IMPLANT: An Integrated MDP and POMDP Learning Agent for Adaptive Games

Chek Tien Tan and **Ho-lun Cheng**
Department of Computer Science
National University of Singapore
Computing 1 (COM1), Singapore 117590
{tanchekt, hcheng}@comp.nus.edu.sg

Abstract

This paper proposes an Integrated MDP and POMDP Learning AgeNT (IMPLANT) architecture for adaptation in modern games. The modern game world basically involves a human player acting in a virtual environment, which implies that the problem can be decomposed into two parts, namely a partially observable player model, and a completely observable game environment. With this concept, the IMPLANT architecture extracts both a POMDP and MDP abstract model from the underlying game world. The abstract action policies are then pre-computed from each model and merged into a single optimal policy. Coupled with a small amount of online learning, the architecture is able to adapt both the player and the game environment in plausible pre-computation and query times. Empirical proof of concept is shown based on an implementation in a tennis video game, where the IMPLANT agent is shown to exhibit a superior balance in adaptation performance and speed, when compared against other agent implementations.

Introduction and Related Work

In modern game worlds, the game mechanics are often defined stochastically to represent uncertainties for interesting gameplay (for example agent actions in an RPG game are defined with hit and miss chances). Virtual agents or non-player characters (NPCs) in the game need to act plausibly in such uncertain environments to provide a good gameplay experience to the player. Moreover, these agents also need to consider various types and styles of different players such that the experience is appropriately customized. However, these aspects of the players are not directly observable.

Therefore a motivation to this paper is the need for theoretically sound methods for game agents to act adaptively in an uncertain and partially observable environment. Cutting edge decision-theoretic methods are often avoided in modern game AI applications primarily due to high computation overhead. Partially Observable Markov Decision Processes (POMDPs) (Kaelbling, Littman, and Cassandra 1998) represent the state of the art in decision theory that naturally models a game agent acting in an uncertain environment with unobservable attributes. Unfortunately, POMDPs solution algorithms suffer the same fate of being computationally intractable (Burago, Rougemont, and Slissenko 1996), especially so in huge modern game worlds.

A related motivation to this paper is the need for a unified agent architecture that adapts to both the player and the game environment. Current research in modern game agent behaviors can be largely split into two camps, one targeted at planning to adapt to the game environment (Remco Straatman and van der Sterren 2006; Hussain and Vidaver 2006; Geramifard, Chubak, and Bulitko 2006; Christian J. Darken 2006; White and Brogan 2006), and the other targeted at planning to adapt to the game player (Charles et al. 2005; Donkers and Spronck 2006; van der Sterren 2006; Thue and Bulitko 2006; Yannakakis and Maragoudakis 2005; Tan and Cheng 2008b). For example Straatman et al (2006) combines level annotation with sensor grid algorithm to allow the agent to dynamically find cover, but leaves the player out of the equation, whereas Tan and Cheng (2008b) devised a generic Tactical Agent Personality (TAP) model that accomplishes player adaptation, but do not consider the game environment at all. This phenomenon can be attributed to the fact that different considerations need to be met for the two problem domains and substantial efforts are needed to reconcile this disjunction. Model-free reinforcement learning (Sutton and Barto 1998) approaches appears to eliminate the modeling problem by not having an explicit model of the environment. However, it suffers from a similar problem of intractability in which an impractical amount of time is usually required for online policy convergence.

To effectively adapt towards both the game environment and player in acceptable computation times, the IMPLANT architecture is presented in this paper. Since the only true partially observable attribute of the game world is the player model, and that the rest of the handcrafted game world are completely observable, the IMPLANT agent extracts a small POMDP model of the player and a large MDP model of the game world. Hence separate action policies can be solved quickly since MDPs are much more tractable than POMDPs. The agent then combines the two policies into a single combined policy whereby a small amount of online learning is performed to ensure that the eventual action is adaptive to any in-game changes. By having the majority of the adaptivity pre-computed, it also ensures that the agent has an AI that works plausibly right out of the box, without having to let the player wait long periods for the learning to converge.

In the remaining sections of this paper, essential background is first given followed by a description of the IMPLANT architecture. The experimental setup is then depicted along with the experimental results as empirical proof of concept. Finally a discussion is made on the results and the paper is concluded along with the plans for future work.

Background of MDP and POMDP

A Markov Decision Process (MDP) can be described as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ where $\mathcal{S}$ is a finite set of world states and $\mathcal{A}$ is a finite set of actions available to the agent. The state-transition function T defines the probability $T(s, a, s')$ of transitioning from state s to state s' when the agent takes action a. The reward function R defines the expected immediate reward $R(s, a)$ received by the agent for taking action a in state s. The discount factor $\gamma < 1$ is used to weigh the rewards according to the time it was received.

A policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, is a mapping that specifies an action to take for each state. The optimal policy, π^*, of an MDP can be found exactly using a method called value iteration, which involves iterating the following Bellman equation:

$$V_{t+1}(s) = \max_{a \in \mathcal{A}} \{R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_t(s')\}, \quad (1)$$

where V_k is the value function that defines the expected total reward with k steps left, and $V_0(s) = R(s)$. In other words, the optimal $t + 1$ horizon policy, π^*_{t+1}, is:

$$\pi^*_{t+1} = \arg\max_{a \in \mathcal{A}} \{R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_t(s')\}, \quad (2)$$

A Partially Observable Markov Decision Process (POMDP) can be described as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma, \mathcal{O}, O, b_0 \rangle$ where $\mathcal{S}$, $\mathcal{A}$, T, R and γ are the same as in an MDP. $\mathcal{O}$ is a finite set of observations, with the corresponding O an observation function such that $O(a, s', o)$ is the probability of making observation o if the agent takes action a and the world transits to state s'.

Now that the state is hidden, the agent needs to keep an internal belief state, b, that summarizes its previous experience. It is basically a vector of probabilities such that $b = \{b(s) | s \in \mathcal{S}\}$, $0 \leq b(s) \leq 1$, $\sum_{s \in \mathcal{S}} b(s) = 1$. b_0 is the agent's initial belief state. Given that the agent makes observation o after taking action a in belief state b, the next belief state b' can be computed using a state estimator function based on Bayes rule:

$$b'(s') = \frac{O(a, s', o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s)}{P(o|b, a)}, \quad (3)$$

where $P(o|b, a) = \sum_{s \in \mathcal{S}} b(s) \sum_{s' \in \mathcal{S}} T(s, a, s') O(a, s', o)$.

To obtain a policy for a POMDP, the agent needs to map each belief state into an action. To solve for the optimal policy, one needs to view the POMDP as a continuous space "belief MDP". With this concept, the transition and reward functions can be converted to their counterparts in this "belief MDP" and value iteration equations similar to those of Equations 1 and 2 can be derived:

$$V_{t+1}(b) = \max_{a \in \mathcal{A}} \{\rho(b, a) + \gamma \sum_{b' \in \mathcal{B}} \tau(b, a, b') V_t(b')\},$$

$$\pi^*_{t+1} = \arg\max_{a \in \mathcal{A}} \{\rho(b, a) + \gamma \sum_{b' \in \mathcal{B}} \tau(b, a, b') V_t(b')\},$$

where $\mathcal{B}$ is the set of all belief states, and τ and ρ are the "belief MDP" transition and reward functions respectively. These equations serve as the basis for POMDP value iteration, which are highly complex due to its continuous nature. Solution algorithms have been based on the fact that the value function of POMDPs are piecewise linear and convex, which can be represented as finite collections of $|\mathcal{S}|$-dimensional vectors, commonly known as α vectors. Still, the problem is intractable when the number of states grow moderately large.

To compute practical solutions in reasonable timings, there have been significant efforts in the development of approximation algorithms. In terms of performance, point-based algorithms have been particularly successful (Pineau, Gordon, and Thrun 2003; Shani, Brafman, and Shimony 2007; Doshi and Roy 2008; Kurniawati, Hsu, and Lee 2008), which basically computes value functions iteratively from sampled points in the belief space. The work in this paper is partly inspired by the idea in Forward Search Value Iteration (FSVI) ((Shani, Brafman, and Shimony 2007)) but approached in a totally different manner. They made use of the MDP to guide their main POMDP value iteration whilst the work in this paper decomposes the problem domain into two parts (MDP and POMDP), obtain the abstract policies separately, then combines them into a single policy.

IMPLANT Architecture

The overall architecture of IMPLANT (Integrated MDP and POMDP Learning AgeNT) is as shown in Figure 1. The central idea is to decompose the game world into an MDP and a POMDP abstract with their respective policies computed. Current states and observations are individually obtained from each abstraction and processed via an Action Integrator (AI) function, which computes a resultant optimal action that the agent performs and affects the underlying game world, which then provides a reward signal that reinforces the action.

The mechanics of the game world as devised by the game designer is normally represented as states and transitions coupled to actions, with the agent behaviors normally driven by some goals. These coincides nicely with the MDP model parameters. The mechanics can therefore be extracted and formatted into the ground MDP specification, denoted as $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$.

Built upon concepts from abstraction theory (Li, Walsh, and Littman 2006), the abstraction process can be formally defined. First a state abstraction function, $\phi : \mathcal{S} \rightarrow \hat{\mathcal{S}}$, can be defined, where $\hat{\mathcal{S}}$ represents the set of abstracted states. Hence $\phi(s) \in \hat{\mathcal{S}}$ is the abstracted state corresponding to the ground state $s \in \mathcal{S}$, and typically, $|\hat{S}| \ll |S|$. Inversely, $\phi^{-1}(\hat{s}) \in \mathcal{S}$ represents the set of states corresponding to the abstract state $\hat{s} \in \hat{\mathcal{S}}$. This function basically means that each abstracted state, $\hat{s}$, is a collection of ground states, s. Now, since the states are changed, the transition function, T, and reward function, R, would also need to be converted

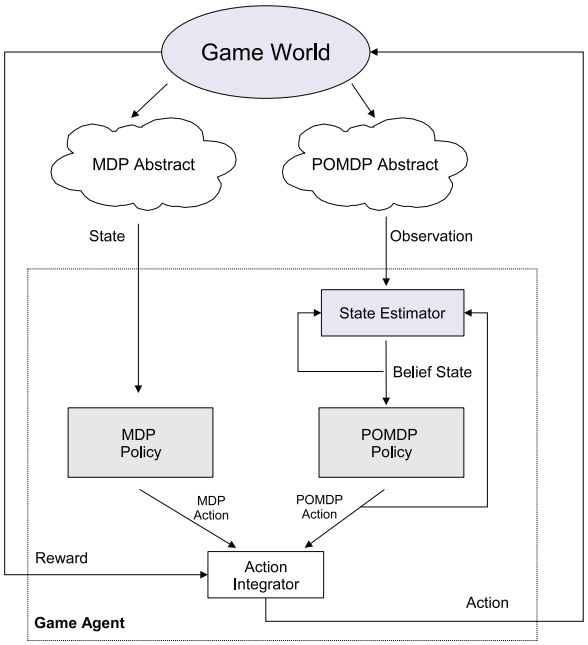

Figure 1: An overview of the IMPLANT architecture. The agent maps a separate MDP and POMDP abstract from the game world and finds optimal policies according to each abstract. For each game state and observation, the agent then obtains a single action via the Action Integrator from the two policies. The action is then reinforced by the reward afterwards.

into their abstracted forms. To ensure that they are well-defined, a weighting function, $W_{\mathcal{S}} : \mathcal{S} \rightarrow [0, 1]$, needs to be imposed, where $\sum_{s \in \phi^{-1}(\hat{s})} W_{\mathcal{S}}(s) = 1$ for each $\hat{s} \in \hat{\mathcal{S}}$.

Additionally, an action abstraction function, $\psi : \mathcal{A} \rightarrow \hat{\mathcal{A}}$, can be defined, where $\hat{\mathcal{A}}$ represents the set of abstracted actions. Similar to state abstraction, a weighting function for actions, $W_{\mathcal{A}} : \mathcal{A} \rightarrow [0, 1]$, is imposed, where $\sum_{a \in \psi^{-1}(\hat{a})} W_{\mathcal{A}}(a) = 1$ for each $\hat{a} \in \hat{\mathcal{A}}$.

Hence the abstract transition function, $\hat{T}$, is defined as follows:

$$\hat{T}(\hat{s}, \hat{a}, \hat{s}') \tag{4}$$
$$= \sum_{a \in \psi^{-1}(\hat{a})} W_{\mathcal{A}}(a) \sum_{s \in \phi^{-1}(\hat{s})} \sum_{s' \in \phi^{-1}(\hat{s}')} W_{\mathcal{S}}(s) T(s, a, s'),$$

where it is relatively straightforward to prove that its probabilities are well-defined ($\sum_{\hat{s}' \in \hat{\mathcal{S}}} \hat{T}(\hat{s}, \hat{a}, \hat{s}') = 1$). Next, the abstract reward function can be defined similarly:

$$\hat{R}(\hat{s}, \hat{a}) = \sum_{a \in \psi^{-1}(\hat{a})} W_{\mathcal{A}}(a) \sum_{s \in \phi^{-1}(\hat{s})} W_{\mathcal{S}}(s) R(s, a). \tag{5}$$

Therefore the MDP abstract, M_{co}, and the POMDP abstract, M_{po}, can now be defined as:

$$M_{co} = \langle \hat{\mathcal{S}}_{co}, \hat{\mathcal{A}}_{co}, \hat{T}_{co}, \hat{R}_{co}, \gamma \rangle, and \tag{6}$$
$$M_{po} = \langle \hat{\mathcal{S}}_{po}, \hat{\mathcal{A}}_{po}, \hat{T}_{po}, \hat{R}_{po}, \gamma, \hat{\mathcal{O}}, \hat{O}, b_0 \rangle,$$

where the subscripts $_{co}$ and $_{po}$ label the MDP and POMDP parameters respectively.

With the MDP and POMDP abstracts defined, they can be solved to produce the optimal policies π_{co} and π_{po} respectively. With the policies generated, the agent needs to decide the current resultant action a to take, computed via the Action Integrator which consists of a two-stage process. The first stage obtains a combined abstract action $\hat{a}$ whereby:

$$\hat{a} = \begin{cases} \hat{a}_{co} \bigcap \hat{a}_{po} & \text{if } \hat{a}_{co} \bigcap \hat{a}_{po} \neq \emptyset, \\ \hat{a}_{co} \bigcup \hat{a}_{po} & \text{else.} \end{cases} \tag{7}$$

This process aims to choose a set of actions optimal to both the POMDP abstract (player model) and the MDP abstract (game environment). In the normal case, this set will be an intersection between $\hat{a}_{co}$ and $\hat{a}_{po}$ (as shown in Figure 2), but in case the intersection is an empty set, the union set provides a backup utility, in which the agent falls back to choose any action that is optimal to either the POMDP or MDP abstract.

The second stage in the AI function defines a further action selection process which determines a single most optimal action $a \in \hat{a}$ based on the in-game situation. Since the offline policy already defines an action set optimal to both the POMDP and MDP abstracts, only a minimal amount of learning needs to be performed online. A simple and efficient learning algorithm is adapted from a previous work by Tan and Cheng (2008a), whereby each action a is assigned a weight ω_a. Action selection then follows a standard ϵ-greedy algorithm (Sutton and Barto 1998) and weight updates are performed via the function $\omega_a = \omega_a + \alpha(\gamma_\tau - \overline{\gamma_\tau})$, where γ_τ is a variable reward signal generated at time τ, $\overline{r_T}$ is a reference point to determine the relative size of γ_τ, and α is a positive step-size parameter to control the magnitude of change.

It can be seen that the majority of adaptation is computed offline and leaves a little online learning capability which can be efficiently performed in-game. This ensures a game AI that can already work very well out of the box, but additionally allows for a small amount of online adaptation to take place as necessary. In the case where the AI function in Equation 7 falls back to the backup union set, the usefulness of this online learning process will become more obvious, where it is given greater responsibility for selecting the optimal action.

In a nutshell, an agent that uses the IMPLANT architecture will fundamentally act optimally with respect to the static game environment, and in addition, as the POMDP belief state becomes more accurate via observation updates, the actions it takes becomes more tailored towards the style of the player as well. It is this ideology that drives this approach.

Results

The architecture is implemented and illustrated in an NPC agent in a tennis video game. In the first set of experiments, the IMPLANT tennis agent acts as an opponent towards the player. Then the implementation is extended to a doubles tennis game settings whereby the IMPLANT agent now act

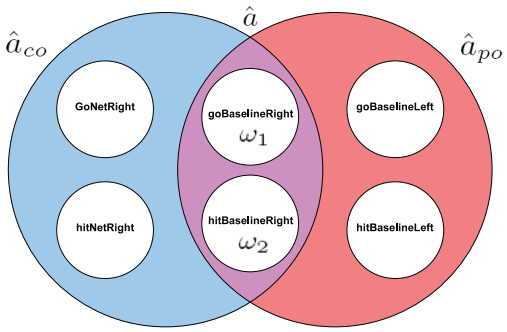

Figure 2: The Action Integrator in a tennis game. The current optimal POMDP abstract action $\hat{a}_{po}$ indicates that the agent should use a strategy biased to towards the baseline whilst the current optimal MDP abstract action $\hat{a}_{co}$ indicates a strategy biased towards the right. The intersection results in the abstract action $\hat{a}$ that contains the actions *hitBaselineRight* and *goBaselineRight*. Thereafter a final action is selected based on the learnt weights ω_1 and ω_2.

Figure 3: The IMPLANT Tennis Agent (bottom right). The agent at the bottom left is the player, and the two agents above are the opponents. In this doubles tennis game setting, the IMPLANT agent makes use of the IMPLANT architecture to adapt to the player and cooperatively defeat the opponent agents

as a cooperative agent as the partner to the player, where the game complexity increases substantially with 2 more opponent agents. A screenshot of the doubles implementation is shown in Figure 3. The results show that in both noncooperative and cooperative settings, the IMPLANT agent is the most feasible in terms of both adaptation performance and speed when compared to various other common AI architectures. Details of the results are given in the subsections that follow.

Basically the states and transitions follows that of a normal tennis sports game. The actions available to each agent basically allows for agent movement and hitting of the ball to target positions. Each hit-type action has a high probability of landing in the targeted region, and some small chance of landing in any of the remaining regions (plus the out position). It is assumed the agents do not fall down or get disoriented, so the movement actions have a deterministic chance in reaching the target regions.

In the states, a *player_model* attribute defines the player's model of play. There are basically 2 player models defined here which are simplified versions of the two common styles of play in tennis, namely the *volleyer* and the *baseliner* (Matsuzaki 2004). The *volleyer* basically likes to stay near the net and take volleys whilst the *baseliner* likes to stay near the baseline and take lobs and ground shots. Note that there is also an *all_court* style which is deliberately not included in the *player_model* attribute so as to test the adaptive capabilities of the architecture in the experiments. The *all_court* style is basically an all-rounded player which can choose to play either the *baseliner* or *volleyer* style, and switches between the two during the game.

Adaptation Performance

The game is ran for 1000 rounds for each player model. The experiments are ran for both the noncooperative singles game and the cooperative doubles game, and scores are plotted in Figure 4. In each experimental set, a different setup of the player model is configured, namely the *volleyer*, *baseliner*, and *all_court* models. As mentioned, the *all_court* model is deliberately excluded from the state attribute *player_model* as defined in the architecture. This *all_court* model is implemented and included in the experiments to purposely test the adaptation capabilities of the architecture.

In a single experimental set, the performance of the IMPLANT agent is compared against a random agent, a scripted agent, a Reinforcement Learning (RL) agent and a POMDP agent. The random agent simply acts randomly. The scripted agent follows a scripted rule which basically hits the ball whenever it is near and switches position in other times. The RL agent uses a Q-learning algorithm (Sutton and Barto 1998) to perform online learning. The POMDP agent contains a pure POMDP architecture whereby all state attributes are modeled in a generic POMDP model. It chooses actions for each belief state based on the POMDP policy it has pre-computed. In the doubles game sets, the opponent agents follow a scripted AI approach.

Generally it can be seen that the IMPLANT agent outperforms all the other 4 types of AI implementations, in both noncooperative and cooperative situations. Firstly, it is a proof of concept whereby it outperforms naive AI implementations (the random and scripted agents). Secondly, it also outperforms other modern adaptive architectures (the RL and POMDP agents) on the whole. Note that it marginally underperforms in the volleyer singles setup when compared to the POMDP agent, which possesses a closely related architecture. However, this slight underperformance is insignificant when the tremendous speed superiority of the IMPLANT agent is shown in the next subsection. It can also be noted that the scripted AI performs rather well in relation to the other adaptive architectures. This can be attributed to the simplification of the tennis game which makes it harder for the adaptive architectures to exploit regularities in play style.

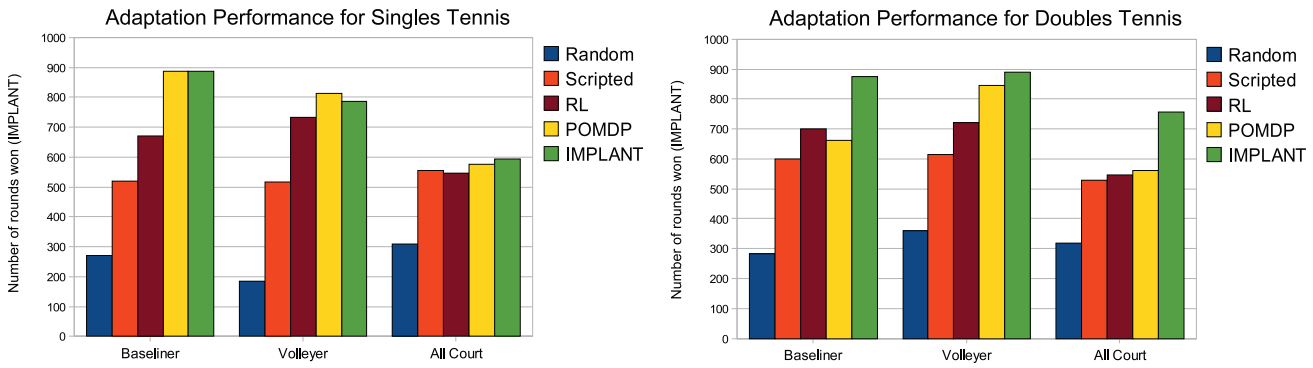

Figure 4: Adaptation performance for the singles (left) and doubles (right) tennis game experimental setups. Each bar set represents a comparison amongst the 5 agent implementations as depicted in the legend, for a particular player model.

Adaptation Speed

Next, a speed test is also performed to evaluate the improvements in pre-computation and querying, and the results for the noncooperative and cooperative setups are as shown in Table 1. Pre-computation time indicates the amount of time it takes for the offline computation of policies whilst query time indicates the time needed for the agent to fetch and perform any online computations to finalize an action for each state during the game. The computation times are based on a machine running an Intel Core 2 Duo 2.33Ghz CPU with 3.25GB of RAM.

To ensure that the pre-computation process for the POMDP runs relatively fast, a modern finite grid PBVI algorithm (Pineau, Gordon, and Thrun 2003) is used for the pure POMDP agent. For the IMPLANT agent, the same PBVI algorithm is used for computing the POMDP policy whilst a classic value iteration algorithm is used for computing the MDP policy.

The RL agent is trained in a series of practice rounds until convergence, which took almost an hour (as shown in Table 1), much longer than the pre-computation time for the IMPLANT agent (a matter of seconds). This training period is necessary for a pure online learning algorithm or else the results would be comparable to the random agent as the Q values need to be updated via experiential learning. Even with a much longer pre-computation time, the RL agent still shows poorer adaptation performance (in Figure 4) than the IMPLANT agent.

It can also be seen in Table 1 that the POMDP agent takes immensely longer to pre-compute policies than the IMPLANT agent. Although this process is offline, it still imposes tremendous overhead when updates need to be made to the model, especially when it is a few hours long. What is more unacceptable is the fact that it takes considerable in-game time (20 seconds query time) to generate an action. This is because at each belief state, it has to select the highest value action based on a dot product of the huge belief state and alpha vectors. The same reason holds for the computation of the next belief state, which adds to the time.

The IMPLANT agent however, is shown to be very fast in both the pre-computation and query times. The low pre-computation time is due to the majority of the problem size being stored in the MDP which has much better tractability than an POMDP in terms of policy computation. As for the query time, it should be noted that it is even comparable to the time needed to generate a random or scripted action, which are almost instantaneous to a normal player. Even though the POMDP agent seems to be almost on par with the IMPLANT agent in terms of adaptation performance in the singles setups, when the speed factor is included here, the IMPLANT agent definitely outperforms convincingly.

Adaptation Speeds for Singles Tennis		
	Pre-computation Time	Query Time
Random	NA	less than 0.001s
Scripted	NA	less than 0.001s
RL	45m32s	less than 0.001s
POMDP	1h28m57s	0.059s
IMPLANT	1.41s	less than 0.001s

Adaptation Speeds for Doubles Tennis		
	Pre-computation Time	Query Time
Random	NA	less than 0.001s
Scripted	NA	less than 0.001s
RL	52m55s	less than 0.001s
POMDP	4h38m48s	9.199s
IMPLANT	1.32s	0.005s

Table 1: Adaptation speeds for the singles (top) and doubles (bottom) tennis game experimental setups. The policy pre-computation and query times are tabulated for the setups. For the random and scripted agents, the pre-computation times are not applicable (NA) and the query times are less than 1 millisecond, which is too fast to be captured in the program.

Discussion and Conclusions

The IMPLANT architecture is proposed in this paper, a planning approach that exploits the fact that the virtual game

world can be decomposed into its MDP and POMDP abstract constituents. Thereafter, optimal policies can be computed separately and combined as a single policy for the agent to act optimally with regards to both the game environment and the player. A small amount of online learning also ensures adaptability when required. A tennis game is implemented as an empirical proof of concept whereby good adaptation performance of the IMPLANT agent is demonstrated against conventional AI implementations as well as a pure POMDP implementation of the same tennis game. The merits in both offline pre-computation speed and online query speed is also shown by comparing against the latter implementations. In addition, the IMPLANT agent is shown to perform well in both noncooperative and cooperative scenarios.

The empirical proof of concept is performed in a rather simplified domain in this paper. As an initial proof of concept, this is assumed to be plausible, but an immediate advancement is to evaluate the framework in an environment in more complicated game domains like that of RTS and RPG scenarios to provide a better generalization of the results. Also as future work, the player modeling component of the architecture can be further formalized to have a generic way of representing player personalities. One way might be to use an action-based formalization like that of Tan and Cheng's (2008b).

The work in this paper also serves as an exploratory point which reveals important research areas within. A further development of the architecture is a generalization beyond the scope of player types. The architecture can be structured as a divide and conquer framework that solves the fully observable and partially observable attributes of the problem state separately, and combines the solution policy for a single agent. It is hoped that the architecture would be most valuable in other problem domains where this characteristic (where double observability can be assumed) is imminent.

Acknowledgements

Special thanks to Associate Professors Dr David Hsu and Dr Lee Wee Sun for their help on the initial phase of the project as well as suggestions on the tennis game implementation.

References

Burago, D.; Rougemont, M. D.; and Slissenko, A. 1996. On the complexity of partially observed markov decision processes. *In Theoretical Computer Science* 157:161–183.

Charles, D.; Kerr, A.; McNeill, M.; McAlister, M.; Black, M.; Kcklich, J.; Moore, A.; and Stringer, K. 2005. Player-centred game design: Player modeling and adaptive digital games. *In Proceedings of the Digital Games Research Conference* 285,298.

Christian J. Darken, G. H. P. 2006. *Findin Cover in Dynamic Environments, AI Game Programming Wisdom 3*. Hingham, Massachusetts: Charles River Media, first edition.

Donkers, J., and Spronck, P. 2006. *Preference-Based Player Modeling, AI Game Programming Wisdom 3*. Hingham, Massachusetts: Charles River Media, first edition.

Doshi, F., and Roy, N. 2008. The permutable POMDP: fast solutions to POMDPs for preference elicitation. In *In AAMAS*, 493–500.

Geramifard, A.; Chubak, P.; and Bulitko, V. 2006. Biased cost pathfinding. *In AIIDE* 112,114.

Hussain, T. S., and Vidaver, G. 2006. Flexible and purposeful npc behaviors using real-time genetic control. *In Proceedings of The IEEE Congress on Evolutionary Computation* 785,792.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *In Artificial Intelligence* 101:99–134.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstractions for MDPs. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 531–539.

Matsuzaki, C. 2004. *Tennis Fundamentals*. United States: Human Kinetics Publishers, first edition.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for pomdps. In *In IJCAI*, 1025 – 1032.

Remco Straatman, A. B., and van der Sterren, W. 2006. *Dynamic Tactical Postion Evaluation, AI Game Programming Wisdom 3*. Hingham, Massachusetts: Charles River Media, first edition.

Shani, G.; Brafman, R. I.; and Shimony, S. E. 2007. Forward search value iteration for POMDPs. In *In IJCAI*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press.

Tan, C. T., and Cheng, H. 2008a. A Combined Tactical and Strategic Hierarchical Learning Framework in Multi-agent Games. *In Proceedings of the ACM SIGGRAPH Sandbox Symposium on Videogames*.

Tan, C. T., and Cheng, H. 2008b. TAP: An Effective Personality Representation for Inter-Agent Adaptation in Games. *In AIIDE*.

Thue, D., and Bulitko, V. 2006. Modeling goal-directed players in digital games. *In AIIDE* 285,298.

van der Sterren, W. 2006. *Being a Better Buddy: Interpreting the Player's Behavior, AI Game Programming Wisdom 3*. Hingham, Massachusetts: Charles River Media, first edition.

White, C., and Brogan, D. 2006. The self organization of context for multi agent games. *In AIIDE*.

Yannakakis, G. N., and Maragoudakis, M. 2005. Player modeling impact on players entertainment in computer games. *In Springer-Verlag: Lecture Notes in Computer Science* 3538:74.