A Reinforcement Learning Behavior Tree Framework for Game AI

Yanchang Fu[1, a], Long Qin[1, b] and Quanjun Yin[1, c]

[1]College of Information and System Management, National University of Defense Technology, Changsha, Hunan, P.R. Changsha, China, 410073;

[a]yachee_fu@163.com, [b]qldbx2007@sina.com, [c]yin_quanjun@163.com

Keywords: Behavior Tree; Reinforcement Learning; Game AI; Agent; Raven.

Abstract. This paper discussed the implementation of behavior tree technology in behavioral modeling domain. Existing framework can’t provide the ability of reasoning while take into account the ability of learning. To solve this problem, we propose a reinforcement learning behavior tree framework based on reinforcement theory. Following our study, a QBot model is build based on the framework in the Raven platform, a popular test bed for game AI development. This paper carried out simulation experiments which include 3 opponent agents. The result shows that QBot outperforms the other 2 Raven_Bots which adopt the default agent model in Raven platform, and thus the result proves that the framework is advanced.

Introduction

Game AI should enable the developers to create a compelling experience for the player [1]. In the known games, the finite state machine (FSM) model is the main method of behavior modeling [2]. FSM is concept friendly and the structure of which is clear. However, the workload of modification and increase of state is huge, so it is inconvenient to use in complex environment system. Fixed rules make the agent looks dull, which is not satisfactory. In order to relieve the shortcoming of FSM, Isla puts forward the behavior tree model [3], which greatly improves the development efficiency, but it is still a fixed-rule decision model. Unity introduced the utility theory into the behavior tree model [4], changed the selection node and made game characters with certain intelligence. But the model still need complex priori knowledge to make the construction result reasonable.

Reinforcement learning is a machine learning framework which is used to solve the problem of Markov Decision Process (MDP) [5]. Instead of setting the action order, reinforcement learning allow the Agent explore actions in the environment and accumulate rewards in order to select optical actions. The mainstream of the reinforcement learning method include Q-Learning, SARSA and TD-Learning, etc.

The contribution of this paper is to combine the reinforcement learning theory with the behavior tree model, and apply it to the game AI behavior modeling, in order to build the agent in the complex game environment. The second part of this paper introduces behavior tree model, reinforcement learning theory and Markov decision process. In the third part, we define the reinforcement learning behavior tree model. In the end of the paper, we conclude and point our next research steps.

Backgrounds

Behavior Tree. A behavior tree is a directed tree, which is composed of nodes and edges[6]. Nodes are divided into leaf nodes and non-leaf nodes. Each node has only one parent node, the node is the parent node, and the node is the child node except root node. There are five kinds of nodes, as shown in Table 1 where the leaf node can be action, non-leaf nodes can be selector, sequence, parallel and decorator. The first three are also known as the control node, which can be used as the parent node that can have more than one child. The decorative node has only one child node. The root node has only one child node and the child node must be a control node.

The behavior tree is an execution stream, and each execution cycle is a tick. Tick from the root node to start the call, traverse each node in accordance with depth first order. Different control nodes have different control strategies. When tick to the action nodes behavior tree will control agents to make corresponding action. If the action in a cycle cannot be finished, it will return a RUNNING status. If it be completed, it will return a SUCCESS status. And if it is impossible to continue, it will return a FAILURE status.

Table 1 The node types of behavior tree

Node type	Succeeds	Fails	Node type
Selector	If one child succeeds	If all children fail	Selector
Sequence	If all children succeeds	If one child fails	Sequence
Parallel	If at least N children succeed	If more than M-N children fail	Parallel

Selector. Selectors are used to find the first child that is not fails. A selector will return immediately SUCCESS or RUNNING when one of its children returns SUCCESS or RUNNING, otherwise it will tick all of its failed children and return FAILURE to its parent. Selectors are equivalent to logical-and operation. Algorithm 1 shows the pseudo code of selector.

Algorithm 1. Pseudo code of selector

```
Selector() function return status
   for each child node[i]
      childStatus <-Tick(node[i])
      if childStatus equals RUNNING
         return RUNNING
      elseif childStatus equals SUCCESS
         return SUCCESS
      end
   end
   return FAILURE
end
```

Sequence. Sequences are used to sequentially execute sub nodes. A selector will return immediately FAILURE or RUNNING when one of its children returns FAILURE or RUNNING, otherwise it will tick all of its successful children and return SUCCESS to its parent. Selectors are equivalent to logical-or operation. Algorithm 2 shows the pseudo code of sequence.

Algorithm 2. Pseudo code of sequence

```
Sequence() function return status
   for each child node[i]
      childStatus <-Tick(node[i])
      if childStatus equals RUNNING
         return RUNNING
      elseif childStatus equals FAILURE
         return FAILURE
      end
   end
   return SUCCESS
end
```

Parallel. Parallel in any case will tick the all of the child nodes, and then calculate the return value of all the child nodes. When any child node returns RUNNING, it returns RUNNING to the parent node, when it reaches a given number of child nodes to return to SUCCESS its return SUCCESS, otherwise return FAILURE. Algorithm 3 shows the process.

Decorator. A decorator's role is to change the return value of the child node, the condition, or control the times of sub nodes to implement.

Action. Actions the real implementation module of behavior trees are all leaf nodes. The return values of actions are to be considered in the simulation environment setting as mentioned above.

Algorithm 3. Pseudo code of Parallel

```
Parallel() function return status
    SumOfSuccess <- 0
    for each child node[i]
        childStatus <-Tick(node[i])
        if childStatus equals RUNNING
            return RUNNING
        elseif childStatus equals SUCCESS
            SumOfSuccess <- SumOfSuccess + 1
        end
    end
    if SumOfSuccess > threshold
        return SUCCESS
    else
        return FAILURE
    end
end
```

Reinforcement Learning. The reinforcement learning theory is developed to solve such a problem how an autonomous agent could choose the best action to achieve its goals around the environment [7]. In this section, we define the general form of learning control strategy based on Markov decision process (MDP), and introduce a reinforcement learning method called Q learning.

In Markov decision process, agent can be aware of different states of its environment set which is denoted by S. And there is a set of actions to be executed denoted by A. In each discrete time step t, the agent senses the current state s_t selects current action and executes it. The environment responds to the choice of the agent, returns the reward $r_t = r(s_t, a_t)$, and generates a successor state $s_{t+1} = d(s_t, a_t)$. The function $r(s_t, a_t)$ and $d(s_t, a_t)$ are the part of the environment, which the agent does not known. The MDP model considers that the function r and d depend only on current action and state. In this paper, we only consider the situation that the action and state are finite.

The mission of Agents is to learn a strategy $\pi: S \rightarrow A$, which is based on the currently observed state s_t to select the next action a_t, i.e., $p(s_t) = a_t$. To do this, we define the cumulative reward $V^\pi(s_t)$ under the policy π and initial state s_t as:

$$\begin{aligned} V^\pi(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned} \quad \textbf{(1)}$$

$0 \leq \gamma < 1$ is a discounted factor, which indicates the proportion of the time delay and the immediate return. If $\gamma = 0$, it is said that only consider the immediate return, and when the γ is closed to 1, it indicates that the future return have a greater degree of importance.

Now we can precisely define the learning mission of agents. We require agents to learn a policy π maximizing $V^\pi(s)$ for any state s. That policy is called optimal policy, using π^* to express:

$$\pi^* \equiv \underset{\pi}{arg\,max}\, V^\pi(s) \quad \textbf{(2)}$$

The cumulative reward function of the optimal policy is denoted by $V^*(s)$. Here

$$\pi^* \equiv \underset{a}{arg\,max}[r(s,a) + \gamma V^\pi(d(s,a))] \quad \textbf{(3)}$$

execute which we think we can get the max reward after state s. However, this shows that only when agent has knowledge of $r(s,a), \forall s, a$ and $d(s,a), \forall s, a$ can we learn V^* under the optimal policy. Unfortunately, we do not have it.

But at certain state $\mathrm{s_t}$, an agent could percept the reward $r(s_t, a_t)$ and the next state $d(s_t, a_t)$ when it acts a certain action $\mathrm{a_t}$. Based on this, Q learning introduces a evaluation function $Q(s,a)$, whose value is the maximum cumulative reward from the state s using the action a as the first move:

$$\pi^* \equiv \underset{\pi}{arg\,max}\, V^\pi(s) \quad \textbf{(4)}$$

After such a definition, we have $\pi^* \equiv \arg\max_\pi Q(s,a)$. Luckily, we can get the update rule of $\hat{Q}(s,a)$ in the current perception condition to approach $Q(s,a)$. In paper[7], the algorithm is given, and the convergence of the algorithm is proved. In this paper we only repeat the description of the algorithm.

Algorithm 4. Q learning algorithm

```
QLearning()
    {gama is the discounted factor}
    for each s, a initializing Q(s,a)<-0
    observe current state s
    loop
        choose an action a and execute it
        receive the current reward r
        observe the new state s’
        Q(s,a)<-r+gama*max_a’_[Q(s’,a’)]
        s<-s’
    end loop
end
```

For action selection rule, we give it a certain probability $1-\varepsilon$ to choose the action that maximize $\hat{Q}(s,a)$, and the probability ε to generate a random action.

Reinforcement Learning Behavior Tree

Definition. In the original definition of the behavior tree, selectors will choose one from its child nodes to tick. The algorithm is required in accordance with the order from left to right to find the first success of the nodes can be performed, which invisible gives nodes on the left the higher weights. This is not reasonable. A more natural consideration is to attach the weight to the child nodes. And selectors tick its child nodes in accordance with the weights from high to low. However, manual weighting values need fully evaluating for the system. Even if taking a lot of manpower and resources, it is difficult to carry on. So this paper puts forward the idea of using reinforcement learning to optimize selectors. The selectors after the transformation are called learning-selector.

At the selectors, we will evaluate the status of agents, which can be realized by discretization of various indicators of agents. In this way, the state and the child nodes of the selector will form a state-action pair (s,a). When the agent triggers a condition, such as hitting an opponent, killing an opponent or getting a supply, we will give it a reward. When the learning-selector is being ticked, it will first evaluate the current state of the agent. Then it will sort the state-action pair in accordance with the $Q(s,a)$ within the certain state, tick the children following this order with probability, and update the Q value.

Algorithm 5. LearningSelector algorithm

```
LearningSelector() function return status
    observe current state s
    for each action a and current s sort Q(s,a)
    disturb this order with a probability e
        for each child node[i]
        childStatus <- Tick(node[i])
        if childStatus equals RUNNING
            return RUNNING
        elseif childStatus equals SUCCESS
            receive the current reward r
            observe current state s’
            update Q value
            return SUCCESS
        end
    end
    return FAILURE
end
```

Propagating Reward. Since the value of the bonus is calculated only in the action node, and the child node of the learning selection node may be a composite, we should provide a way to spread the value of the reward. Fortunately, the behavior tree is a recursive structure, which means after the completion of the sub node tick, it will return to the parent node. So in turn we return the value of status at the same time, until it encounters the first learning-selector node, so as to ensure the learning mechanism and behavior tree compatible. There are four kinds of composite nodes in the reinforcement learning behavior tree: sequence, selector, parallel and learning-selector. In the simulation process, some process has determined weights, for example, it is assumed that the priority of attacking base is higher than attacking airport, in which we use the selector. In some cases, the weight of process is difficult to define, for example rifle or grenade which is better when attacking one soldier. This may be related to the distance, the life value, the number of ammunition. In this case the learning-selector.

Experiments

Scenarios. In this paper, the experiment is carried out under the Raven game [8]. Raven is a 2D shooting game developed by Buckland Mat in his book *Programming Game AI* By Example. It includes the elements of health, ammunition, weapons, supplies, obstacles, etc. Fig. 1 shows the running Raven game.

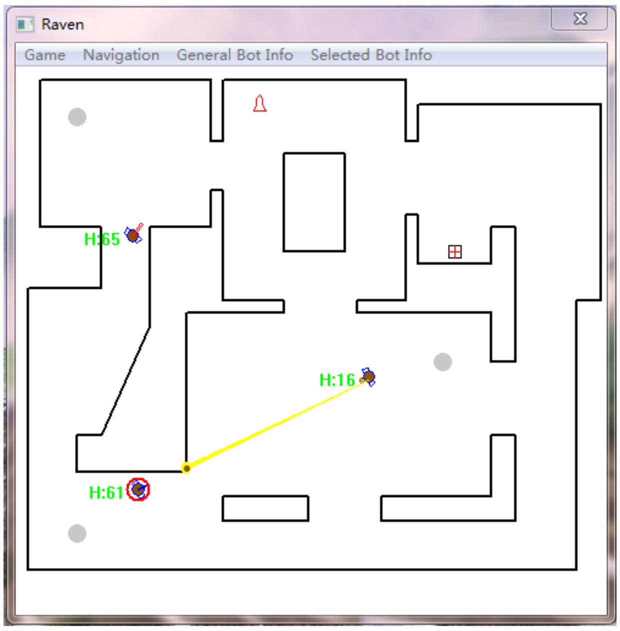

Fig. 1. Raven Platform

At the selectors, we will evaluate the status of agents, which can be realized by discretization of various indicators of agents. In this way, the state and the child nodes of the selector will form a state-action pair (s, a). When the agent triggers a condition, such as hitting an opponent, killing an opponent or getting a supply, we will give it a reward. When the learning-selector is being ticked, it will first evaluate the current state of the agent. Then it will sort the state-action pair in accordance with the $Q(s, a)$ within the certain state, tick the children following this order with probability, and update the Q value.

The Agent designed in Raven is based on the goal driven agent model, which can be seen as the prototype of the utility behavior tree [4]. We called this agent as Raven_Bot. Fig. 2 gives the Raven_Bot decision diagram.

In the experiment, we set three agents to fight each other, two of which are Raven_Bot, and the other one is QBot. The goals of each one is to kill more opponents and get less damage, until there is an agent killing more than 100 opponents and the number of kills minus deaths is at least 30. We carried out a total of 10 experiments, before which the QBot learning for 10 hours.

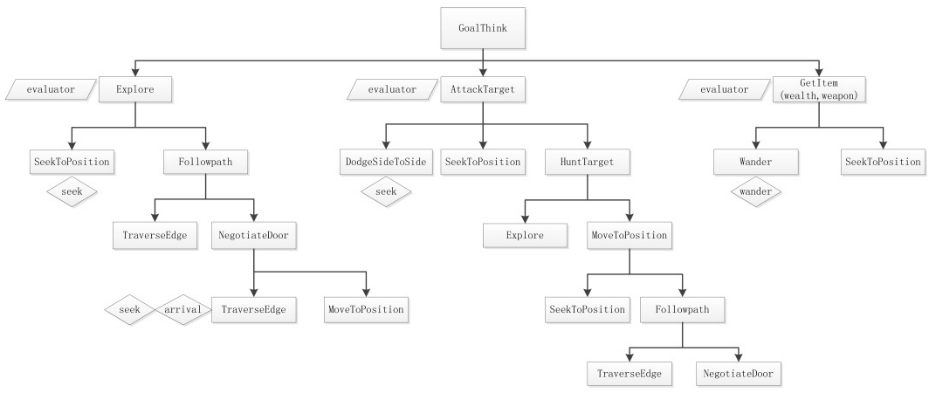

Fig. 2. Raven_Bot decision diagram

Parameter Setting. In the experiment, we set 7 variables that affect the state of the QBot:

- *Health:* indicates the health of QBot, which vary from 0 to 100. The agents (both QBot and Raven_Bot) will die if this value is less than 0. We discretize it into 5 ranks.
- *EnemyInView:* represents the current number of Agent contained in the QBot's field of view, the value of which may be 0, 1 and 2.
- *EnemyHealth:* shows the health of the nearest enemy of QBot, whose value is discretized into 3 ranks.
- *DistanceToEnemy:* represents the distance between the nearest enemy of the QBot and it, whose value is discretized into 5 ranks.
- *DistanceToSupply:* represents the distance to the nearest supply, which can promote the value of health by 10. It is discretized into 3 ranks.
- *SupplyToBeStolen:* if the QBot wants to get supplies and locking one SA to catch, this value represents the distance between SA and the nearest enemy of SA. We discretize this value into 3 ranks.
- *DistanceToWeapon:* represents the distance to the nearest weapon that the QBot does not have which is discretized into 3 ranks.
- *WeaponToBeStolen:* if the QBot wants to get weapons and locking one WB to catch, this value represents the distance between WB and the nearest enemy of WB. We discretize this value into 3 ranks.

The total states of our experiment are. Whenever the QBot consume the enemy health by 1, it will get 1 reward. Whenever the QBot kills an enemy it will get extra 100 rewards. When the QBot catches a supply and refresh 50 healths, it will get 60 rewards, and when it catches a weapon that it does not have before, it will get 10 rewards. But if the object is stolen, it will get punishment with 10. When the QBot is hit by 1 health, it will get punishment with 0.5.

Result. The result of experiments is show in Table 2.

Table 1 The node types of behavior tree

Experiment	QBot Kill	QBot Die	RBotA Kill	RBotA Die	RBotB Kill
1	304	274	309	326	321
2	176	146	157	169	151
3	323	293	363	379	349
4	244	214	264	275	228
5	128	98	143	163	128
6	259	229	225	246	301
7	272	242	235	248	283
8	366	336	411	418	414
9	334	304	317	333	320
10	334	304	346	360	359
Mean	304	274	309	326	321

The result of the experiment are very exaggerated, QBot won all the games. This is mainly due to the no destination walk of Raven_Bot, which will enable them to encounter enemy. While they kills lot but the probability of death will be increased, and the rate is 1:1. QBot will avoid being killed, however, their kills sometimes is lower than Raven_Bot. This shows that QBot is more intelligent than Raven_Bot. And it is also proved that reinforcement learning behavior tree is a feasible method of game agent construction.

Conclusions and Future Work

We can see that the combination of behavior tree and reinforcement learning can be used as the framework of adaptive behavior modeling in game environment. This is a good action selection framework that we demonstrate in the experiment. The construct agent based on the framework shows a better performance against the others.

For future work, we will focus on the following research contents. Convergence is important for reinforcement learning, and we need to verify the convergence of the reinforcement learning behavior tree. In this paper, we have a very simple definition of the states, and the cascade of these simple states will lead to an exponential explosion. So how to effectively construct the states needs to be further studied. Behavior tree is a kind of recursive structure. Combined with reinforcement learning, it can be used as a hierarchical reinforcement learning framework, through which we can build more complex agent in the game scene.

Acknowledgements

The work described in this paper is sponsored by the National Natural Science Foundation of China under Grant No. 61573369 and No.61473300. We declare that there is no conflict of interests regarding the publication of this article. We appreciate fruitful discussion with the Sim812 group: Shiguang Yue, Lin Sun, and Qi Zhang. Finally, we appreciate feedback from our reviewers.

References

[1] Dill, Kevin, and L. Martin. “A game AI approach to autonomous control of virtual characters.” Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC). 2011

[2] Dawe, Michael, et al. “Behavior Selection Algorithms.” *Game AI Pro: Collected Wisdom of Game AI Professionals* (2013): 47.

[3] Isla, Damian. "Handling complexity in the Halo 2 AI."*Game Developers Conference*. Vol. 12. 2005.

[4] Merrill, Bill. "Building Utility Decisions into Your Existing Behavior Tree."*Game AI Pro: Collected Wisdom of Game AI Professionals*(2013): 127.

[5] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[6] Petter Ogren. "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees." *AIAA Guidance, Navigation, and Control Conference* 2012.

[7] Mitchell, Thomas M. "Machine learning." *Boston et al* (1997).

[8] Buckland, Mat. *Programming game AI by example*. Jones & Bartlett Learning, 2005.