Use Of Online POMDP with Heuristic Search Applied to Decision Making in Real Time Strategy Games

Thiago F. Naves * Carlos R. Lopes†

Federal University of Uberlândia, Faculty of Computing, Brazil

ABSTRACT

Decision making in real-time strategy games (RTS) is a complex task due to the number of actions available and the environment with partial information. Partially observable Markov decision process (POMDP) is an approach that provides good performance and reward values in environments with the limitations discussed. However, its use in RTS games is limited due to real-time constraints and difficulty in abstracting a complete set of actions into a single decision-making. This work proposes the application of online POMDP with heuristic applied to decision making in the gaming domain of StarCraft. An architecture that allows the consideration of macro actions and a modification in the AEMS algorithm with use of game time as additive, are proposed. The results show that POMDP is applicable to RTS games, satisfying time constraints and achieving good game results against StarCraft standard AI.

Keywords: POMDP, decision making, planning, actions, RTS games

1 INTRODUCTION

Making decisions in environments with uncertain information and presence of enemies to the agent is a complex task due to the amount of possible states and actions that can be considered. Real-time strategy games (RTS) are a domain where problems related to environmental uncertainty and decision-making model are placed at a high level of demand, especially due to real-time constraints. Partially observable Markov decision process (POMDP) [2], provides a generic model for decision making in environments with the characteristics of RTS games.

The decision-making process in RTS Games involves the choice of actions to be performed during a match. However, actions have preconditions for execution, and the choice must be made taking into account factors such as strategy, presence of enemies and game time of the match. In addition, the execution of an isolated action does not reflect major changes in the environment, it is necessary to consider a set of actions with a specific objective.

The use of POMDP for decision making in RTS games, may be unfeasible due to the execution time spent by the framework in generating belief states in a horizon with high amount of game elements. The use of POMDP for decision making in RTS games, may be unfeasible due to the execution time spent by the framework in generating belief states in a horizon with high amount of elements of the game. However, the online version of the algorithm using heuristics to limit the amount of belief states, prove to be effective in real time environments with actions and preconditions. Thus, we explore the use of online POMDP with the Anytime Online Search (AEMS2) [7] algorithm for decision making in RTS games. A variant to the algorithm that uses game time as a parameter for decision making it is also proposed.

*e-mail: tfnaves@ufu.br

†e-mail: crlopes@ufu.br

2 BACKGROUND

2.1 RTS Games

In RTS Games the player should manipulate various resources and units, by developing economies, focusing on defeating one or more enemies. Raw resources, like minerals, allow the creation of other resources, such as barracks, that are used to create mobile and airborne units. We use the StarCraft [1] game as testbed environment. Among games of this genre, StarCraft has one of the largest amounts of resources, actions and preconditions.

There are three kinds of races available to play: Terran, Protoss and Zerg. They have different features and units that provide a number of strategies for attack and defense against enemies. During a match, the playing time is measured in the seconds that have already been played. Certain types of actions and strategies are executed at specific time intervals of the game. To produce any resource in the game you need to perform actions. These have as preconditions resources that must be generated before their execution. Actions can generate resources, control units and execute attacks against enemies within the game.

2.2 POMDP

POMDP is a framework whose model can be represented by the tuple $\langle S,A,O,T,Z,R\rangle$, with each of these elements as follows: S is the set of all possible states of the environment; A is the set of all possible actions that the agent can perform in the environment; O is the set of observations that can be received directly from the environment; T is a state transition function $T : SxAxS \rightarrow [0,1]$, where $T(s,a,s') = Pr(s`|s,a)$ representing the uncertainty by the probability of the agent performing an action $a \in A$ in a state $s \in S$ and going to a state $s` \in S$; Z is an observation function $SxAxZ \rightarrow [0,1]$, where $Z(s,a,o) = Pr(o|a,s)$ generates the probability of the agent perceiving $o \in O$ after action a and go to state $s`$; R is the immediate reward function $SxA \rightarrow R$, given that the agent performed the action a in the state s.

In online POMDP, when the agent is in a state s and receives an observation o, it performs the belief state update and computes an optimal policy that maximizes the total reward for execution of each action a [8]. The objective is to achieve a sequence of actions that maximizes the reward, for this a policy $\pi : B \rightarrow A$ that references an action a for a state belief $b \in B$ is used. The agent is uncertain as to the state in which it is due to uncertainty of the environment, so belief states are usually represented by a probability distribution function, following the definition: $b = b(s)|s \in S, 0 \leq b(s) \leq 1, \sum_{s\in S} b(s) = 1$. The total reward is induced by the function $V(b,\pi)$, from the belief state b, and the current policy π, being computed according to the equation 1 below:

$$V(b,\pi) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)|b,\pi] \quad (1)$$

with the discount $\gamma = [0,1]$. Unlike the traditional POMDP implementation, the online version interleaves planning and execu-

[1] StarCraft: Brood War (expansion) was released in 1998 by Blizzard Entertainment.

tion. Whenever a policy is executed and the agent receives a new observation, it is made a belief state update. This process forms a loop between updating the states (planning) and using the chosen action (execution). The belief state update is the most complex process, a tree that performs a lookahead beginning with the belief state b_0 is generated. The tree is expanded by generating new states from observations and actions, the actions are connected to the states to compute the total reward. The action belongs to the branch where the best reward is chosen and executed, restarting the process of updating states. The belief state update is done by the equation 2 below:

$$b'(s') = \tau(b,a,o) = \eta Z(s',a,o) \int_{s \in S} T(s,a,s')ds \qquad (2)$$

where the transition function T and the observation function Z are used, η is a normalization constant.

3 Related Work

The use of Case Based Reasoning (CBR) is greatly explored in decision-making in RTS games. In [5] the technique is exploited primarily in the extraction of replays data and later retrieval of cases during the game. Next, [6] improve the model with an online architecture with interlaced planning and execution. The domain testing of the works is the game Wargus.

In relation to the use of Markov Processes, in RTS games there are a few contributions. The [1] work proposes the use of a Markov Decision Process (MDP) for controlling units in StarCraft, but the domain uncertainty is ignored due to focus only on locally controlled units. This work is based on [11], which uses QLearning to learn from replays about combat between units of the game. Both works focus on small scale combat with only a few units.

The work of [4] cites and explores the use of sampling with Monte Carlo tree search. With the naÃ¯ve sampling algorithm it is possible to manage the amount of branches in an RTS game with superior performance to other sampling methods, even when the game gets more complex.

4 Architecture For Online POMDP in RTS Games

We will exemplify an basic architecture we use to configure online POMDP for decision making in RTS Games. With this architecture it is possible to obtain sensory data of the match as game time and amount of enemies in the form of observations for the POMDP. The actions chosen by the POMDP are sent to the game and executed in the disputed match. The architecture can be expanded to support new modules and algorithms to manage other game challenges.

The basic architecture of online POMDP for RTS games is shown in Figure 1. The decision-making process begins in the *GameWorld*, with the game match in progress. Sensory data, such as amounts of resources, units, positioning, enemy information, among others, are captured by BWAPI[2] in the form of observations. In the Figure are the processes and techniques that work together with the POMDP for decision-making. The goal is to choose the best possible action when observations are received from the game environment and perform this action, receiving new observations and keeping this process repeatedly. In each process there is an arrow coming out or coming from it, which indicates what kind of data this process provides or requests. The order in which the decision is made is indicated by the numbers (I, II, II), respectively, presents in the last part of the name in each arrow.

According to Figure 1, BWAPI is able to capture the data and execute commands within the game, being responsible for the communication between our architecture and StarCraft. The BWAPI sends the observations in the form of a vector ρ, where $\rho[q] = [0,n]$, q is the maximum number of observations that can be received

[2]https://github.com/bwapi/bwapi

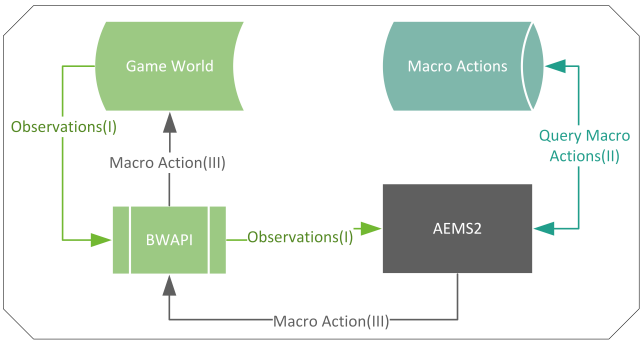

Figure 1: General architecture of online POMDP for RTS games.

equal to 16, and n is the amount of resources or units of each observation, in form:

$$\rho_t = (0,150,5,1,0,0..), \rho_{t+1} = (1,50,6,1,0,2..), ... \qquad (3)$$

where t is the time of the game in which the vector was captured. When an index of the vector is 0, it means that the respective observation of that index was not observed. Any other value means the amount of resources or units of that observation in the current moment. Thus, in a vector of observations ρ at time t, no unit was destroyed (first index with value of 0) and 150 *minerals* were already gathered (second index with value of 150). At a time $t+1$ in the game, it can be observed that a unit was destroyed, its quantity and the *minerals* decreased; however, a new unit was built (third index with value 6) and two enemy units were seen in the map (last index with value of 2).

With the observations being delivered to the POMDP by the BWAPI the decision process is initiated, being represented in Figure 1 by the the AEMS2 execution policy. The POMDP does not use the standard game unitary actions, i.e., isolated actions, like moving a single unit or single collection of minerals. In this step, it uses macro-actions. These are used as sets of actions, representing a specific strategy that contains a series of actions that produces certain resources and is represented by an identifier label. This abstraction is useful so that the POMDP does not have to decide for a single action every second, the set of all macro-actions represents the universe of possible POMDP choices. The macro-actions are identical representations to the so-called build orders, which are strategies of production of resources and combat catalogued by communities of StarCraft players.

When a macro-action is chosen by the POMDP it is sent to the BWAPI that performs the actions within the game environment, receiving new observations and restarting the decision-making process. The algorithm 1 shows the code of online POMDP.

Algorithm 1 Online POMDP()

```
1: Static or Preprocess(offline) : G,L,U
2: G ← b_0
3: while GameEnd() ≠ true do
4:     while DecisionTime() = true do
5:         b ← AEMS2(ρ,G,L,U)
6:     end while
7:     Execute(BestAction(b))
8:     ρ ← GetObservation()
9:     NewRoot(G,b,Sample)
10: end while
```

In line 1 of the algorithm 1 the constant or offline processing is listed. G is the expansion tree used to generate belief states, and

make lookahead in AEMS2, it is an AND-OR tree; L is the lower bound, and $U$is the upper bound used in AEMS2. In line 7, the $Execute()$ function picks the highest macro-action reward of the two policies. The $NewRoot()$ function puts the belief state generated after the execution of the macro-action as the root of the G tree.

5 AEMS GAME TIME

AEMS(game time) is the execution policy integrating our online POMDP architecture. Its execution is carried out continuously, whenever new observations are received. AEMS uses error minimization based heuristics, given the use of approximate offline solutions. The algorithm was chosen because it contains several desirable properties to operate in the RTS gaming domain. AEMS2 is the version where probability of an action to be optimal $P(a|b)$ equal to its maximum upper bound. We propose to change this definition, and for convenience we will mention the algorithm as AEMS.

Before using the algorithm, an offline processing is done, to define the lower and upper bounds that is used in the computation of the AEMS heuristic. The lower bound is obtained by executing a blind policy [10], with the repetitive execution of the equation 4, which in game matches simulations calculates exactly the minimum expected reward. Matches used for calculation are obtained in the form of game replays between professional StarCraft players. For the upper bound we used the Fast Informed Bound (FIB) algorithm [10], which does not consider the total uncertainty of the environment, but with a degree necessary to determine probability of belief states with information based on different actions. Thus, with the complexity of the RTS game domain, the execution of the algorithm is done within a few minutes, the threshold value in each state estimates how much a macro-action tends to reveal which state belief represents the current situation of the environment. The update of the belief state is done with the equation 5, where α operator represents the bounds. The depth limit for updating states was given in time equivalent to 5 seconds.

$$\alpha^a_{t+1}(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')\alpha^a_t(s) \quad (4)$$

$$\alpha^a_{t+1}(s) = R(s,a) + \gamma \sum_{o \in O} \max_{\alpha_t \in \Gamma} \sum_{s' \in S} O(s',a,o)T(s,a,s')\alpha^a_t(s') \quad (5)$$

The result of the equations is used in the error approximation of using lower and upper bounds in AEMS, given by $\varepsilon(b) = U(b) - L(b)$. To calculate the maximum total reward in lookahead with belief states, in each generation it is always necessary to choose the branch node that is part of the highest total reward. AEMS uses a binary definition for the choice, given by $P(a|b)$. We propose a change to this definition, using the game time that is measured during a match, along with replays dataset information, this becomes an additive that is used in the calculation of $P(a|b)$. The policy version with this additive will be called AEMS(game time). Using the upper bound makes macro-actions that are normally performed at a game time being chosen to be execute at game times where other macro-actions would be better. This occurs due to the game actions that are present in different macro-actions. Equation 6 shows the proposed definition:

$$P(a|b) = \begin{cases} 0 & \text{if } U(a',b) < (minorB) \\ Ulr(a',b) & \text{otherwize} \end{cases} \quad (6)$$

where $U(a',b)$ is the upper bound on the value of action a'; $minorB$ is a variable that stores the largest lower bound found to each node that is generated in the tree, if a new highest bound is found, the value of the variable is updated; $Ulr(a',b) = U(a',b) * Lr(a',b)$, where $Lr(a',b)$ is the ratio of the macro-action used to update the belief state with the current game time that is in progress. The value of LR was obtained with a simple linear regression of least squares, where the game time is the explanatory variable and the macro-actions are the dependent ones, the regression was executed offline. Thus, besides the upper bound value, the pertinence additive of the macro-action with the current game time in the choice of the best candidate is used. In an example, macro-actions with high upper bound value but with frequent use at the beginning of the game will have their value offset when considered in a game with a few minutes already played. When $Ulr(a',b)$ is computed, a macro-action can at most increase its bound by 30% of the original value, thus, the regression has no greater contribution in the choice of the action than the upper bound.

AEMS(game time) also computes the probability of reaching a fringe belief state at a specific depth using $P(a|b)$, creating the definition $P(b^d)$. With all the necessary elements, the heuristic to calculate the error for a node is given by $E(b^d)$. Among the desirable properties of the $E(b^d)$ heuristic in relation to the RTS game domain, we can highlight: prefer nodes with loose or wide bounds, these nodes are usually different and visiting them helps make better decisions; favors the exploration of belief states most likely to be found in the future, avoiding rework with calculations; the definition of $P(a|b)$ increases the performance of the algorithm, with our proposed change performance is maintained and are considered more macro-actions with good upper bounds and pertinence with the current game time of a match.

6 EXPERIMENTS AND DISCUSSION OF RESULTS

In order to evaluate the use of the online POMDP with the suggested architecture, tests were made with matches played in StarCraft. The methods used are: AEMS is the use of the traditional algorithm without the game time additive; AEMS(game time) that is used in the online POMDP architecture with game time additive of $P(a|b)$; HSVI [9] is a heuristic algorithm with similar characteristics to AEMS, in the generation of belief states the algorithm returns that approximate value that exceeds the upper bound. SARSOP [3] is the only algorithm that does not use heuristics, it is based on the strategy of approximate values with point-based iterations. The game class used was chosen randomly between *Terran* and *Protoss*.

Table 1 presents the results of the performance test. All results represent the average of the values except the belief states and offline time. The reward is calculated based on the winning probability of each macro action chosen by the algorithm. After the chosen macro action begins to execute, the amount of enemy destroyed, built resources, accumulated minerals and average time without losing any resources is captured. For each of these, a reward of +10 is awarded. For each base resource destroyed, direct confrontation with smaller enemy numbers than the enemy, presence of new enemies in less than 10 seconds, and idle resources on the base are discounted -10 of the reward. The bound error reduction is given by $BR(b) = 1 - \frac{U_T(b) - L_T(b)}{U(b) - L(b)}$, which is the difference between the bounds values obtained offline and those obtained in the online execution. In the tests the opponent was StarCraft standard AI and all the matches lasted less than 7 minutes.

In relation to the time spent for each decision making, AEMS obtained the best result with 0.412 seconds. AEMS (game time) came in second with a small difference due to the time taken to obtain the data and calculate the game time additive. SARSOP had its offline execution measured, match data was simulated and the algorithm had a limit of 1.5 seconds to return a macro action, time interval greater than that obtained by online algorithms. Approximate value strategy requires many iterations and state generation with several portions of state space.

The amount of belief states generated by AEMS was higher

Table 1: Performance test results in 75 runs in the StarCraft environment using random map matches. Average confidence interval of ± 0.92 for all results. N. Actions | A | = 37; N. Observation | O | = 8; N. Opponents = 1

	Time (sec.)	Belief States	Reward	Bound Error Reduction
AEMS	0,412	12647	28,2 ± 2,35	28,4
AEMS (game time)	0,554	10447	37,5 ± 2,91	30,2
HSVI	0,864	9874	22,4 ± 1,89	19,8
SARSOP	1,451	7341	12,7 ± 0,81	14,3

among the algorithms. AEMS(game time) tends to generate states that are more promising based on the game time actual. This criterion restricts the number of states visited and later expanded, but the algorithm achieved a higher average reward than the AEMS. SARSOP could generate more states if there was no time constraint, however the time spent would make the algorithm not suitable for use within the RTS game environment.

AEMS(game time) achieved rewards and bound error reduction slightly above the AEMS, due to the calculation of the best action with use of game time. With this calculation the rewards obtained are greater when the match is in a battle dynamic, these values surpass even more the values of offline bounds increasing the error reduction. Offline time represents the time spent to calculate the initial bounds, the RTSSample used less time because it calculates only the maximum limit and uses the Rollout algorithm for this task. .

In relation to the rewards, AEMS(game time) achieved the best average. With 75 games played with the algorithm were 56 wins, 7 losses and 12 games without result, where the game played time exceeded 10 minutes. The macro actions chosen in the third minute of game were mostly offensive, so the enemy is unprepared and does not have enough defense resources. Already in the AEMS, were chosen offensive and defensive macro actions, characterizing a more balanced strategy. Thus, the enemy manages to prepare the first attack. The AEMS(game time) behavior is due to the tendency of the players to carry out attacks in the beginning of the game, in the second and third minute of game in average.

7 Conclusion

This work presented the use of online POMDP with heuristics applied to the RTS game domain. An architecture that abstracts the need to choose sets of actions and obtain data directly from the game environment is presented. The macro action concept is discussed and enables a long-term implementation of actions by the POMDP. This architecture can be changed with the addition of new modules and algorithms. It was explored the use of the AEMS algorithm, which was modified using game time, a characteristic present in any game of the genre. Based on game time, macro actions can be chosen based on the analyzed behavior of experienced players.

The results show that the POMDP can be used in RTS games achieving good results and fulfilling the time restrictions. The amount of generated belief states allows exploring promising states taking into account different strategies. The rewards against the standard AI of the game can be maximized with use of other algorithms, with increased performance of the POMDP and more states beliefs generated.

Acknowledgments

This research is supported in part by Coordination of Higher Level Personnel Improvement (CAPES) and Faculty of Computing (FACOM) from Federal University of Uberlândia.

References

[1] R. C. L. G. E. Bargiacchi, R. Verschoor and D. Diederik. Decentralized solutions and tactics for rts. In *BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence, Delft, The Netherlands, November 7-8, 2013*, 2013.

[2] L. P. Kaelbling, L. M. Littman, and C. R. A. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[3] H. Kurniawati, D. Hsu, and W. S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland., 2008.

[4] S. Ontañón. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research*, 58:665–702, 2017.

[5] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram. Case-based planning and execution for real-time strategy games. *International Conference on Case-Based Reasoning*, pages 164–178, 2007.

[6] S. Ontanón, K. Mishra, N. Sugandh, and A. Ram. On-line case-based planning. *Computational Intelligence*, 26(1):84–119, 2010.

[7] S. Ross and B. Chaib-Draa. Aems: An anytime online search algorithm for approximate policy refinement in large pomdps. In *IJCAI*, pages 2592–2598., 2007.

[8] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[9] T. Smith and R. Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527. AUAI Press, 2004.

[10] T. Smith and R. Simmons. Point-based pomdp algorithms: Improved analysis and implementation. *21th Conference on Uncertainty in Artificial Intelligence*, pages 542–547, 2005.

[11] S. Wender and I. Watson. Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar. *IEEE Conference on Computational Intelligence and Games (CIG)*, 2012.