2012 IEEE 24th International Conference on Tools with Artificial Intelligence

Object-Oriented Representation and Hierarchical Reinforcement Learning in Infinite Mario

Mandar Joshi, Rakesh Khobragade, Saurabh Sarda and Umesh Deshpande
Computer Science and Engineering
VNIT, Nagpur
Nagpur, India
mandarjoshi.90@gmail.com

Shiwali Mohan
Computer Science and Engineering
University of Michigan, Ann Arbor
Ann Arbor, USA

Abstract— **In this work, we analyze and improve upon reinforcement learning techniques used to build agents that can learn to play *Infinite Mario,* an action game. We extend the object-oriented representation by introducing the concept of object classes which can be effectively used to constrain state spaces. We then use this representation combined with the hierarchical reinforcement learning model as a learning framework. We also extend the idea of hierarchical RL by designing a hierarchy in action selection using domain specific knowledge. With the help of experimental results, we show that this approach facilitates faster and efficient learning for this domain.** *(Abstract)*

Keywords-hierarchical reinforcement learning; action games; object-oriented representation; action selection (keywords)

I. Introduction

Action games are an interesting platform for research in artificial intelligence and machine learning. They encourage quick decision making, as penalties are often incurred if the agent fails to finish the game quickly. Due to the high degree of relative motion between game entities, action games demand good physical reactive skills. Furthermore, players have to often choose between competing goals and make decisions that can give favorable long term rewards.

In this work, we concentrate on designing a reinforcement learning (RL) agent for a variant of a popular action game, Super Mario Bros, called *Infinite Mario* developed for RL-competition, 2009 [1]. This domain, like most action games, poses a challenge in the form of a very large search space (in terms of time and memory).

In most domains, the core objective of RL agents is to learn a good policy – a mapping of states to actions. Policies are usually represented as a table of states and actions, with the value in each cell indicating the utility of taking a particular action in that particular state. Such an approach, while adequate for simpler domains, is unsuccessful in complex domains, such as *Infinite Mario*, because of the very large state space. This makes the action selection problem particularly difficult. To enable efficient learning in such domains, some form of generalization and abstraction is needed.

Through this work, we explore how different state representations influence the agent's ability to learn the game playing task. We use an object-oriented representation to model the behaviour of game entities and extend this representation to incorporate the concept of object classes, which help the agent to learn quickly by constraining the state space. We then show how a hierarchical approach to reinforcement learning, which divides the game playing task into simpler subtasks, can help in directing the agent's exploration. We extend the idea further by designing a hierarchy in action selection as well, using domain specific knowledge about the behaviour of game entities. With the help of experimental results, we claim that this approach provides for a higher domain specific performance level as it brings about a reduction in both state and action spaces.

II. Related Work

For an extensive overview of RL related work, we refer to [2]. Several hierarchical reinforcement learning (HRL) techniques have been developed e.g., MAXQ [3], Options [4], HAMs [5]. A summary of work in HRL is given by [6]. The motivation and scope for research in computer games is documented in [7]. Driessens [8] combined RL with regression algorithms to generalize in the policy space. Marthi et al. [9] applied hierarchical RL to scale to complex environments where the agent has to control several effectors simultaneously. Ponsen et al. [10] employed a deictic state representation that reduces the complexity compared to a propositional representation in a real-time strategy (RTS) game Battle of Survival. Their approach allows the adaptive agent to learn a generalized policy, i.e., it is capable of transferring knowledge to unseen game instances. The game environment for Battle of Survival consists of a 32 X 32 grid world where the adaptive agent's task is to reach the goal state without being destroyed by the enemy soldier agent. In contrast, *Infinite Mario* has a continuous state space and contains numerous game entities, each having its own characteristic behavior.

In the domain of action games, Diuk et al. [11] introduce an Object-Oriented representation that is based on interaction of game entities called objects. They claim that this representation enables multiple opportunities for generalization and demonstrate the applicability of this representation by designing an agent that learns to play Pitfall, an action game. Mohan and Laird [12] use this representation to propose a reinforcement learning framework that gives promising results on a subset of game instances for the Infinite Mario domain. Their approach aims at creating learning agents that begin with little or no background knowledge about the game environment. In contrast, we use domain specific knowledge and extend the object oriented representation by introducing the concept of object classes, which play a major role in constraining the state space. We also use domain specific knowledge to design a hierarchy in action selection. We integrate this action selection hierarchy with the learning framework and show that this approach gives better results in terms of the average final reward, as well as the time required to learn the game playing task.

III. Preliminaries

Reinforcement learning is a discipline of machine learning, which studies how agents can learn to optimize their behavior based on the reward signal received from the environment. Markov Decision Processes (MDPs) are a mathematical framework used to model reinforcement learning problems. Most of the current research in RL is based on the MDP formalism [2].

A. Markov Decision Processes

The finite MDP is characterized by a fully observable discrete, finite environment. The environment is assumed to possess *Markov Property,* i.e., the next state, *s'* depends on the current state, *s* and action, *a*. But given *s* and *a*, it is conditionally independent of all the previous states and actions.

The objective of the RL agent is to choose actions to maximize its cumulative rewards. The eventual "solution" to the MDP is a policy, π. A policy defines the learning agent's way of behaving at a given time. It is a mapping from perceived states of the environment to actions to be taken when in those states and is denoted by π: $\mathbf{S} \times \mathbf{A} \rightarrow [0, 1]$, where **S** is the finite set of states and **A** is a set of available actions.

B. SARSA

SARSA (s_t, a_t, r, s_{t+1}, a_{t+1}) is an on-policy temporal difference learning algorithm for Markov decision processes [13]. It is based on the following update equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * \delta \qquad (1)$$

$$\delta = r_{t+1} + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \qquad (2)$$

Here, $0 \leq \alpha \leq 1$ is the learning rate parameter and $0 \leq \gamma < 1$ is the discount factor for future rewards.

The proposed agents use SARSA for policy update and ε-greedy for action selection.

C. Semi-Markov Decision Process

The semi-Markov decision process (SMDP) is a generalization of the MDP which incorporates temporally extended actions. Environment transitions in SMDPs have stochastic time duration. Providing the means to incorporate temporally extended actions allows an RL agent to handle complex tasks.

Many versions of HRL are based on temporally extended actions ([4], [5]). In our solution, the levels in the hierarchy use an SMDP model for reinforcement learning to choose among temporally extended actions

IV. Problem Specification

Infinite Mario is a reinforcement learning domain developed for RL-competition, 2009. It is a complete side scrolling game with destructible blocks, coins, monsters, pits, and raised platforms. The objective of the game is to maximize the total reward by collecting coins and killing monsters on the way to the finish line. One run from the starting point to the finish line (or to the point where Mario gets killed) is called as an episode. If Mario reaches the finish line, the game restarts on the same level.

A. Environment

At any given time-step, the agent perceives the visual scene as a two-dimensional [16 x 22] matrix of tiles, where each tile can have 13 different values. The agent has access to a character array generated by the environment corresponding to the given scene. The domain also provides attributes like speed and position of the dynamic elements as double precision arrays. Fig. 1 shows a snapshot of the game environment.

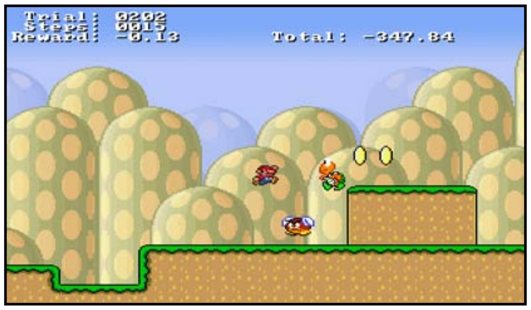

Figure 1. A snapshot of the game environment at difficulty level 1, seed 121

Each action is composed of three action classes – motion (left, right, stationary), jump (yes, no), and speed (high, low). Thus, there are a total of 12 actions that are available to the agent at any time step.

For each interaction with the entities in the environment, the agent receives a reward. The rewards could be positive or negative.

The RL competition software is capable of generating several instances of the game with great variability in difficulty level. As the difficulty level increases, interaction with elements becomes more complex, i.e., the agent has to deal with more entities of different types. This makes decision making more difficult. A detailed specification of the environment is given in [14].

B. Challenges

- *Large state space*: One of the main challenges of the domain is a large and continuous state space. As reinforcement learning uses the trial and error paradigm, large state spaces are a major deterrent to the learning process. To make learning tractable, some form of discretization and abstraction is required. The learning framework, that uses the object-oriented representation combined with hierarchical RL, enables the agent to constrain the state space and facilitates effective learning.
- *Action selection*: At any time step, the visual scene is occupied by a large number of objects and corresponding goals. The action selection problem, given these multiple goals, is combinatorial in the number of objects and is largely intractable. Additionally, some of the actions may be irrelevant or useless in context of a particular goal. The hierarchy introduced in action selection helps in constraining the action space.

C. Benchmark : Memory Agent

RL Competition, 2009 provides a sample learning agent which is used as a benchmark for the proposed solution. The sample agent does not use any reinforcement learning but learns through memorization. Though this strategy is very simple, it enables the agent to learn quickly. However, the final reward obtained by the agent is restricted by its limited exploration. Furthermore, the agent cannot transfer its learning in between different levels of the game. An RL framework, on the other hand, is capable of this transfer.

V. Agent Design

In this section, we discuss the proposed solutions to the problem. We first address the issue of a large state space by discussing various state representation schemes. We then talk about how hierarchical reinforcement learning combined with an action selection hierarchy can help tackle the action selection problem.

A. Flat RL agents

In [11], object oriented representation is used to model another video-game, *Pitfall.* In this representation, the environment is described as a collection of objects – entities that the agent can interact with and thus affecting the agent's behavior. For the *Infinite Mario* domain, the set of *objects* includes monsters, blocks, coins, raised platforms, pits and pipes. An object is characterized by its attributes, e.g., an instance of the object *monster* will have the (x and y components of the) distance of the monster from Mario, speed of the monster, and its *type* as attributes. Mohan and Laird also used a similar representation in their solution for the *Infinite Mario* domain in [12].

However, including all objects in the visual scene as a part of the state will result in a large state table. As an extension to the object oriented representation used in [12], we divide objects into various classes, and each class defines the range of inclusion of the object instance in the state. This range is pre-programmed using domain-specific knowledge about the way Mario interacts with the game entities, e.g., a monster's range is larger than that of a coin or a question block, as monsters pose a threat to Mario. The central idea behind using object classes is to exclude objects not influencing the agent's immediate actions from the state. The state table in our approach is incremental, i.e. a state is added to the state table only when it is encountered for the first time in the game.

Fig. 3a shows the performance of the agent that uses the state representation described above on seed 121, difficulty level 0. For reasons that will become apparent later on, we refer to this agent as the multiple-object agent. The multiple-object agent converges to a policy in about 720 episodes and earns an average final reward of 140.64 over the last 100 trials on difficulty level 0. After sufficient training, the agent outperforms the memory agent in terms of the average final reward due to better exploration that this design provides. The multiple-object agent is able to converge to a policy at difficulty level 0 because the introduction of object classes aggregates several similar ground states into smaller abstract states. On difficulty level 1, however, the agent is unable to learn a good policy (as can be seen from Fig. 4a). On this level, Mario is threatened by multiple monsters which are in close vicinity and due to this, the state table increases at an even larger rate. A stronger form of abstraction is, thus, required to enable the agent to learn on higher difficulty levels.

One way to do this is to represent the state as a single object selected from those that lie within the range specified according to their corresponding object class. The selection is made according to programmed selection rules based on domain knowledge. The objective is to select the object that should most influence agent behavior. We refer to this agent design as the single-object agent.

Fig. 3b shows the performance of the single-object agent on difficulty level 0 of seed 121. Seed 121 of the game has been specifically chosen to enable comparison with the results presented in [12]. The agent converges to a policy in 601 episodes and earns an average final reward of 137.68 over the last 100 trials. After sufficient training, the single-object agent too outperforms the memory agent in terms of the average final reward. When compared to the multiple-object agent, the single-object agent earns a lower average final reward. Its final reward is limited by the fact that the design does not allow the agent to exploit all the sources of reward in its vicinity.

On difficulty level 1 (Fig. 4b), the single-object agent performs "better" in the sense that the number of episodes where Mario reaches the finish line is higher for the single-object agent compared to the multiple-object agent. The agent, however, fails to learn a good policy. This is due to the fact that the agent's ability to interact with multiple monsters is crucial for learning a good policy at higher difficulty levels. As the state consists of only a single object, the agent finds it difficult to choose optimal actions because it does not have all the information that should influence decision making. Moreover, a strategy that uses programmed selection rules to determine whether an object should be included into the state may be too general for some cases. For instance, it may be reasonable to include a monster into the state when the choice is between a monster and a coin, but choosing between multiple monsters may be less obvious as can be seen from Fig. 1.

The results show that while some form of abstraction is crucial to making learning tractable, it may also limit the reward the agent earns. The agent design has to strike a balance between the detail captured in a state and the increase in the size of the state table that may result from a detailed state representation. These designs have not yet addressed the challenge of action selection explicitly. If the agent design allows for directed exploration, the agent may be able to learn a good policy at higher difficulty levels.

B. *Hierarchical RL Agents*

The basic idea behind hierarchical RL is to divide a complex task into a hierarchy of simpler subtasks. Each subtask can then be modeled as a single Markov Decision Process. Hierarchical approaches to reinforcement learning (RL) problems promise many benefits:

- Improved exploration (because exploration can take “big steps” at high levels of abstraction)
- Learning from fewer trials (because fewer parameters must be learned and because subtasks can ignore irrelevant features of the full state)
- Faster learning for new problems (because subtasks learned on previous problems can be re-used) [3]

At the lower (action) level of the hierarchy, the HRL agent utilizes a state representation similar to the multiple-object agent in that it does not restrict the number of objects to be included in the state to a pre-determined number. Monsters and pits being objects that pose a threat to Mario, the agent now gives them priority by not adding objects (other than pits and monsters) if the state already contains at least one monster or a pit. This enables the agent to strike a balance between survival and the need for higher reward. While such a representation does result in an increase in the state-space (when compared to the single-object agent), it is offset by improved and directed exploration that the HRL framework provides. The state representation at the subtask level also uses object classes. However, the double precision attributes are rounded off to the nearest integer. Thus, the state representation at the lower (action) level of the hierarchy is more detailed as compared to that at the higher (subtask) level.

In our approach, the game playing task is accomplished using combinations of four subtasks, namely *MoveRight*, *GrabCoin*, T*ackleMonster* and *SearchQuestionBlock*. Each subtask is initiated with respect to a specific object in the state and spans over multiple one step actions as shown in Fig. 2. Here, subtasks can be thought of as temporally extended actions and the entire problem can be modeled as an SMDP.

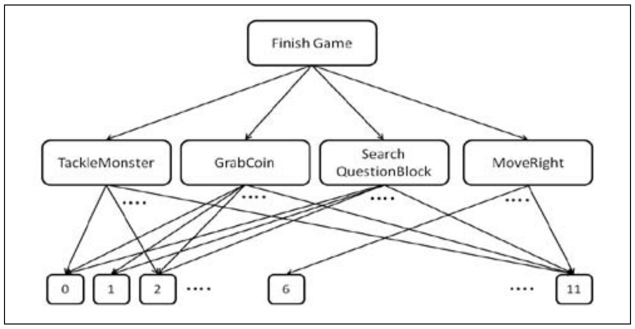

Figure 2. Subtask hierarchy in Infinite Mario

Learning to carry out a subtask, therefore, involves identifying a set of primitive actions that allow the agent to interact “successfully” with the selected object. Termination criteria are defined separately for each subtask to identify whether or not the interaction has been successful. For example, the terminal state for *TackleMonster* subtask is reached when the agent either kills or avoids the monster.

After the state is created, the agent tries to select one of the objects in the state and initiates an appropriate subtask. For example, if the agent selects a coin, it initiates the *GrabCoin* subtask to grab it. The object selection is done by learnt object selection preferences, where a value is associated with each object in the state and this value indicates the utility of selecting that object. The agent uses SARSA to update this value after the termination of every subtask, based on the cumulative sum of rewards received from the environment.

Additionally, whenever a new state is encountered, the action optimal for the *nearest similar* state is selected. We find the nearest similar state as follows. A search is made into the state table to find the set of all states which have the same entities as the present state, *j*. For every member, *i* of this set the state distance from the present state is calculated using Eq. (3).

$$\text{StateDistance}_{ij} = \sum_{\text{entities}} \sum_{\text{attributes}} |\,\text{att\_value}_i - \text{att\_value}_j| \quad (3)$$

The state for which the state distance is minimum and below a threshold value is the nearest similar state. If no such state exists, a random action is selected.

Figures 3c and 4c show the average performance of this HRL agent on difficulty level 0 and 1 respectively, on seed 121. On difficulty level 0, the agent converges to a policy in 400 episodes and earns an average final reward of 140.08. It not only learns faster but also achieves a higher reward when compared to the single-object agent. This is because the agent now has more information about the reward sources around it due to a more detailed state representation. The hierarchical RL model introduces abstraction in state representation at higher levels while retaining the benefits of

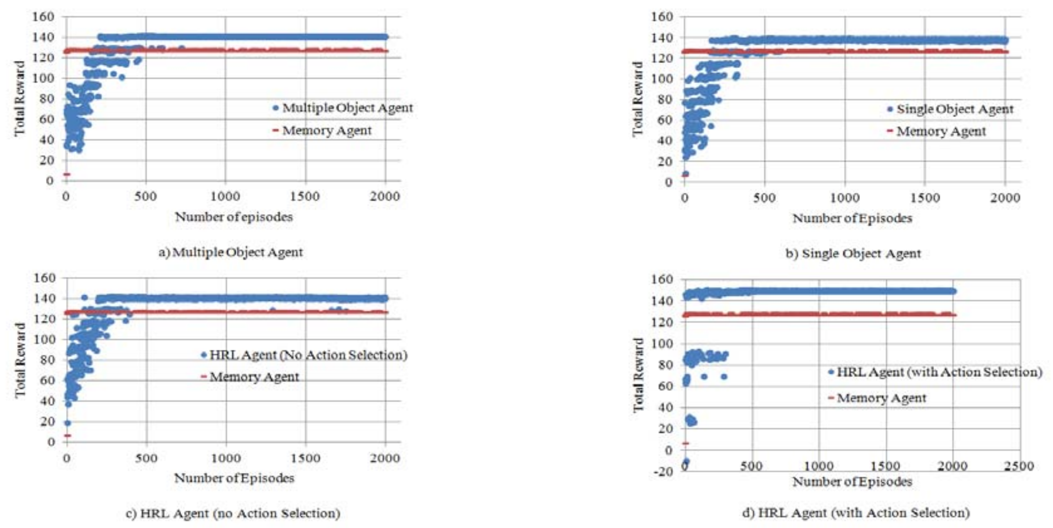

Figure 3. Performance on difficulty level 0, seed 121 of Infinite Mario. All results have been averaged over 10 runs of 200 episodes each.

detailed state representation at lower levels. By exploiting the interplay between high-level and low-level representations, the agent learns more efficiently and more accurately.

The learning framework described above also allows for incorporating a hierarchy in action selection. The first level of the action selection hierarchy uses qualitative measures (depending on the present state) to reject irrelevant and useless actions corresponding to the interaction with the selected object. These measures are based on the characteristic properties and behavior of the game entities. For example, in certain cases, it might be possible to predict a collision between Mario and other game entities. Consider the case where a monster is very close to Mario. Depending on the position and speed of the monster, the agent may rule out certain actions (like remaining stationary) which might lead to a collision. Similar reductions in action space can be done for other objects. Once this reduction is done, the agent uses the learning framework to select an optimal action from the remaining set of actions. Thus, action selection at the second level of the action selection hierarchy is quantitative.

Fig. 3d and Fig. 4d show the average performance of HRL agent with an action selection hierarchy on difficulty level 0 and 1 respectively, on seed 121. On difficulty level 0, this agent converges to a policy in 300 episodes and earns an average final reward of 149.48 over the last 100 trials. The considerable difference in the final reward between the two HRL agents is due to the reduction in action space brought about by the action selection hierarchy. The impact of the action selection hierarchy is even more striking when results on difficulty level 1 are compared (Fig. 4c and Fig. 4d). Without an action selection hierarchy, the HRL agent fails to learn a good policy on difficulty level 1. On the other hand, the HRL agent with an action selection hierarchy is able to converge to a policy within 1000 trials. Having an action selection hierarchy allows the agent to constrain the action space. As subtasks are actually temporally extended actions, a reduced action set greatly enhances the agent's ability to learn subtasks quickly.

While comparing with RL agent designs in [12], we use the same performance measures – the average final reward, and the number of trials it takes the agent to converge to a policy. Thus, the HRL agent with an action selection hierarchy performs better than the agents in [12] in terms of both the performance measures, on difficulty level 0. On difficulty level 1, the proposed agent converges to a policy in about 1000 episodes and earns an average final reward of 127.64 while agents proposed in [12] failed to learn a good policy. The improved performance can be attributed to the two principal reasons. The division of objects into classes helps in constraining the state space. The hierarchy introduced in action selection allows for intelligent action selection because the agent has fewer relevant actions to choose from.

VI. LIMITATIONS

The introduction of a range of inclusion for various object classes plays a major part in constraining the state space. However, arriving at a specific value for a range of inclusion with regard to an object class may require considerable experimentation. If the proposed range is too long, the state space increases and renders the entire concept meaningless. On the other hand, if the range is too short, the agent may not have enough information to take an optimal action.

Another limitation of this design is that frequent changes in subtasks may confuse the agent and hamper its ability to choose optimal actions. The agent may encounter cases when it has to *abandon* a selected subtask if it enters a state where a competing subtask has higher priority. Since the agent was unable to finish the subtask it chose earlier, it becomes difficult for the agent to judge the utility of choosing that subtask.

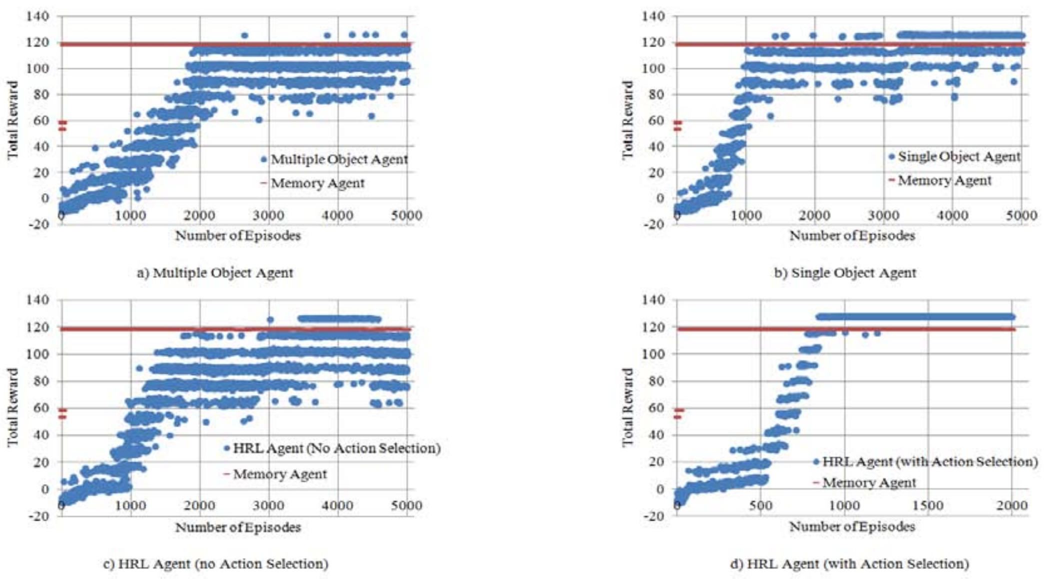

Figure 4. Performance on difficulty level 1, seed 121 of Infinite Mario. All results have been averaged over 10 runs.

VII. Conclusion and Future Work

The *Infinite Mario* domain is characterized by continuous, enormous state-action spaces and complex relationships between participating entities which encourage good decision making. To learn near optimal policies, the ability of agents to constrain state and action spaces is often crucial.

Through this work, we combine an object-oriented representation with hierarchical reinforcement learning to bring about a reduction in state and action spaces in the Mario domain. We also show that introducing a hierarchy in action selection in the above framework improves performance.

In this domain, the game environment contains multiple sources of rewards. This translates into multiple competing sub goals for the agent. In future, we would like to investigate designs where the agent can learn the effect and utility of its actions on multiple competing subtasks. Another major limitation of this work is that the proposed agent is unable to transfer previously acquired knowledge to identify optimal actions in related (but not necessarily *similar*) states. We plan to study function approximation schemes that can represent the states and actions as a combination of features. We are interested in approaches that can represent domain knowledge as a feature of a state and not merely as a tool for constraining action spaces.

References

[1] B. Tanner and A. White, "RL-Glue: Language-indepenent software for reinforcement learning experiments," *The Journal of Machine Learning Research,* vol. 10, pp. 2133-2136, September 2009.

[2] R. Sutton and A. Barto, Introduction to reinforcement learning, 1st Edition ed., MIT Press, 1998.

[3] T. G. Dietterich, "The MAXQ method for hierarchical reinforcement learning," in *the Fifteenth International Conference on Machine Learning*, 1998.

[4] R. S. Sutton, D. Precup and S. Singh, "Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning," *Artificial Intelligence,* vol. 112, pp. 181-211, 1999.

[5] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems [NIPS-97]*, 1997.

[6] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," in *Discrete Events Dynamic Systems: Theory and Applications*, 2003.

[7] J. Laird and M. van Lent, "Human-Level AI's Killer Application: Interactive Computer Games," in *the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000.

[8] K. Driessens, "Relational reinforcement learning," 2004.

[9] B. Marthi, S. Russell, D. Latham and C. Guestrin, "Concurrent hierarchical reinforcement learning.," in *The 20th National Conference on Artificial Intelligence*, 2005.

[10] M. Ponsen, P. Spronck and K. Tuyls, "Hierarchical reinforcement learning with diectic representation in a computer game," in *the 18th Belgium-Netherlands Conference on Artificial Intelligence*, 2006.

[11] C. Diuk, A. Cohen and M. Littman, "An object oriented representation for efficient reinforcement learning," in *the 25th International Conference on Machine Learning*, 2008.

[12] S. Mohan and J. Laird, "An object-oreiented approach to reinforcement learning in an action game," in *the 7th Artificial Intelligence for Interactive Digital Entertainment Conference*, 2011.

[13] G. Rummery and M. Niranjan, "Online Q-Learning using connectionist systems," 1994.

[14] [Online]. Available: http://2009.rl-competition.org/mario.php.