High-level Reinforcement Learning in Strategy Games

Christopher Amato *
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
camato@cs.umass.edu

Guy Shani *
Department of Computer Science
Ben-Gurion University
Beer-Sheva 84105 Israel
shanigu@bgu.ac.il

ABSTRACT

Video games provide a rich testbed for artificial intelligence methods. In particular, creating automated opponents that perform well in strategy games is a difficult task. For instance, human players rapidly discover and exploit the weaknesses of hard coded strategies. To build better strategies, we suggest a reinforcement learning approach for learning a policy that switches between high-level strategies. These strategies are chosen based on different game situations and a fixed opponent strategy. Our learning agents are able to rapidly adapt to fixed opponents and improve deficiencies in the hard coded strategies, as the results demonstrate.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Virtual agents, Reinforcement Learning, Video games

1. INTRODUCTION

Most multi-player video games are distributed with a built in artificial intelligence player that allows humans to play against the computer. Building such players is a complicated task because the AI player has to be challenging, but the game still has to be winnable by the human. Modern games often supply a rich environment with a multitude of world features that may be important and possess a rich set of possible decisions that players must make. Because creating an AI system for a video game does not require considerable hardware resources, yet may require contributions from many different research areas in order to produce a realistic system, it has been proposed as an accessible testbed for building human-level AI systems [5].

Strategy games are an important and difficult subclass of video games. In games such as Warcraft [1] and Civilization[2] players build cities, train workers and military units, and interact with other human or AI players. The goal of these games is to make the best use of limited resources to defeat the opposing players. The large state space and action set, uncertainty about game conditions as well as multiple cooperative and competitive agents make strategy games realistic and challenging.

In this paper, we use Civilization IV as our testbed. Civilization IV is an extremely complex strategy game in which players evolve a culture through the ages, starting in 4000BC and ending in 2050AD. Each player becomes a nation leader and over a series of turns, cities must be built and managed, armies created and technologies researched. Besides the large scope of the game, what sets Civilization apart is that there are many paths to victory. These include the traditional military domination as well as cultural, diplomatic, technological and time victories. The creators of Civilization IV made a considerable effort to build very different AI strategies that attempt to win the game by achieving one or more of these goals.

Perhaps the most obvious way to model an AI player in a strategy game is to use a game-theoretic model. In this paper, however, we choose to take a single agent approach which learns a best response strategy against a fixed player. We believe that it is a reasonable assumption due to both anecdotal and experimental evidence showing that humans often play fixed or slowly changing strategies. Many humans approach complex multi-stage games by playing a reasonable strategy and will only make isolated changes when a failure is recognized. Even in simple games, it has been shown that humans often consider simplistic models of their opponents such as that they are playing against a fixed policy [9]. This results in strategies that may not be responsive to changes in opponents policies. In more complex games, such as strategy games, it is likely that humans will play equally or even less complex policies. While multiagent (game-theoretic) learning may model these situations more accurately, single agent learning may perform well due to increased scalability and the ability to capture the important aspects of these games.

Hence, we use a single agent reinforcement learning approach [10] to learn a policy for switching high-level strategies under the fixed opponent strategy assumption. Our set of candidate strategies is the set of pre-designed world leader personalities, which may be war-seeking, culture-oriented, or expansion-directed. Assuming that each such personality is favorable in different circumstances, we learn when it is

*This work was completed while both authors were at Microsoft Research in Redmond, WA

[1]www.blizzard.com/us/war3/

[2]www.civiv.com/

Cite as: High-level Reinforcement Learning in Strategy Games, Christopher Amato and Guy Shani, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

Figure 1: Screenshot of troops and cities in Civilization IV

Figure 2: Screenshot of a negotiation in Civilization IV

best to use which personality. This approach allows us to leverage the existing low-level knowledge and learn a higher quality policy.

We perform this learning based on state features such as the difference in military strength between the player and opponent(s), the amount of unoccupied land remaining etc. This allows the agent to learn a policy by evaluating the different strategies given the current game state and choosing the one that performs best in each given situation. We experiment with a set of basic reinforcement learning methods [11], such as Q-learning [13] and model-based Dyna-Q [10], using a fixed set of state features. We demonstrate that even in this complicated game, reinforcement learning can be used to improve hard-coded AI players.

The remainder of the paper is organized as follows. We first provide background on Civilization IV as well as our learning framework which is based on Markov decision processes (MDPs) and reinforcement learning. We then discuss our approach for using reinforcement learning in Civilization IV. In section 4, we describe our experimental results, showing that performance can be increased after a small number of learning episodes when playing against a fixed policy. Finally, we supply an overview of the related work on AI in video games and then conclude.

2. BACKGROUND

We first discuss the Civilization IV game, and then provide an overview of reinforcement learning in general and Markov decision processes in particular.

2.1 Civilization IV

The testbed we chose was the turn-based strategy game Civilization IV. As mentioned above, Civilization IV is a very large and complex game where players become a nation leader and evolve their civilization in an attempt to defeat a single or multiple enemies. The player interacts with other leaders through war, commerce, technology exchange and pacts. Movement of troops can be seen in Figure 1, while a common negotiation can be seen in Figure 2.

In Civilization, the player has a very large set of possible actions. For example the player can build one of dozens of buildings and units in each of his cities, research a huge tree of technologies, move units between cities and attack the opponent cities. In this paper we choose to take a high-level view of the game, allowing the game to automatically handle these low-level decisions, and focus only on choosing a high-level strategy for winning the game.

The built-in AI strategies for Civilization IV are created in the form of historic leader personalities. For example, Genghis Khan is war-seeking, while Gandhi attempts to win through cultural or diplomatic leadership. These leaders have two types of differences. First, as we mentioned previously, each leader has a different personality that emphasizes a different strategy for winning the game. Second, leaders have different sets of bonuses, such as special units that only they can create, reduced maintenance costs for buildings, or stronger army units. The designers of the game have matched each personality with appropriate bonuses that are beneficial under the personality strategy. The game designers have made these choices in order to give players a feel of these historic leaders, but as these strategies follow a "personality", they may not be the best method for winning the game. While imperfect, we note that these built-in AI leaders are extremely hard to win against. In fact, the humble authors admit that they were unable to win the game after many hours of gameplay at the mediocre "Prince" level, which we experimented with.

It is reasonable that different situations in the game may require different personalities to handle. For example, when the world is still unexplored, it may be beneficiary to use a personality that emphasizes growth, and when the opponent becomes weak it may be appropriate to become war-seeking, build an army and crush its civilization. While the initial bonuses are fixed, humans often change their personality and thus their strategy for winning the game given the conditions they observe. Therefore, our approach seeks to create a more intelligent, human-like opponent.

While the specific details of the world state in Civilization IV are often hidden from the players, through the so-called fog of war, many global details are available. The player can

at any time view a set of scores for military power, technological advancement, population and so forth for each of the other players. These scores can help the player to understand its relative weaknesses and strengths and make educated decisions. We make use only of these available world features, thus creating "fair" automated players that operate under the same limitations that a human has. The game also synthesizes a score from all these various components, which we will use as the basis for rewarding the player.

One of the main reasons that Civilization IV was chosen was because the game developers have published a large portion of the game source code as an SDK[3]. This SDK allows adding new civilizations, units, and buildings as well as changing the gameplay and AI behavior. We used this publicly available SDK to interact with the game and implement our various learning procedures.

2.2 Markov decision processes

For learning, we choose to use Markov Decision Processes (MDPs) [6] as the general framework. MDPs are a common method for modeling sequential decision-making with stochastic actions. We learn a policy for an MDP through reinforcement learning approaches.

We represent the learning problem as an MDP, defined as a tuple $\langle S, A, P, R \rangle$ with:

- S, a finite set of states with designated initial state s_0.
- A, a finite set of actions.
- P, a set of state transition probabilities: $P(s'|s,a)$, the probability of transitioning from state s to s' when action a is taken by the agent.
- R, a reward function: $R(s,a)$, a real-valued immediate reward for taking action a in state s.

An MDP unfolds over a series of steps. At each step, the agent observes the current state, s, chooses an action, a, and then receives an immediate reward that depends on the state and action, $R(s,a)$. The agent begins in the initial state s_0, which is assumed to be known. The state transitions according to the distribution P as given above and the process continues. The goal is to find a policy, which is a mapping, π, from states to actions, that maximizes the sum of rewards over the steps of the problem. In this paper, we consider the infinite horizon problem which unfolds over an infinite number of steps. To maintain a finite sum, a discount factor $\gamma \in [0,1)$ is used. The value of a policy π at state s can be calculated as:

$$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s} P(s'|s,\pi(s)) V^{\pi}(s')$$

Where $\pi : S \rightarrow A$ is a mapping from states to actions according to policy π.

2.3 Reinforcement learning

When we do not know the transition and reward models, we can use reinforcement learning methods to learn a policy. Reinforcement learning is an approach to learn policies for agents acting in an unknown stochastic world, observing the states that occur and the rewards that are given at each step [11].

[3]www.firaxis.com/downloads/Patch/CvGameCoreDLL_v161.zip

2.3.1 Q-learning

The first approach we use is Q-learning [13]. This method updates the value of a state-action pair after the action has been taken in the state and an immediate reward has been received. Values of state-action pairs, $Q(s,a)$ are learned because the resulting policy is more easily recoverable than learning the values of states alone, $V(s)$. Q-learning will converge to an optimal value function under conditions of sufficiently visiting each state-action pair, but often requires many learning episodes to do so [14].

When an action a is taken in state s, the value of a state-action pair, or Q-value, is updated as

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma Q(s') - Q(s,a) \right)$$

where $\alpha \in [0,1]$ is the learning rate, r is reward that is observed, γ is the discount factor, s' is the next state, and $Q(s) = \max_a Q(s,a)$.

The actions are taken according to some exploration policy, such as an ϵ-greedy approach. This method chooses the action that maximizes the Q-value with probability $1-\epsilon$ and a random action with probability ϵ. These policies are chosen in order to balance the exploration of uncertain states and actions with the exploitation of the current policy. It is also common in a stationary environment to decay the exploration rate (ϵ) as a policy is learned as another way to begin to deal with this tradeoff.

In multiagent domains, Q-learning is no longer guaranteed to converge due to the environment no longer being stationary. Nevertheless, it has been shown to be effective [8, 12]. When the other players use fixed policies, they can be considered part of the environment and the problem again becomes an MDP. In this case, the Q-learner will learn a best response policy. Thus, Q-learning is optimal in the case when the other players do not change policies and can be robust to situations in which they do.

2.3.2 Model-based Q-learning

Q-learning is a model-free method. That is, it learns a policy directly, without first obtaining the model parameters — the transition and reward functions. An alternative is to use a model-based method that learns the model parameters and uses the model definition to learn a policy.

Learning a model consists of learning the transition probabilities and reward values for each state and action. If a good model is learned, an optimal policy can be found by planning methods because the model parameters are now known. Rather than first building a correct model and then finding a policy from that model, we learn the model and the Q-values at the same time with the Dyna-Q approach [10].

Dyna-Q can learn the Q-values more quickly than Q-learning by using the model to generate learning experiences and does not require a model to be fully learned before a policy can be found. Thus, the agent learns both the Q-values and the model through acting in the environment. The model is then used to simulate the environment and the Q-values are updated accordingly. As the model becomes a better representation of the problem, the Q-values will be more accurately updated and convergence will occur more quickly. The Dyna-Q algorithm operates exactly like Q-learning, except for the addition of model learning

Algorithm 1: Dyna-Q

input : current Q-values, Q, immediate reward r, state s and action a
output: updated Q-values, Q
begin
  $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
  $P(s'|s,a) \leftarrow updatePAverage(s,a,s')$
  $R(s,a) \leftarrow updateRAverage(s,a)$
  for $i = 0$ **to** $numIter$ **do**
    $s' \leftarrow randomPreviouslySeenS()$
    $a' \leftarrow randomPreviouslyTakenA(s')$
    $s'' \leftarrow sampleFromModel(s',a')$
    $r' \leftarrow fromModel(s',a')$
    $Q(s',a') \leftarrow Q(s',a') + \alpha(r + \gamma Q(s'',a'') - Q(s',a'))$
  return Q
end

and an offline planning phase at each step. These additions allow learning to take place without an explicit model learning phase because the model and Q-values are learned simultaneously. Nevertheless, the inclusion of a model allows learning to be hastened.

Dyna-Q is shown in Algorithm 1. First the regular Q-learning update takes place and the probability and reward models are updated as averages given the new information. That is, the transition probability is the number of times s' occurs after being in state s and taking action a divided by the number times the agent was in state s and chose action a:

$$P(s'|s,a) = count(s,a,s')/count(s,a)$$

The reward value is the average of the rewards received in state s after choosing action a:

$$R(s,a) = (count(s,a) * R(s,a) + r)/(count(s,a) + 1)$$

The model sampling occurs in the *for* loop. For some designated number of iterations the model is sampled and the Q-values are updated accordingly. This is done by first uniformly choosing a state that has been previously encountered, s'. An action, a', that has been taken in s' is then uniformly chosen and based on the transition model, a resulting state s'' is sampled. The reward for s' and a' is then found from the reward model. These values are then used to update the appropriate Q-values.

2.3.3 Factored state representations

In many environments states can be described as an assignment to state features [4]. If there is some independence between feature transitions or rewards, this representation can provide significant power in learning over fewer episodes. However, assuming independence between features that are in fact dependent can cause us to learn an improper model and thus an imperfect policy.

Assuming that features transition independently, we can write:

$$P(s' =< f'_1, ..., f'_n > |s,a) = \prod_{i=1}^{n} P(f'_i|s,a)$$

where $P(f'_i|s,a)$ is the probability of feature f_i after action a has been taken in state s.

The transition functions for each one of these features can then be learned independently of the others in Dyna-Q. That is, rather than learning $P(s'|s,a)$, we learn separate functions for each $P(f'_i|s,a)$. This reduces the transition model parameters from $|S|^2|A|$ to $|F||S||A|$, where $|F|$ is the number of features. Thus, we require fewer learning episodes in order to learn the model parameters, leading to faster learning.

3. A REINFORCEMENT LEARNING APPROACH FOR CIVILIZATION IV

The basis of our approach is to learn a policy for switching between high-level strategies. During each game, the agent observes certain current game information. Based on this information, the agent chooses a strategy — a leader personality — for the next step. At this next step, the agent receives a reward, again observes the new game information and chooses a strategy again. If the game is played again, the learning continues. Below, we consider a game with only two players, but the approach could be generalized to any number of players.

3.1 Learning in Civilization

As we explain above, we focus here on the question of properly selecting a strategy given the current game conditions. This is done by learning the value of playing each of the different leader personalities in different game scenarios. Given these values, we can choose the personality with the highest value in the current condition. This approach can produce an AI system that performs better against a human opponent and allow game developers to automatically mix a set of fixed strategies in order to form a higher quality policy.

We assume here that a human will play a fixed strategy (however complicated), as was discussed previously. Even against a fixed opponent, it is crucial to learn quickly. While we assume the game will be played repeatedly, we cannot expect a human to play hundreds of games while waiting for the AI to improve. Thus, we develop an approach that does not require a large number of training episodes in order to produce an improved policy.

3.2 Modeling Civilization as an MDP

Because choosing a high-level strategy may require several game turns to bear fruit, we allow strategy switching (an MDP step) only every few turns. The new strategy is allowed to run for this fixed number of turns, after which we observe the outcome. In our case, a decision is made every 10 turns, resulting in at most 46 steps per game. The details of how the states, actions and rewards are represented are explained below.

It should be noted that these parameters were chosen based on educated guesses after playing the game, but extensive analysis was not conducted. It is quite likely that these could be improved, which would also improve the performance of the resulting algorithms. Our goal was to choose a simple and general model that is not dependent on "tweaking" of the parameters.

3.2.1 State space

We define the state space with a set of four state features: population difference, land difference, military power difference and remaining land. We call these features f_1, f_2, f_3 and f_4 and calculate their values based on the scores provided in the game. These features were chosen because they provide very general information about the relative status of a civilization. Also, each player has access to these values and can compute the resulting features. Because we consider games with two players, the difference features are therefore the difference in score between the two players, while the remaining land is found by subtracting the land currently occupied by both players from the total amount of land in the game.

States are then discretized over the possible values. For population, land and power differences, the feature was given one of three values based on the difference in scores. That is

$$f_i = \begin{cases} 2, & \text{if } diff > 10 \\ 1, & \text{if } -10 < diff < 10 \\ 0, & \text{if } diff < -10 \end{cases}$$

where *diff* represents the difference in value between the agent and the opponent. For example, if the difference in power between the players is 26, $f_3 = 2$. The remaining land feature has three values as well, determined by whether there is over 50% of land remaining, between 20% and 50% or less than 20%. Again, this discretization was chose to be general, but increased performance could likely be achieved by using different intervals for each feature. Combining these features produces 81 possible states.

3.2.2 Action space

As we explain above, an action is a choice of a new strategy for making low-level decisions. We use the set of built-in personalities as the allowed strategies. We limited our action space to four leaders: George Washington, Frederick II, Mahatma Gandhi and Genghis Kahn. These leaders were chosen because they possess each of the eight possible personality traits and have diverse preferences for war, buildings etc. Washington is Financial and Organized, Frederick is Creative and Philosophical, Gandhi is Industrious and Spiritual and Genghis Kahn is Aggressive and Expansive. These traits, along with other heuristics, define preferences for each leader. These leaders can be seen as they appear in the game in Figure 3.

3.2.3 Reward model

We define the immediate reward given at each step based on the score provided by the game. That is, the immediate reward is the difference in total score between the agents. This measures how the agent is playing in relation to the opponent. While it is possible to lose the game while having a higher score than the opponent, the score is obviously highly correlated with the final outcome of the game. This reward was chosen in pursuit of our goal to produce a player that adapts its policy and wins more often. We define the difference in score as

$$thisStepScore = myTotalScore - yourTotalScore$$

3.3 Learning approaches

Figure 3: Leaders in Civilization IV (clockwise from top left): Frederick II, Mahatma Gandhi, Genghis Kahn and George Washington

In this paper we used basic reinforcement learning methods. This was done to demonstrate the applicability of reinforcement learning to the problem of strategy selection. Thus, it is likely that more advanced methods will provide better results. The approaches we used were those discussed above: Q-learning, Dyna-Q, and Dyna-Q over the factored state space. The code implementation for these methods, which provides a framework for reinforcement learning in Civilization IV is available at:
http://www.cs.umass.edu/∼camato/Civ4.html

4. EXPERIMENTS

To demonstrate the applicability of our approaches, we performed learning against a single AI opponent playing the fixed policy provided by the game. We seek to determine if the game policies can be improved by our learning methods. We note again that at the "Prince" level which we experimented with, winning against the built-in AI is very challenging. Each game was played in the small "duel" sized map with the standard game speed, starting era, water level and climate. Our learners started with a random policy and learned over 50 and 100 training episodes. 500 testing episodes were then used to determine the quality of the learned policy.

The parameters used in each of our learning algorithms were $\alpha = 0.25$, $\gamma = 0.9$, $\epsilon = 0.2$ for training, while $\epsilon = 0.0$ was used for testing. For both the flat and factored versions of Dyna-Q, 25 steps of learning from the model were used at each step of the problem. It is also worth noting that all of these methods used very little computation time. The most computationally intensive method, the flat model-based Dyna-Q, added only about one second at each

problem step. Thus, all of these learning methods would be transparent to an end user playing the game.

For the experiments, we provide results for playing Frederick against Washington and Gandhi against Genghis Kahn. We selected these pairs because they represent very different approaches for winning the game. In addition to the preferences and characteristics of the leaders, initial bonuses are given in the form of beginning technology and unique troops. These bonuses remain fixed even when the personality changes.

To determine the improvement over these fixed strategies in the game, we first determined the percentage of games each leader would win against the chosen opponent. This standard policy is referred to as *fixed* in the figures and tables below, while a random policy which randomly chooses a leader personality at each step is called *random*. These were run 1000 times to improve the precision of the resulting estimates. To determine if each learner's policy is better than that of the fixed or random policies, we compare them using a one-tailed Welch's t-test. This test accounts for possibly different variances, but due to large sample sizes, similar tests will likely have similar values.

Frederick vs. Washington

When playing Frederick against Washington without any learning, Frederick won 54.1% of the time. A random policy started as Frederick won 51.2% of the time. Figure 4 shows the results of the different learning methods after 50 and 100 steps of learning and the mean number of games won by the fixed and random policies. Confidence intervals of 95% are also included in each bar of the figure. Table 1 describes the statistical significance of these results. The t-values using Welch's t-test are provided for both the random and fixed polices along with the one-tailed p-values. These p-values are rounded up using standard intervals to increase readability. We consider results equal to or below 0.05 to be statistically significant and as a result, present them in bold font.

The figure shows the percentage of games won by Q-learning (Q), model-based learning (M) and learning with the factored model (FM) after 50 and 100 learning episodes. In each case, the percentage won after learning was higher than that of the fixed and random policies. When statistical significance is also considered, model learning for both 50 and 100 episodes as well as the factored model after both 50 and 100 episodes are statistically significantly higher than the random policy with 95% probability or higher. When compared to the fixed policy, model-based learning in both cases and the factored model after 100 episodes are statistically significantly higher. The other results are suggestive, but may require more learning before becoming significantly better than the fixed policy played by the game AI.

The results are in line with common expectations. The model allows the learner to improve at a faster rate, permitting the model-based learner to perform well with a small number of learning episodes. Likewise, the factored model learner will learn even more quickly, but because the independence assumption does not capture some aspects of the state space that are captured by the flat model learner, solution quality is slightly lower.

The resulting policies that are learned display many tendencies that one would expect in the game. For instance, where there is a power advantage for the learner, Genghis Kahn will often be chosen. When the game is even and almost all the land has been taken, Washington is often chosen to stabilize the economy of the civilization and provide a balanced endgame.

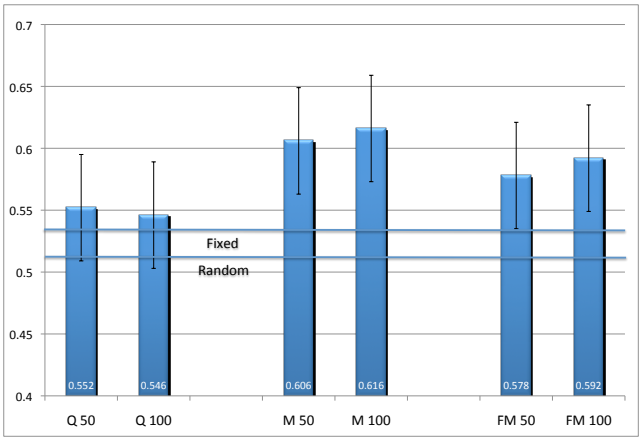

Figure 4: Results of Frederick learning to play against Washington

	Random		Fixed	
	t-value	p-value	t-value	p-value
Q50	1.46	0.10	0.40	0.35
Q100	1.24	0.15	0.18	0.45
M50	3.48	**0.0005**	2.41	**0.01**
M100	3.86	**0.0001**	2.79	**0.005**
FM50	2.43	**0.01**	1.36	0.1
FM100	2.95	**0.005**	1.88	**0.05**

Table 1: The significance of results for Frederick vs. Washington. Results significant at the 0.05 level or beyond are in bold.

Gandhi vs. Genghis Kahn

In these games, the fixed policy won 73.1% of the games, while a random policy won 77.6%. The high winning percentage is likely because Gandhi's bonuses are stronger, but we can see that Gandhi's personality is often not the best one to play in this situation. Figure 5 shows the results of each of the learners after 50 and 100 learning episodes as well as the mean number of games won by the fixed and random policies. We also provide 95% confidence intervals for each case. Table 2 provides the statistical significance of these results.

Here, the model learning (M) after both 50 and 100 learning episodes and the factored model learner (FM) after both 50 and 100 episodes are statistically significantly better than the random policy. All learners are statistically significantly better than the fixed policy. This is partly due to the fact that the learners are initialized with a random policy. These results show that even a high quality initial policy can be quickly improved with reinforcement learning.

The trends of the different learners are similar to above.

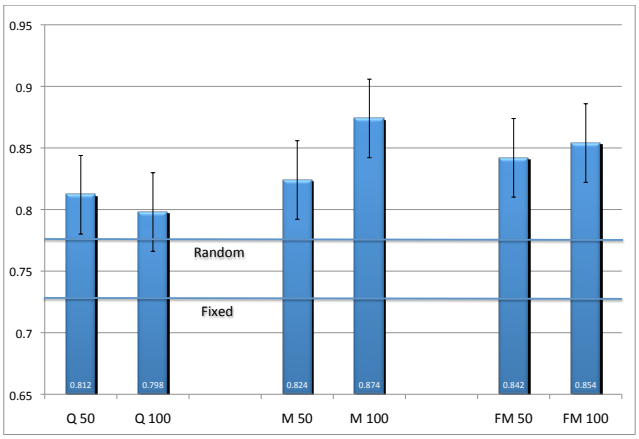

Figure 5: Results of Gandhi learning to play against Genghis Kahn

	Random		Fixed	
	t-value	p-value	t-value	p-value
Q50	1.63	0.10	3.61	**0.0005**
Q100	0.99	0.20	2.94	**0.005**
M50	2.23	**0.05**	4.21	**0.00005**
M100	4.93	**0.00001**	7.00	**<0.000001**
FM50	3.14	**0.001**	5.15	**<0.000001**
FM100	3.79	**0.0001**	5.82	**<0.000001**

Table 2: The significance of results for Gandhi vs. Genghis Kahn. Results significant at the 0.05 level or beyond are in bold.

The Q-learner performs the worst, while the model-based learners learn more quickly and win more often. The factored model learner wins more often after 50 episodes, but after 100, the flat model-based learner overtakes it. Again, this is likely due to the fact that the features do not completely characterize the state space.

In the resulting policies, many choices are the same as above, but many are different. Genghis Kahn is sometimes chosen when there is a power advantage, but he is chosen more often when there are advantages in land and population, but when power is even or lower than the opponent. Presumably, this is to increase the power of the civilization to better attack or defend in the future. Also, Washington is chosen in a large number of instances. These include when there is a power advantage or when there is little land remaining, but the learner is leading in other areas. This seems similar to the above case when Washington is used at the end of the game to strengthen the civilization.

5. RELATED WORK

Many researchers have studied methods for planning and learning in video games in general and strategy games in particular. We review below some related work in planning, learning and game theoretic approaches for AI systems in video games.

Researchers have used planning to explore the resource gathering problem of a Warcraft II open source clone, Wargus [2]. The authors seek to minimize the number of steps needed to reach certain resource goals (such as gathering 100 gold, or gathering jointly 50 gold and 50 trees, or training 10 troops). Classical planning is used with the ability to "lock" resources such as workers (peasants) to prevent conflicting plans. Issues such as action durations and different numbers of agents are also addressed by alternating between planning and scheduling, which orders the actions to allow for concurrency. Subgoals (such as increasing the amount of a specific resource) and other heuristics (such as adding peasants) are used to speed up planning and increase resource production. The approach performs about as well as a human and better than the other planning algorithms used in a set of small Wargus scenarios.

Planning was also used to look at a simplified assault scenario in which groups of troops need to defeat other troops or an enemy base, again in the Wargus game [1]. Defending your own base or sequential waves of attack are not considered. Online planning is done at each decision epoch, which is defined as until the troops become idle again. Planning is done at the group level (assuming groups of troops for the player and opponent) using UCT, a Monte-carlo planning algorithm. Only two actions are used, "join group" or "attack." Hand tuned parameters were used for estimated action durations and determining the effectiveness of actions. On some small scenarios, their planner does better than heuristics and can perform better than a human player.

Another approach also explored the army deployment problem in real-time strategy games, but from a game-theoretic point of view [7]. The authors assume a set of strategies is given (such as quickly attacking, only defending, etc.) and that the player and opponent will choose from these. All pairs of strategies are simulated to determine the success of each, and these simulated values are used to construct a payoff matrix. A Nash equilibrium strategy is then found. Some heuristics are used to make the simulations run faster and again, the troops are separated into groups. This approach was tested against each strategy type as well as a random player. The Nash equilibrium strategy did about as well as one fixed strategy (purely defending), but better than the rest.

One of the few learning approaches in video games used reinforcement learning to learn a policy in a fighting game called Tao Feng [3]. Starting from a random policy, a small number of macro-actions were used (such as punching forward and back, kicking forward and back, blocking for 10 steps, moving in different directions, etc.) and a policy was learned using a variant of Q-learning called Sarsa [10]. A linear function approximator was used with features that consisted of the distance to opponent, whether obstacles were present to the sides, the previous opponent action and whether the player was in the air, standing or crouching. After a large number of trials (over 3000), a policy was learned that could defeat the built-in AI.

Reinforcement learning has also been used in Civilization IV [15]. In this paper, Q-learning was used to learn how to place new cities in the map. The learning took place based on states that consisted of the x and y coordinates of current cities as well as the order they were built. When cities were generated by the standard game AI, the learner could place

them on any land square. The reward for city placement was defined as the difference in game score since placing the last city. In a shortened version of the game (50 turns), the learner could outperform the standard AI at placing up to two cities. This took place after a large number of training episodes (over 2000).

Comparison with our work

Our approach shares some similarities with earlier work, but seeks a different goal. Unlike the planning approaches for strategy games, our approach tackles the complete problem (rather than just resource gathering or attacking). We also use learning to adapt a policy based on the conditions of the game and the strategy of an opponent. In contrast to game theoretic approaches, we learn to switch the strategy during the game instead of choosing a fixed policy for the whole game. And unlike previous learning methods, we tackle the more complex problem of a complete strategy game. Even though we are solving this complex problem, we also strive to learn with many fewer learning episodes. Also, while the learning used in [15] may not generalize to other games or even changes in a given game map, ours can generalize to many different game scenarios and problem types.

6. CONCLUSION

In this paper, we studied a reinforcement learning approach for switching high-level behaviors in a complex strategy game, Civilization IV, assuming an opponent is playing a fixed strategy. Three different learners are used: Q-learning, model-based Q-learning (Dyna-Q) and a factored model version of Dyna-Q. These approaches were able to quickly learn higher quality policies, often in as little as 50 training episodes. We showed that these policies could win more often than both random policies and the hand-tuned (fixed) policy of the game AI.

This shows that reinforcement learning can provide powerful tools which allow agents to adapt and improve quickly, even in complex scenarios such as strategy games. These techniques are very general and may also be applicable in other domains. Example domains include the stock market or algorithm portfolios where expert strategies are available or low level policy details cannot be changed.

In the future, we are interested in extending this line of research in several ways. For instance, because the factored model improves the value of a policy more quickly, but value then stops improving it would be interesting to learn the factored model first and then use this policy to continue learning with the flat model. This should decrease learning time and increase solution quality. Also, we intend to examine more advanced reinforcement learning and machine learning methods in general for strategy games. Techniques that bias exploration towards more promising parts of the state space and the use of feature extraction to learn the set of features could improve the learning time and scalability of our methods. We hope that others will also use our RL toolbox for Civilization IV to explore more ways of using reinforcement learning in this rich domain.

7. ACKNOWLEDGEMENTS

The authors would thank the members of the Machine Learning and Applied Statistics group at Microsoft Research, Redmond for their helpful comments on this work.

8. REFERENCES

[1] R.-K. Balla and A. Fern. UCT for tactical assault planning in real-time strategy games. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009.

[2] H. Chan, A. Fern, S. Ray, N. Wilson, and C. Ventura. Online planning for resource production in real-time strategy games. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, Providence, RI, 2007.

[3] T. Graepel, R. Herbrich, and J. Gold. Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, Reading, UK, 2004.

[4] T. Hester and P. Stone. Generalized model learning for reinforcement learning in factored domains. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2009.

[5] J. E. Laird and M. van Lent. Human-level AI's killer application: Interactive computer games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.

[6] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

[7] F. Sailer, M. Buro, and M. Lanctot. Adversarial planning through strategy simulation. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, Honolulu, HI, 2007.

[8] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147–166, 1995.

[9] D. O. Stahl and P. W. Wilson. On players' models of other players: Theory and experimental evidence. *Games and Economic Behavior*, 10:218–254, 1995.

[10] R. S. Sutton. Dyna, an integrated architecture for learning, planning and reacting. In *Working Notes of the 1991 AAAI Spring Symposium on Integrated Intelligent Architectures*, 1991.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[12] G. Tesauro and J. O. Kephart. Pricing in agent economies using multi-agent Q-learning. *Autonomous Agents and Multi-Agent Systems*, 5(3):289–304, 2002.

[13] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[14] C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[15] S. Wender and I. Watson. Using reinforcement learning for city site selection in the turn-based strategy game Civilization IV. In *Proceedings of CIG'08: IEEE Symposium on Computational Intelligence and Games*, Perth, Australia, 2008.