the SUBSUMPTION agent learns to go back and forth between tiles to out-maneuver COILY. This can be seen in Figure 12.

The FRIENDLYCATCHER is active whenever a friendly green agent (SAM and GREENBALL) is within 2 blocks of Qbert. This learner encourages Qbert to catch SAM for 300 points and GREENBALL for 100 points. Stopping Sam is also important because he changes the colors of blocks back to their original values, making Qbert do more work.

The BLOCKFINDER is always active and keeps track of the colors of blocks within 2 blocks' distance of Qbert. It specifically keeps track of the number of adjacent blocks that are not of the desired color, encouraging Qbert to change them to their correct color to eventually complete the level. It receives a 25 point reward for turning a block to the goal color, and a 3100 point reward for completing a level.

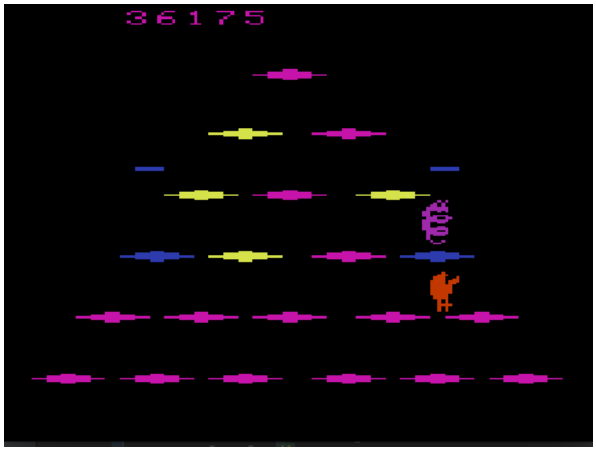

Figure 12: Screen capture during Qbert's back-and-forth maneuver on the bottom corner to avoid COILY.

Note that, although one learner's decision always has priority for executing an action, multiple learners can still learn simultaneously. For example, while Qbert is avoiding an enemy, the block finder learner can still be learning from the actions being taken.

The score results of the SUBSUMPTION agent with seeds of 123, 459 and 598 can be seen in Figures 13 to 15. We can see that the agent performs relatively well for all of these seed values, showing that the agent generalizes well. To fully appreciate the upward trend of learning, it would require more than 100 episodes, but the training is relatively time consuming.

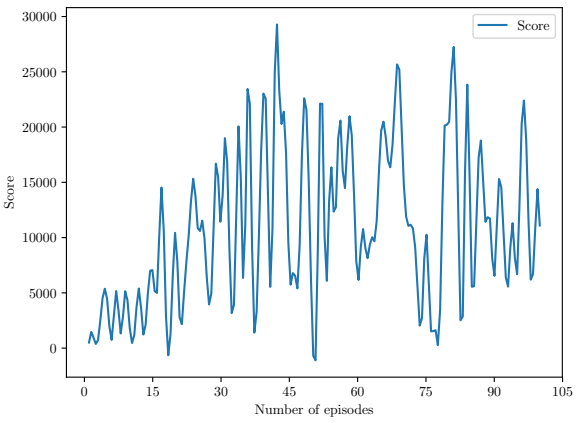

Figure 13: Score achieved by the SUBSUMPTION *agent with a seed of 123.*

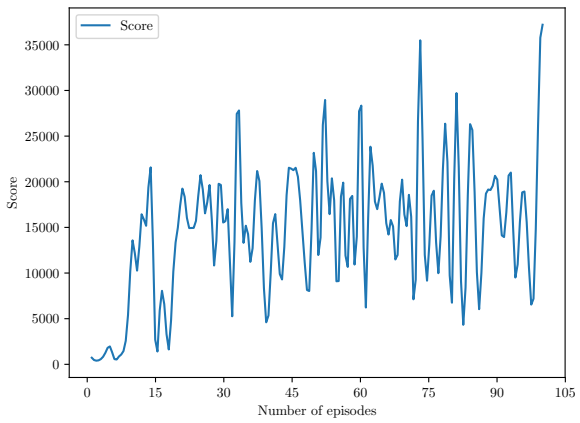

Figure 14: Score achieved by the SUBSUMPTION *agent with a seed of 459.*

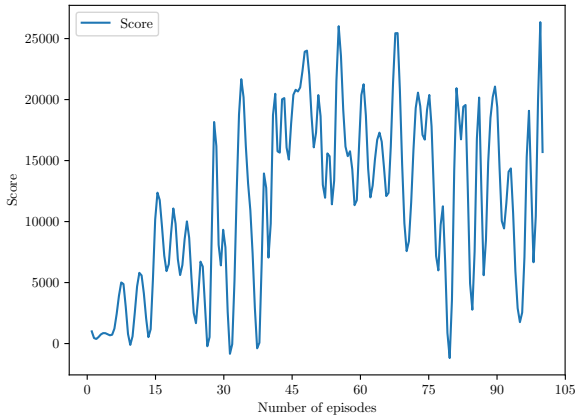

Figure 15: Score achieved by the SUBSUMPTION *agent with a seed of 598.*

With repeated training, the highest score achieved by the agent is around 41800 and the maximum level reached is 9. There are multiple possible improvements to be made to the agent's behaviour. One would be to discourage it from staying near the top of the pyramid, since this is where PURPLEBALL spawns. This could possibly be achieved by keeping the ENEMYAVOIDER learner active even when there are no enemies, or only when Qbert is near the top of the pyramid. Another problem is that Qbert doesn't seem to explore very well when he is not avoiding COILY. This could possibly be addressed by encouraging Qbert further to find uncolored blocks by specifying how many blocks are left without the goal color.

Acknowledgments

There was a discussion with Andrei Purcarus, Andrew Lowther and Juliette Valenchon concerning extracting relevant data from the ALE and high-level learning strategies. Notably, the positions of various important bytes in the ALE RAM were discussed, such as the bytes indicating a safe move update and the byte indicating a level change.

References

[1] Aaron Defazio and Thore Graepel. A comparison of learning algorithms on the arcade learning environment. *arXiv preprint arXiv:1410.8620*, 2014.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[4] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence*, 55(2-3):311–365, 1992.