

# Mixed Deep Reinforcement Learning-behavior Tree for Intelligent Agents Design

Lei Li<sup>a</sup>, Lei Wang, Yuanzhi Li<sup>b</sup> and Jie Sheng

*Department of Automation, University of Science and Technology of China, Hefei 230027, Anhui, China*

**Keywords:** Reinforcement Learning, Behavior Tree, Intelligent Agents, Option Framework, Unity 3D.

**Abstract:** Intelligent agent design has increasingly enjoyed the great advancements in real-world applications but most agents are also required to possess the capacities of learning and adapt to complicated environments. In this work, we investigate a general and extendable model of mixed behavior tree (MDRL-BT) upon the option framework where the hierarchical architecture simultaneously involves different deep reinforcement learning nodes and normal BT nodes. The emphasis of this improved model lies in the combination of neural network learning and restrictive behavior framework without conflicts. Moreover, the collaborative nature of two aspects can bring the benefits of expected intelligence, scalable behaviors and flexible strategies for agents. Afterwards, we enable the execution of the model and search for the general construction pattern by focusing on popular deep RL algorithms, PPO and SAC. Experimental performances in both Unity 2D and 3D environments demonstrate the feasibility and practicality of MDRL-BT by comparison with the-state-of-art models. Furthermore, we embed the curiosity mechanism into the MDRL-BT to facilitate the extensions.

## 1 INTRODUCTION


Designing an intelligent agent confronted with complex tasks in diverse environments is generally known as an intractable challenge. A universally accepted definition of intelligent agents in (Wooldridge and Jennings, 1995) indicates that the agent can operate automatically, perceive environments reactively and exhibit goal-settled acts initiatively, which demands for the ability of observing, learning and behaving. The techniques of employing such intelligent agents have profound impacts on a wide range of applications including computer games, scenario simulations, robot locomotion.


Behavior Tree (BT), expressed by (Dromey, 2003) in the mid2000s, is a well-defined and graphical framework for modelling AI decision behaviors. As a replacement of Finite State Machines (FSM) and a favorable AI approach utilized inherently in games, BT owns the features of re-usability, readability and modularity. However, an excellent design of BT needs enough experience and efforts when the behaviour representations of agents become increasingly complicated. It's apparent in the fact that agents with

these constrained behaviors have difficulty responding towards dynamically changing environments.

Reinforcement Learning (RL) as one of the paradigms and methodologies of machine learning based on Markov Decision Process (MDP) has been scaled up to a variety of challenging domains, such as AlphaGo (Silver et al., 2016) and AlphaGo Zero (Silver et al., 2017), Atari game (Mnih et al., 2013), Simulated Robotic Locomotion (Lillicrap et al., 2015), StarCraft (Vinyals et al., 2017), even Vehicle Energy Management (Liessner et al., 2019). Accordingly, deep RL gradually emerges with the significant advance of neural network. Compared with behavior trees, agents augmented with RL not only can potentially take adaptive strategies, but also learns incrementally a complex policy. But deep RL models are always accompanied with poor sampling efficiency and limited convergence, lacking a guarantee of an optimal result. Another cause for the bounded applicability is the difficulty of designing a reward function that encourages the desired behaviors all through training. With respect to hyperparameters, most methods depend on special settings and easily get brittle with a small change.

Whether an appropriate model can implement an intelligent agent with given demands is contingent upon effective design mechanisms and applicable ex-

<sup>a</sup>  <https://orcid.org/0000-0003-3496-9752>

<sup>b</sup>  <https://orcid.org/0000-0002-3068-8569>

ecution. As discussed in prior contents, there exist ubiquitous shortcomings in practicability above. It's significant for us to embed the deep RL into BT and offer a valid model pattern. According to (de Pontes Pereira and Engel, 2015), a series of sub-tasks can be abstractly transformed into a reinforcement learning node and in turn BT, profiting from its hierarchical architecture, can be enhanced reasonably by absorbing these nodes. In both theoretical and experimental aspects at last, the model named MDRL-BT can availably incorporate heterogeneous deep RL nodes and normal BT nodes to produce a considerable improvement in intelligent agents design.

## 2 RELATED WORK

The fundamental theories of BT arouse out of (Mateas and Stern, 2002; Isla, 2005; Florez-Puga et al., 2009). (Mateas and Stern, 2002) provided a behavior language designed specifically for authoring believable agents with rich personality as a primitive forerunner. (Isla, 2005) centering on scalable decision-making used BT to handle complexity in the Halo2 AI. (Subagyo et al., 2016) enriches behavior tree with emotion to simulate multi-behavior NPCs in re evacuation.

The deep RL originates from the paper (Mnih et al., 2013) with the enforcement of CNN network directly. (Mnih et al., 2015) has stricken a great success by developing a deep Q-network (DQN) in this field. (Schulman et al., 2015) proposes Trust Region Policy Optimization (TRPO) in policy optimization. On the basis of TRPO, Proximal Policy Optimization (PPO) (Schulman et al., 2017) takes the minibatch update and optimizes a surrogate objective function with stochastic gradient ascent. Soft actor-critic (SAC) are proposed by Haarnoja (Haarnoja et al., 2018) to maximize expected reward and entropy in Actor-Critic (AC).

The concept of integrating RL into BT has been put forward to alleviate the endeavors of manual programming in some research. (Zhang et al., 2017) combines BT with MAXQ to induce constrained and adaptive behavior generation. (Dey and Child, 2013) presents Q-learning behaviour trees (QL-BT). In the (de Pontes Pereira and Engel, 2015), a formal denition of learning nodes is applicable to address the problem of learning capabilities in constrained agents. Yanchang Fu in (Fu et al., 2016) carries out simulation experiments including 3 opponent agents with RL-BT. In (Kartasev, 2019) there are detailed descriptions of Hierarchical reinforcement learning and Semi Markov Decision Processes in BT.

Compared with the relevant works, the contribu-

tions of this paper is intended to contain the following aspects: firstly, we demonstrate a general model MDRL-BT combined flexibly with different deep options and a simple training procedure to design an intelligent agent. Besides, we investigate potential traits of MDRL-BT for an effective training model. The latent variable generative models and primitive process of learning can be strengthened with mixed deep RL algorithms including PPO and SAC by comparative experiments. Furthermore, we set up experiments on Unity 3D environment for high quality physics simulations and revise a simple and unified reward function about scores and time. Finally, the MDRL-BT model with curiosity is implemented practically in an empirical 2D application.

The remainder of this paper is structured in the following: The introduction of intelligent agents and corresponding research are presented firstly. Afterwards, the theories of BT and RL and analysis of MDRL-BT architecture are introduced in detail. In the model section, we outline the framework based on options and bring the RL nodes into BT. At the same time, we facilitate the execution of the model. In the experiments, we build up some experiments to search for better performance and draw some conclusions from the results. Finally, we summarize the work and look forward to the future research direction.

## 3 PRELIMINARIES

Formally speaking, a behaviour tree is composed of some nodes and directed edges where internal nodes called composite nodes and leaf nodes known as action nodes are connected by edges.

Each node is classified by the execution strategy in the following. *Sequence*, analogies to logical-and operation, returns *Failure* once one of the children fails, otherwise *Success*. Note that *Fallback* nodes, equivalent to logical-or, are appropriate for executing the first success nodes. *Condition* nodes represent a proposition check and instantly return *Success* if the condition holds or *Failure* if not yet. All *Action* nodes having specific codes return *Success* if the action correctly completes, *Failure* if it is impossible to continue and *Running* when the process is ongoing.

The execution of BT operates with a tick generated by a root node at a given frequency, which propagates in depth first. When receiving the signal, the node invokes its execution, enables the corresponding behaviors or traverses the tick to children. Each node except root completes with the return status of *Running*, *Success* or *Failure*, which is transferred to the

parent for determining the next routine. Until the root node terminates, a new tick always comes into being from root with a cyclical loop.

The key problem of RL aims at maximizing cumulative rewards in the interactions with environments. At time step  $t$ , the agent in a given state  $s_t \in S$  decides on an action  $a_t \in A$  with respect to a mapping relationship called the policy  $\pi : S \rightarrow A$ , then receives a reward signal  $r_t$  and reaches a new state  $s_{t+1} \in S$ . In a given environment,  $S$  is a complete description of state space and  $A$  often represents the set of all valid actions. Generally speaking, the entire sequence of states, actions and reward can be considered as an infinite-horizon discounted Markov Decision Process (MDP), defined by the tuple  $(S, A, p, r)$ . The state transition probability  $p$  demonstrates the probability density of the next state  $s_{t+1}$  in the condition of the current state  $s_t \in S$  and action  $a_t \in A$ . To represent the long-term cumulative reward, the discount factor  $\gamma$  is considered to avoid the infinite total reward:  $R_t = \sum_{i=t}^T \gamma^{i-t} r_i(s_i, a_i)$ .

In Q-learning, to evaluate the expected return of a policy, a value function is defined:  $V^\pi(s) = E_\pi[R_t | s_t = s]$  and the state-action value function is the expected return for an action  $a$  performed at state  $s$ :  $Q^\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a]$ . From the Bellman equation, the recursive relationship can be shown:  $Q^\pi(s_t, a_t) = E_\pi[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$ . In DQN (Mnih et al., 2015), a deep convolutional neural network is used to approximate the optimal action-value function as follows:

$$Q^*(s, a) = \max_{\pi} E[r_t + \gamma r_{t+1} + \dots | s_t = s, a_t = a, \pi] \quad (1)$$

In the policy network (Silver et al., 2014), log loss and discount reward are used to update the policy guide gradient. The policy can be updated as the equation:

$$\nabla_{\theta} J(\mu_{\theta}) = E_{s \sim \rho^{\mu}} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) |_{a=\mu_{\theta}(s)}] \quad (2)$$

Mathematically, the advantage function which is crucially important for policy gradient methods is defined by  $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$ . Proximal policy optimization (PPO) (Schulman et al., 2017) breaks down the return function into the return function by the old strategy plus other terms with the monotonic improvement guarantee and gives a definition of the probability ratio:  $r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}$  and  $r_t(\theta_{old}) = 1$ . The main objective of PPO is the following:

$$L(\theta) = \hat{E}_t [\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t)] \quad (3)$$

Soft Actor Critic(SAC) (Haarnoja et al., 2018), an extended stochastic off-policy optimization approach based on actor-critic formulation, centers on entropy

regularization to maximize a trade-off between exploration and exploitation with the acceleration of the learning process. The agent at every time step obtains an augmented reward proportional to the expected entropy of the policy over  $\rho_{\pi}$  with trade-off coefficient  $\alpha$ :

$$J(\pi) = \sum_{t=0}^T E_{(s_t, a_t) \sim \rho_{\pi}} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))] \quad (4)$$

## 4 MODEL

The general approach for maintaining the superiority of RL and BT together is to apply the option framework to BT. On this basis, deep RL algorithms can be imbedded unaffectedly in the learning nodes to obtain observation information and make decisions in accordance with the learned policy. In the meantime, the learning nodes are claimed to keep the feasible and constrained characteristics of normal nodes. Deriving from recursive BT, the generated model MDRL-BT stresses on a relatively simple and efficient realization and implements an optimized execution with these nodes.

### 4.1 The Option Framework in BT

The central trait of MDRL-BT focuses on an idea that a big task can be decomposed into multiple smaller tasks in BT and several nodes associated with a task can aggregate into a learning node. Each divided task reduces the non-linear increase of dimensionality with the size of the observation and action space, similar to the essence of Hierarchical Reinforcement Learning (HRL) and Semi Markov Decision Processes (SMDP) based on the option framework proposed in (Sutton et al., 1998). In the framework, a primary option is initialized in a certain state and then a sub-option is adopted by the learning strategy. After that, the sub-option proceeds until it terminates and another option continues like the running process of BT.

An option is a 3-tuple consisting of three elements  $\langle I, \pi, \beta \rangle$  where:  $I \subseteq S$  indicates the initial state of option,  $\pi : S \times O \rightarrow [0, 1]$  ( $O = \bigcup_{s \in S} O_s$ ) represents the semi-markov policy which is a probability distribution function based on state space and option space,  $\mu : S \times O \times A \rightarrow [0, 1]$  with additional action space defines the intra-option policy. For each state  $s$ , the available options are represented by  $O(s)$ . When the present state  $s$  is an element of  $I$ , a corresponding option is successfully initialized. The bellman equation

for the value of an option  $o$  in state  $s$  can be expressed:

$$Q_O^\pi(s, o) = Q_O^\mu(s, o) + \sum_{s'} P(s'|s, o) \sum_{o' \in O_s} \pi(s', o') Q_O^\pi(s', o') \quad (5)$$

The current option chooses the next option  $o$  with the probability  $\pi(s, o)$  during the execution and then the state can change to  $s'$ . Moreover, the definition of the intra-option value function is:

$$Q_O^\mu(s, o) = \sum_a \mu(a|s, o) Q_U(s, o, a) \quad (6)$$

where  $Q_U : S \times O \times A \rightarrow \mathbb{R}$  represents the action value in a state-option pair according to (Bacon et al., 2017):

$$Q_U(s, o, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) U(o, s') \quad (7)$$

$\beta : S \rightarrow [0, 1]$  is the termination condition and  $\beta(s)$  means that state  $s$  has the probability  $\beta(s)$  of terminating and exiting the current option. The value of  $o$  upon the arrival of state  $s'$  with the probability  $\beta(s')$  of option termination,  $U(o, s')$  is written as:

$$U(o, s') = (1 - \beta(s')) Q_O^\mu(s', o) + \beta(s') V_O(s') \quad (8)$$

In the context of BT, the termination condition  $\beta$  is bound up with the return status of *Failure* or *Success*. A new episode starts when an option is activated by a signal tick for a timestep and ends up with the termination of option. With regard to *Running*, it accounts for the process of an option node with a consecutive series of uninterrupted ticks in BT. This would imply that the ticks complete the synchronization with RL algorithm. As far as an MDP problem is concerned, the option collects the actions, rewards and states stored in trajectories  $\mathcal{D}$ , which updates the option-option policy  $\pi$  or intra-option policy  $\mu$ . In general, traditional composite and decorator nodes in BT have a fixed policy for calling their children sequentially. In this paper, we remove the limitations for the utilization of option framework so that the policy  $\pi$  could rearrange the execution order of children.

## 4.2 Reinforcement Learning Nodes

Based on the previous theorem (de Pontes Pereira and Engel, 2015), learning action and composite nodes are referred to as the extensions of the normal BT nodes. These learning nodes not only successfully are equipped with the learning capacities, but also maintain the readability and modularity in hierarchical BT framework. For learning composite nodes, the notion of learning fallback nodes is defined as follows.

**Definition 1.** A learning fallback node below attached with children  $c_1, c_2, \dots, c_n$  as possible choosing

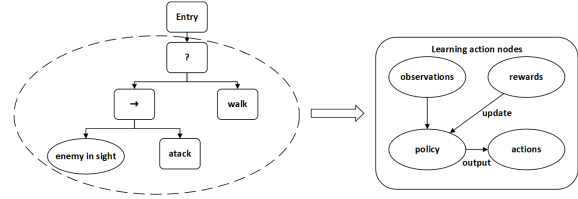


Figure 1: The transformation from several normal nodes to a learning node with observations, reward functions, policy and actions.

sub-options can be modelled as an option with an input set  $I = S$ , an output order  $(c_{i_1}, c_{i_2}, \dots, c_{i_n})$ , a termination  $\beta = 1$  if any  $\text{Tick}(c_i) \in \text{Success}$  or all  $\text{Tick}(c_i) \in \text{Failure}$  and a policy  $\pi$ .

The learning *Fallback* nodes can query the relevant children every episode in learned priority instead of the constant order. The corresponding learned policy  $\pi$ , according to the observations mainly correlated with the state of the environment, devotes to deciding one of the children and then holding a series of updates during execution. The learning *Sequence* resembling learning *Fallback* just differs in the termination condition  $\beta = 1$  if any tick  $(c_i) \in \text{Failure}$  or all ticks  $\in \text{Success}$ . The example of learning composite nodes involving SAC is illustrated in algorithm 1.

**Definition 2.** A learning action node can be modelled as an option with an input set  $I = S$ , actions  $a \in A_s$ , a termination condition  $\beta$ , and a policy  $\mu$ .

In most cases, MDRL-BT abstracts a subtree into a learning action node for potential performance and

Algorithm 1: SAC composite nodes with N children.

**Input:** an input set  $I = S$ , initial state value function parameters  $\phi, \bar{\phi}$  and soft Q-function parameters  $\psi, \bar{\psi}$ , tractable policy  $\theta$  parameters, count steps  $k = 0$ .

**Output:** *Failure, Success, or Running*

- 1: state is *Failure* if *Sequence*, *Success* if *Fallback*
- 2: **if** a tick arrives **then**
- 3: collect global state  $s_k$ , run policy  $\pi_\theta$ , take action  $a_k \in A$  and get an index order  $i_1, i_2, \dots, i_N$
- 4: **for**  $j \leftarrow 1$  to  $N$  **do**
- 5:  $childstatus \leftarrow \text{Tick}(child(i_j))$
- 6: **if**  $childstatus = \text{Running}$  **then**
- 7: return *Running*
- 8: **else if**  $childstatus = state$  **then**
- 9: goto  $\rightarrow$  line 11
- 10:  $state \leftarrow \sim state$
- 11: receive reward  $r_k$ , add tuple  $(s_k, a_k, r_k, s_{k+1})$  to trajectories  $\mathcal{D}_k$ , update the parameters  $(i \in \{1, 2\})$ :  
 $\phi \leftarrow \phi - \lambda_V \hat{V}_\phi J_V(\phi), \psi_i \leftarrow \psi_i - \lambda_Q \hat{V}_{\psi_i} J_Q(\psi_i)$   
 $\theta \leftarrow \theta - \lambda_\pi \hat{V}_\theta J_\pi(\theta), \bar{\phi} \leftarrow \tau \phi + (1 - \tau) \bar{\phi}, k \leftarrow k + 1$
- 12: return state

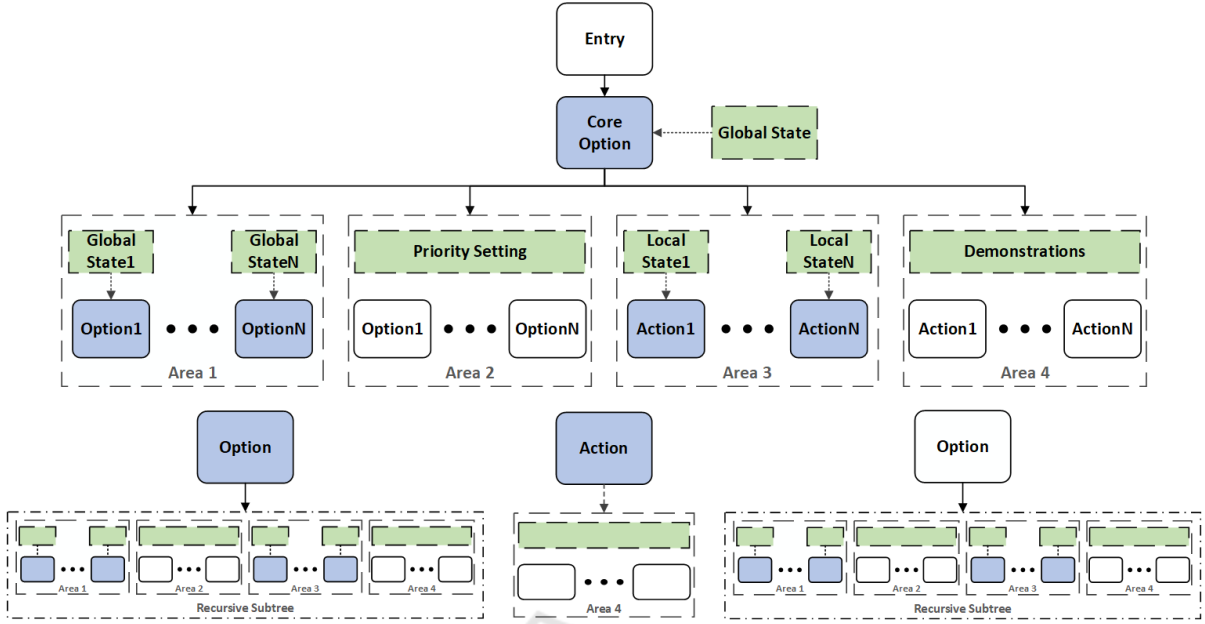


Figure 2: The structure of MDRL-BT contains four categories. The blue option nodes refer to the two kinds of learning nodes with deep RL algorithms. The blank nodes indicate the normal BT nodes and the green rectangle with imaginary line is not a tree node but the input of external state of environment or initial design settings.

simplification. In figure 1, the entire BT can be turned into a learning action node for an enemy attack in this simple case. In short, the learning action nodes carry through an MDP to define the evolution of agent states. The learning action nodes with PPO can be summarized in algorithm 2.

Algorithm 2: PPO learning action nodes.

**Input:** an input set  $I = S$ , initial policy parameters  $\theta$ , value function parameters  $\phi$ , count steps  $k = 0$ .

**Output:** *Failure*, *Success*, or *Running*

- 1: **if** a tick arrives **then**
- 2:     collect local state  $s_k$ , run policy  $\pi_{\theta_{old}}$ , take action  $a_k \in A$ , and receive reward  $r_k$
- 3:     **if** the task goal is finished **then**
- 4:         return *Success*
- 5:     **else if** impossible to finish **then**
- 6:         return *Failure*
- 7:     **else**
- 8:         add  $(s_k, a_k, r_k, s_{k+1})$  to the trajectories  $\mathcal{D}_k$ , compute rewards-to-goes  $\hat{R}_t$ , and compute the advantage estimates  $\hat{A}_t$ , based on value function  $V_{\phi_k}$ .
- 9:         Update the policy parameter:  

$$\theta = \underset{\theta}{\operatorname{argmax}} \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \operatorname{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t)]$$

$$\phi = \underset{\phi}{\operatorname{argmin}} \hat{E}_t[(V_{\phi}(s_t) - \hat{R}_t)^2]$$
- 10:          $\theta_{old} \leftarrow \theta, \phi_{old} \leftarrow \phi, k \leftarrow k + 1$
- 11:         return *Running*

### 4.3 MDRL-BT Architecture

In this section, we will systematically illustrate the extended architecture of MDRL-BT and analyze respectively the different functions of every node area. As stated in figure 2, it is a recursive BT as a whole with a core option and four types of divided areas, in keeping with the hierarchies of option framework. It deserves to be mentioned that every periodic tick represents a temporal level of timescale signal propagating from top to down and activates the running courses of triggered nodes. The core option as a representative of the core logic abstraction from complicated tasks can also be replaced with normal composite nodes with a fixed execution setting. The children of core option is roughly classified on the grounds of types of nodes. Although the four areas are distinct, every area is directly connected with the core option and can be interspersed disorderly with every independent individual of other areas. The parameter  $N$  in every area is also different, ranging from zero to infinity.

The blue option nodes are the learning composite nodes with discrete outputs. In conjunction with global state input relative to the local state, this type of nodes can distill global observations to dispense the order index of children. As seen in figure 2, the recursive sub-tree can follow the option node with the same framework of four areas generation after generation. So the recursive sub-tree can be large or small depending on specific tasks and in this perspective the

MDRL-BT combined with this type of node can be in possess of flexible structure and have certain generality in applications.

The blue action nodes are the learning action nodes which collects the local state to improve policy with continuous or discrete actions. The nodes with meticulous reward function can facilitate implementation of subtask and simplify large-scale architecture. In the presence of several learning action nodes with the similar action space, reward function and task goals, it is recommended that a RL brain can be independent of these nodes and keep parallel connection for reusability and time saving, such as details in Experiment 1. Accompanying the MDRL-BT with this blue action nodes in essence extends the BT to MDRL-BT.

Composite nodes and action nodes are subsumed together into blank nodes. For the blank option nodes they can be conventional identified types of *Fallback*, *Sequence*, *Parallel*, *Decorator* mentioned above. The priority setting,  $p(s) \rightarrow \mathcal{R}$  mapping the state to priority value, is the function of the execution order designed initially as the input. The blank nodes of *Action* or *Condition* with typical commands are common indivisible units. As the granularity of MDRL-BT, the executable action nodes can be defined prefer-

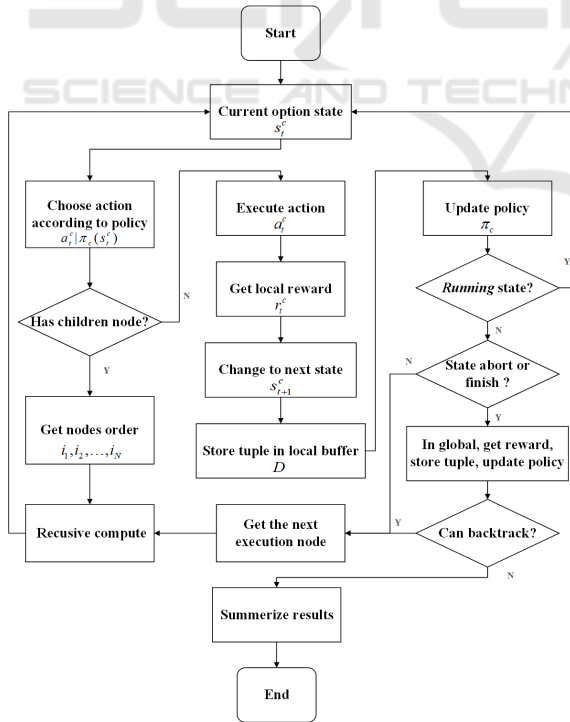


Figure 3: The execution process of MDRL-BT. It is actually a nested process with inter-option and intra-option policy distinguished by having children. The traditional nodes suffer from stationary policies and are ignored selectively.

ably with the demonstrations for solving complicated problems and improving learning efficiency of the tree.

A myriad of flexible classical algorithms are incorporated concurrently when different blue nodes are adopted, giving rise to the nature of BT. It's plausible that MDRL-BT can reap the advantage of BT and RL and can vary with practical applications to cater for designers' needs. The next part would introduce the execution process and effective reward function.

#### 4.4 MDRL-BT Execution

There are  $N$  learning children of core option with unfixed execution time interval  $T_n$ , policy  $\pi_n$ , the corresponding state  $S_n$ , action space  $A_n$ , reward  $R_n$  ( $n \in [1, N]$ ) and  $M$  normal nodes with time interval  $T_m$  and degree of goal completion  $G_m$ . Uniformly the core option has  $T_c, \pi_c, S_c, A_c, R_c$ . Along with the beginning of MDRL-BT, the core option gathers observation  $s^c \in S_c$  and take an action  $a^c \in A_c$ , get the tuple results of index order  $(i_1, i_2, \dots, i_N)$ . At the time  $t$ , we can get the following equation.

$$(i_1, i_2, \dots, i_N)_t = a_t^c | \pi_c(s_t^c), \quad a_t^c \in A_c, s_t^c \in S_c, \quad (9)$$

The execution flow of MDRL-BT can be summarized detailedly in figure 3. Up to now, the tuple  $\langle s_t^c, a_t^c, r_t^c, s_{t+1}^c \rangle$ , where  $s_{t+1}^c$  is the global state of next episode, can be stored in its buffer trajectories  $\tau$  for experience replay. The subsequent procedure may be easily adapted to other learning options with a plurality of subspaces because of the recursive inference for subtree. In the local time interval, the children motivate MDPs, considered explicitly as the sub episode of the high level.

It's an assumption that the first  $k$  children nodes has finished with *Success* status and the  $i_{k+1}$  node aborts the next execution. It is derived that the total time of core option is the sum of first  $k$  normal execution time where  $\epsilon$  is the total error of transforming time and  $T_c$  corresponds to the amount of the whole tree execution time.

$$T_c = \sum_{j=1}^k T_{i_j} + \epsilon \quad (10)$$

The reactive rewards function needs to reflect the tendency of goal-achieving in a sense. In the ordinary way, the rewards are tied to the execution time and degree of task completion, and should be normalized theoretically for training performance. In this paper, the abort status doesn't exist in scenarios where the core option can execute the all children nodes  $k = M + N$ . Hereby, the mixture rewards of core op-

tion can be computed in the following.

$$r_t^i = \begin{cases} f_i^1(T_i(t)) + f_i^2(G_i(t)) & i \in \text{normal nodes} \\ f_i^1(T_i(t)) + R_i(t) & i \in \text{learning nodes} \end{cases} \quad (11)$$

$$r_t^c = \text{Normalize}(\sum_{i=1}^{M+N} r_t^i), \quad r_t^i \in R_i \quad (12)$$

where  $f_i^1$  is always a piecewise function for reducing the time error  $\epsilon$  and  $f_i^2$  is a mapping function for goal achievement. The equation is not the only rewards design but can sometimes be a more reasonable choice than the others.

Thus far, MDRL-BT has made use of option framework to ascertain usability in a theoretical manner. This conjugated model with this hierarchical design for the intelligent agents, compatible with the structure of BT, mixes nodes together and can overcome the weakness of the RL and BT. In the model, it turns out to be that the learning can be undertaken in tandem by mixed RL nodes and the constrained running is solely in the charge of BT. MDRL-BT with the underlying option framework has circumvented the conflicts between BT and RL and can be easily altered for different targets. In the next section, we will do some experiments for comparison and manifest operability.

## 5 EXPERIMENTS

Several valuable experiments with various complexity are performed in this section to identify the characteristics of MDRL-BT and validate the intelligent agents. Most experiments are consequently conducted and measured on Unity 3D environments close

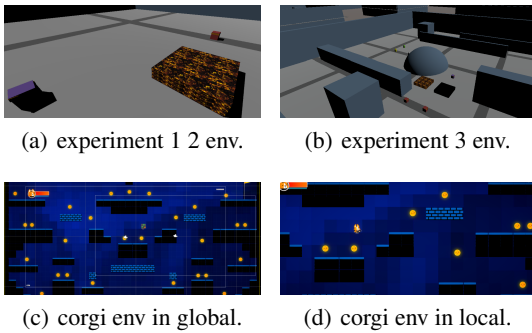


Figure 4: The unity 3D environments (a)(b) of experiments and the high fidelity makes it possible for agents to bring evidence towards the application of model. Details and discussion of experiments are released in the description section. Corgi environment (c)(d) with an extendable engine is a friendly 2D game where agents can finish some simple tasks.



(a) experiment 1 2 env.

(b) experiment 3 env.

Figure 5: The plane environments. Every tagged objects would be labelled by arrows. Four discrete actions only be taken by the agent to achieve the task move forward, move backward, turn left, turn right. Furthermore, the agent is provided by a view fan field composed by a number of ray sensors as the primary observations which can detect the corresponding objects. The apparent information of relative positions, rotations and distances, are added collectively up to 109 and 1490 observations.

to real-world situations for generality and veracity. In order to take a deep dive into the traits of MDRL-BT, the training models are configured with different constructions and components as comparison. The Unity ML-Agents Toolkit (Juliani et al., 2018), accessible to the wider research, serves as an open-source project and can be used to train intelligent agents through a simple-to-use Python API. In (Noblega et al., 2019), adaptable NPC-agent with PPO has been devised in Unity ML-Agents Toolkit environments as an enlightenment of experimental simulations.

### 5.1 Description

Inspired by (Sakr and Abdennadher, 2016) in which rescue and saving simulation involves task planning and realistic estimations, experiment 1 is established by extending simulated fire control scenarios (de Pontes Pereira and Engel, 2015) to a 3D environment in figure 4(a)(b) for agent training. Experiment 1 and 2 almost take place in an identical surroundings where independent RL can accomplish the benchmarks and the proposed MDRL-BT would combat the challenges of baselines. With regard to complex relationship in experiment 3, significant advances in performance are made by MDRL-BT irrespective of training and testing. And an agent attached with MDRL-BT on a 2D game about corgi engine is heightened by the driven-curiosity learning for a further expansion.

The standard of scores acquired by agent for evaluation is measured by the degree of target completion and the frequency of collisions with the walls. Simultaneously, the total time of each episode is collected separately for the estimation of completion speed.

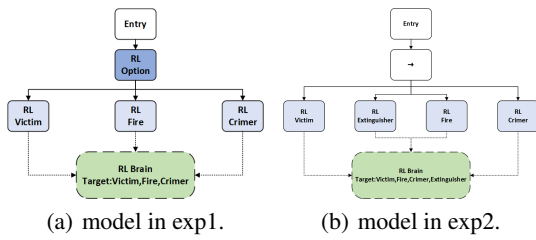


Figure 6: The models of two experiments. The solid lines are ticks flowing between nodes and the imaginary lines proceed with an interaction between blue nodes and green brain. The RL option in dark blue is a learning composite node in all experiments and the RL nodes in watery blue are learning action nodes aiming at sub-tasks like saving the victim. Other blank nodes are nothing but normal nodes.

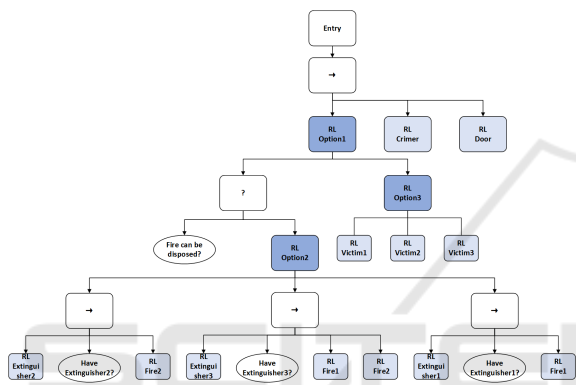


Figure 7: The mixed model described is composed of 12 watery blue PPO learning action nodes with a shared brain, 3 dark blue SAC composite learning options with different configurations and another 9 blank normal BT nodes.

In order to assess the results equally and exactly, the mean values of every experiment upon thirty-two thousand times are calculated. Consistent with the general learning process, incremental steps and rewards of feedback during the training are kept track of to understand the convergence.

**Experiment 1.** As enumerated in plane figure 5(a), there are four types of objects characterized by victim, fire, criminal and agent. The task refers to it that the agent is bound to save victim, extinguish the fire and catch criminal as soon as possible. For every episode, the agent is commanded to accomplish the task spontaneously but the four objects are initially placed or reset in a random pattern to eliminate the training contingency. This scenario is surrounded markedly by high walls in figure 5(a)(b) to prohibit stepping outside. Afterwards, the ground without friction appears so smooth that the agent with manual control is indeed difficult to manipulate in discrete action spaces.

As aforementioned, the state-of-arts of off-policy SAC (labelled as SAC) and on-policy PPO (PPO)

are implemented separately to establish a test baseline and opsive behavior designer is integrated by Unity NavMeshAgent to build a normal BT without the need of training (BT). To employ the versatile framework of MDRL-BT appropriately, learning action nodes at first are assigned as children of a sequence node in BT which is identical to the construction in figure 6(a) except option nodes. The sequence node as a core attempts to control the main process and the action nodes with SAC ( $BT_{sac}$ ) and PPO ( $BT_{ppo}$ ) opt to act from the learned strategy. In the cost of more observations and changing inputs, the different action nodes can be connected with a shared RL brain to speed up training in terms of the similar strategy.

Undoubtedly, the sequence node above with a constrained querying pattern may lead to a sub-optimal consequence on account of hardly inevitable order, which is also the universal self-imposed restrictions in general BT. We replace the sequence node by RL option node as the figure 6(a) shows, which is indicative of the breakthrough point of the limited structure. The core option node serves as a global decision maker to explore a better consequence, calculating specific target value and scheduling the expected queries with learning policy. Accordingly, there are two promising alternative options with PPO ( $Option_{ppo}$ ) and SAC ( $Option_{sac}$ ) for further training. For removing the impacts of learning action nodes, PPO action nodes aren't modified in the two models. Taking the time of training into consideration, we also use a previously trained PPO action node in the start to apply the option with SAC( $Option_{pre}$ ).

**Experiment 2.** On the foundation of the preceding subject in experiment 1, an extinguisher marked green is placed as a vital part of the environment and the sequential order of three independent tasks is demanded to confirm the positive features of BT in experiment 2. The agent certainly acquires an extinguisher intended for addressing the fire issue ahead of time, otherwise approaching the fire within certain distances leads to a punishment of reward and score. The model described in figure 6(b) is dominated by a sequence node in this restrictive and flexible circumstance. For comparative analysis, the training holds fixed steps of  $3 * 10^7$ .

**Experiment 3.** An increasingly complicated request arises that the agent struggles to undertake three victims saving and put out two types of fire previously, then catch a criminal and enter the door to restart a period lastly in figure 5(b). Meanwhile, three extinguishers with corresponding tags are associated with the assumption that extinguisher1 only deals with fire1, extinguisher2 only for fire2 but extin-



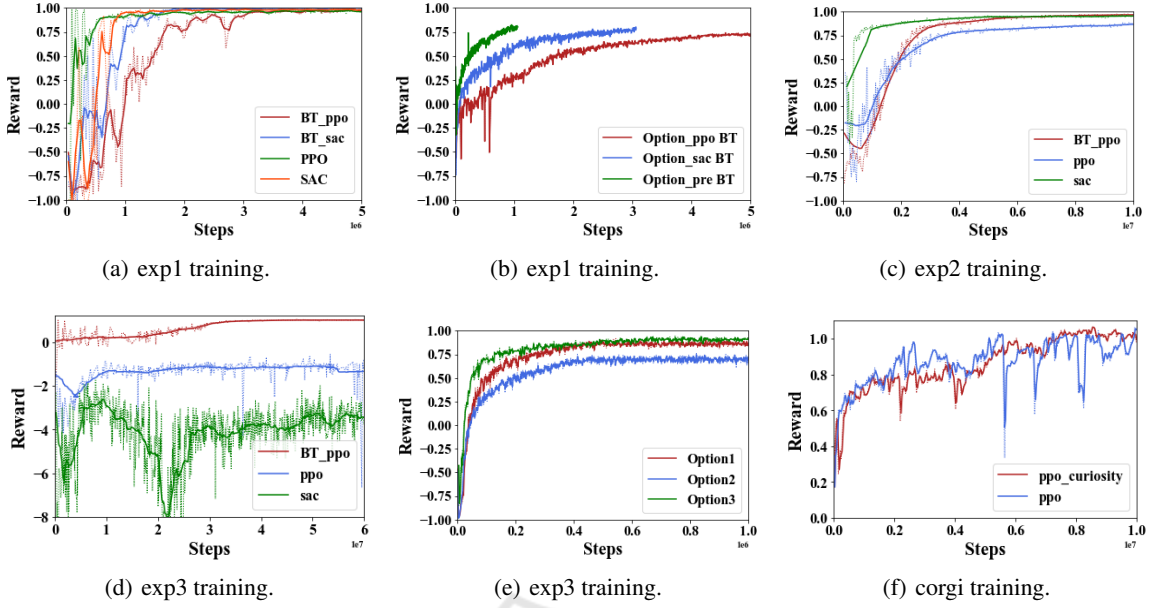


Figure 8: The training curve of experiments. Because learning composite options are different from learning action nodes in training frequency. So experiment 1 and experiment 3 would have two graphs. Experiment 2 and corigi without learning composite nodes only have one graph. Every curve with corresponding tag means an algorithm with the same conditions in a graph.

guisher3 for fire1 and fire2 both. That’s the same case that every object is generated arbitrarily in any corner of the environment and more walls are added to hamper the movement of the agent. What’s more, the victims, extinguishers and fire may be randomly absent at the outset of every episode. This highlights the dynamically changing of the environment and reaches the number of  $2^8$  kinds of different situations totally. In addition, the active criminal is moving all the time with a slow velocity and will stay away from the agent within a certain distance in the face of agent.

## 5.2 Results and Analysis

In this section, the performances for the trials are subjected to contrastive analysis. The examinations are carried out to corroborate the benefit of mixed components.

**PPO vs SAC.** Figure 8(a) indicates that independent PPO and SAC can rapidly converge. Although the reward of SAC with the preponderance of sample-efficient learning can get close to one in a short time and in contrast it takes a long time for PPO to arrive, it is apparent in table 1 and figure 9 that in the perspective of scores and time the PPO outperforms readily SAC due to the influence of the sensitive hyperparameters tuning and frequent policy updates of SAC in discrete action space. Therefore, it is hypothesized that PPO is more applicable to the fre-

quent movement manipulation of agents than SAC which is the same difference between  $PPO(BT_{ppo})$  and  $SAC(BT_{sac})$ .  $PPO(BT_{ppo})$  appears more stable and behaves as well as the independent RL on the table 1 but it takes more steps to converge. Adopting PPO nodes can yield the better objective of scores and time.

**MDRL-BT vs RL.** According to the quantitative analysis of training steps in experiment 2, the frame-

Table 1: Evaluation statistics. Exp is about experiment type. S and T respectively show scores and time. Mean means an average value. The full score reaches 100 and the unit of T is seconds.

| Exp | Model          | Mean S  | Mean T  | Train      |
|-----|----------------|---------|---------|------------|
| 1   | BT             | 92.4365 | 13.2821 | -          |
|     | PPO            | 96.5761 | 9.0265  | $10^7$     |
|     | SAC            | 95.4983 | 13.1198 | $10^7$     |
|     | $BT_{ppo}$     | 96.5513 | 9.5955  | $10^7$     |
|     | $BT_{sac}$     | 96.1467 | 11.2166 | $10^7$     |
|     | $Option_{ppo}$ | 97.4605 | 8.8977  | $5 * 10^6$ |
|     | $Option_{sac}$ | 97.7533 | 8.5394  | $3 * 10^6$ |
|     | $Option_{pre}$ | 97.7208 | 8.2931  | $1 * 10^6$ |
| 2   | PPO            | 71.2845 | 12.2972 | $3 * 10^7$ |
|     | SAC            | 84.8081 | 17.5410 | $3 * 10^7$ |
|     | $BT_{ppo}$     | 94.2582 | 12.3749 | $3 * 10^7$ |
| 3   | other          | -       | -       | $1 * 10^8$ |
|     | mixed          | 90.2595 | 19.5666 | $1 * 10^8$ |

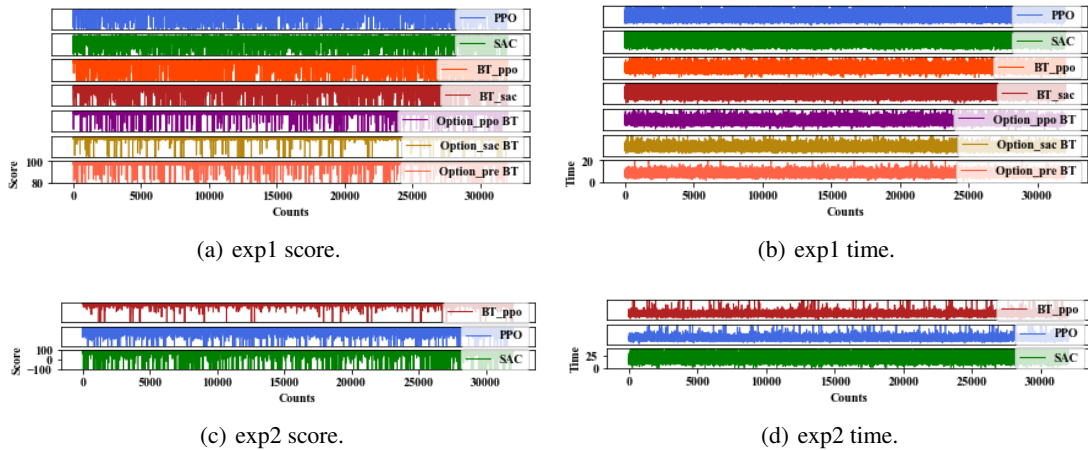


Figure 9: The distribution of 32000 tests suggests the average value and the standard deviation. The corresponding conclusion can be drawn clearly on the grounds of the results. In experiment 3, only the mixed model can get a positive score in a limited time. So the result isn't shown.

work for modeling constrained yet adaptive agents reveals the character of BT and surpasses the behavior of PPO and SAC quite a few. The sequential strategy time-consuming for the independent RL algorithms is exhibited favorably by MDRL-BT for simplification and efficiency.

**Option Nodes.** Table 1 especially shows that all models with learning core option and PPO action nodes can outperform the other methods and present an optima. Respectively,  $Option_{sac}$  with less training steps but thoroughly performs better than  $Option_{ppo}$  from figure 8(b) and table 1, because SAC is particularly appropriate for low frequency updates and its sample efficiency consequently exceeds PPO.  $Option_{pre}$  with pre-trained action nodes can almost keep in line with  $Option_{sac}$  in scores with the less steps. It is indicated that  $Option_{pre}$  can be a superior choice in complicated models for the reduction of training steps. To sum up, learning action nodes with PPO deals with complex environmental dynamics and option nodes with SAC quickly handle planning and scheduling in the construction of MDRL-BT.

**MDRL-BT vs Others.** The behavior of wall touching or breaking rules with a reduction of scores and rewards in experiment 3 makes the scenario noticeably troublesome and the agent must be in possession of some intelligence to manage and conduct its behavior across the environment. In figure 8(d), the independent PPO and SAC which beforehand fall into a local dilemma hardly proceed with training and a simple model of  $BT_{ppo}$  is incapable of achieving good performance due to the dynamic rewards and the punishment of far too much walls touching. Nevertheless, the MDRL-BT with the model in figure 7 successfully addresses the issues as table 1 and figure 8 (e) shows.

**2D.** On the basis of 3D experiment, we explore the 2D game environment in figure 4(c)(d) built by corgi engine to verify the other features of MDRL-BT model. The corgi agent with  $BT_{ppo}$  is required to collect the coins scattered all over the corners in a limited time. What's more, curiosity is employed in the MDRL-BT and in figure 8(f) both models can be trained well enough but curiosity (Burda et al., 2018) can get a little better result.

## 6 CONCLUSIONS

This paper has researched a mixed model for inventing an intelligent agent. We do some surveys on contextual backgrounds and related studies to explore the promotion of agent designs. Enough efforts about the combination of deep RL and BT have been made by digging deep into the theoretical basis and existing correlations. As a specialization of option-framework, MDRL-BT architecture is refined on the strength of deep learning nodes and BT construction. We accomplish the execution synchronization of RL and BT and define an appropriate rewards function to prescribe the desired decisions. Several virtual simulations are implemented on Unity 2D and 3D environments to employ semantics and structure of MDRL-BT. The mixed model also varies slightly with the complexity for displaying the special attributes.

The insights gained from results may be of assistance to intelligent agents. MDRL-BT, reflecting the integrated advantage in the theorem, empirically outweights the BT and RL and can be successfully applied to 2D and 3D environments. When especially faced with complicated affairs or sequential tasks, the

MDRL-BT keeps the mind of hierarchies by decomposing a main issue into several simple questions to provide a rational alternative solution. As evident from the result, MDRL-BT doesn't need elaborate reward design to guarantee the training convergence relative to general RL algorithms. In the design of mixed models, its a better choice to use PPO action nodes with a shared brain and SAC composite nodes, even pre-train nodes. So as to a real available application, general RL algorithms or normal BT can be used for simple tasks and by the way, MDRL-BT can be a candidate for complex problems.

MDRL-BT has a certain extensibility because of recursive BT framework and RL foundations. Sometimes further exploration for extending MDRL-BT by importing other mechanisms such as curiosity in the sparse reward distribution can be an exciting avenue. However, there will be enormous work to finish from the unobvious consequence. In the future work, the correlative theory and applicable scene about the additional algorithms can be investigated for better performance.

## REFERENCES

- Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*.
- Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrrell, T., and Efros, A. A. (2018). Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.
- de Pontes Pereira, R. and Engel, P. M. (2015). A framework for constrained and adaptive behavior-based agents. *arXiv preprint arXiv:1506.02312*.
- Dey, R. and Child, C. (2013). Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 1–8.
- Dromey, R. G. (2003). From requirements to design: formalizing the key steps. In *International Conference on Software Engineering and Formal Methods*.
- Florez-Puga, G., Gomez-Martin, M., Gomez-Martin, P., Diaz-Agudo, B., and Gonzalez-Calero, P. (2009). Query-enabled behavior trees. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):298–308.
- Fu, Y., Qin, L., and Yin, Q. (2016). A reinforcement learning behavior tree framework for game ai. In *2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering*, pages 573–579.
- Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICLR 2018 : International Conference on Learning Representations 2018*.
- Isla, D. (2005). Gdc 2005 proceeding: Handling complexity in the halo 2 ai. Retrieved October, 21:2009.
- Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Matar, M., and Lange, D. (2018). Unity: A general platform for intelligent agents. *arXiv:1809.02627*.
- Kartasev, M. (2019). Integrating reinforcement learning into behavior trees by hierarchical composition.
- Liessner, R., Schmitt, J., Dietermann, A., and Bker, B. (2019). Hyperparameter optimization for deep reinforcement learning in vehicle energy management. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 134–144. INSTICC, SciTePress.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- Mateas, M. and Stern, A. (2002). A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4):39–47.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Venness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- Noblega, A., Paes, A., and Clua, E. (2019). Towards adaptive deep reinforcement game balancing. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 693–700. INSTICC, SciTePress.
- Sakr, F. and Abdennadher, S. (2016). Harnessing supervised learning techniques for the task planning of ambulance rescue agents. In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART*, pages 157–164. INSTICC, SciTePress.
- Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. *arXiv preprint arXiv:1502.05477*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Den Driessche, G. V., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.
- Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference*

- on Machine Learning - Volume 32*, ICML'14, page 13871395. JMLR.org.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.
- Subagyo, W. P., Nugroho, S. M. S., and Sumpeno, S. (2016). Simulation multi behavior npcs in fire evacuation using emotional behavior tree. In *2016 International Seminar on Application for Technology of Information and Communication (ISemantic)*, pages 184–190.
- Sutton, R. S., Precup, D., and Singh, S. P. (1998). Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98*, page 556564, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhn-evets, A. S., Yeo, M., Makhzani, A., Kitter, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. (2017). Starcraft ii: A new challenge for reinforcement learning.
- Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115152.
- Zhang, Q., Sun, L., Jiao, P., and Yin, Q. (2017). Combining behavior trees with maxq learning to facilitate cgfs behavior modeling. In *2017 4th International Conference on Systems and Informatics (ICSAI)*, pages 525–531.