StarCraft Micromanagement with Reinforcement

Learning and Curriculum Transfer Learning

Оглавление

[Аннотация 2](#_Toc30084565)

[I. INTRODUCTION 2](#_Toc30084566)

[II. PROBLEM FORMULATION AND BACKGROUNDS 6](#_Toc30084567)

[A. Problem Formulation 6](#_Toc30084568)

[B. Reinforcement Learning 7](#_Toc30084569)

[C. Curriculum Transfer Learning 10](#_Toc30084570)

[III. LEARNING MODEL FOR MICROMANAGEMENT 11](#_Toc30084571)

[A. Representation of High-Dimension State 11](#_Toc30084572)

[B. Action Definition 13](#_Toc30084573)

[C. Network Architecture 13](#_Toc30084574)

[IV. LEARNING METHOD FOR MICROMANAGEMENT 14](#_Toc30084575)

[A. Parameter Sharing Multi-agent Gradient-Descent Sarsa(λ) 14](#_Toc30084576)

[B. Reward Function 17](#_Toc30084577)

[C. Frame Skip 18](#_Toc30084578)

[V. EXPERIMENT SETTINGS 18](#_Toc30084579)

[A. StarCraft Micromanagement Scenarios 18](#_Toc30084580)

[B. Training 19](#_Toc30084581)

[VI. RESULTS AND DISCUSSIONS 20](#_Toc30084582)

[A. Small Scale Micromanagement 20](#_Toc30084583)

[B. Large Scale Micromanagement 22](#_Toc30084584)

[C. Strategies Analysis 23](#_Toc30084585)

[VII. CONCLUSION AND FUTURE WORK 25](#_Toc30084586)

[ACKNOWLEDGMENT 26](#_Toc30084587)

# Аннотация

**Оригинал**

Real-time strategy games have been an important field of game artificial intelligence in recent years. This paper presents a reinforcement learning and curriculum transfer learning method to control multiple units in StarCraft micromanagement. We define an efficient state representation, which breaks down the complexity caused by the large state space in the game environment. Then a parameter sharing multi-agent gradient-descent Sarsa(λ) (PS-MAGDS) algorithm is proposed to train the units.

Стратегические игры в реальном времени является важной областью игр с искусственным интеллектом в последние годы. В этой статье представлен метод обучения с подкреплением и метод постепенного трансферного обучения, с помощью которых осуществляется управление множеством боевых единиц в StarCraft. Мы определяем эффективное представление состояний, которое устраняет сложность, вызванную большим пространством состояний в игровой среде. Затем для обучения юнитов предлагается алгоритм с разделением параметром между агентами с градиентным спуском Сарса (λ лямбда) (PS-MAGDS).

The learning policy is shared among our units to encourage cooperative behaviors. We use a neural network as a function approximator to estimate the action-value function, and propose a reward function to help units balance their move and attack. In addition, a transfer learning method is used to extend our model to more difficult scenarios, which accelerates the training process and improves the learning performance. In small scale scenarios, our units successfully learn to combat and defeat the built-in AI with 100% win rates.

Политика обучения распространяется среди наших юнитов, чтобы поощрять совместное поведение. Мы используем нейронную сеть в качестве аппроксиматора функции для оценки функции «действие-значение» и предлагаем функцию вознаграждения, которая помогает юнитам сбалансировать их движение и атаку. Кроме того, метод трансферного обучения используется для расширения нашей модели до более сложных сценариев, что ускоряет процесс обучения и повышает эффективность обучения. В мелкомасштабных сценариях наши юниты успешно учатся сражаться и побеждать встроенный ИИ со 100% вероятностью выигрыша.

In large scale scenarios, curriculum transfer learning method is used to progressively train a group of units, and shows superior performance over some baseline methods in target scenarios. With reinforcement learning and curriculum transfer learning, our units are able to learn appropriate strategies in StarCraft micromanagement scenarios.

В крупномасштабных сценариях метод постепенного обучения используется для постепенного обучения группы юнитов и показывает превосходную производительность по сравнению с некоторыми базовыми методами в целевых сценариях. Благодаря обучению с подкреплением и постепенному трансферному обучению наши юниты способны выучить подходящие стратегии в сценариях микроуправления StarCraft.

Index Terms—reinforcement learning, transfer learning, curriculum learning, neural network, game AI.

# I. INTRODUCTION

ARTIFICIAL intelligence (AI) has a great advance in the last decade. As an excellent testbed for AI research, games have been helping AI to grow since its birth, including the ancient board game [1], [2], [3], [4], the classic Atari video games [5], [6], and the imperfect information game [7]. These games have a fixed, limited set of actions, and researchers only need to control a single agent in game environment. Besides, there are a large number of games including multiple agents and requiring complex rules, which are much more difficult for AI research.

Искусственный интеллект значительно улучшился за последнее десятилетие. Являясь отличным испытательным полигоном для исследований ИИ, игры помогают ИИ развиваться с самого его зарождения, включая древние настольные игры (го и гомоку)[1], [2], [3], [4], классические видео игры Атари, и игры с неполной информацией(в 7 говорится об ИИ для покера). Эти игры имеют фиксированный, ограниченный набор действий и исследователи могут управлять только одним агентом в игровой среде. С другой стороны, существует большое количество игр, включающих несколько агентов и требующих сложных правил, которые намного сложнее для исследования ИИ

In this paper, we focus on a real-time strategy (RTS) game to explore the learning of multi-agent control. RTS games are usually running in real-time, which is different from taking turns to play in board games [8]. As one of the most popular RTS games, StarCraft has a huge player base and numerous professional competitions, requiring different strategies, tactics and reactive control techniques. For the game AI research, StarCraft provides an ideal environment to study the control of multiple units with different difficulty levels [9].

В этой статье, мы фокусируемся на стратегиях реального времени для исследования обучения мультиагентному управлению. RTS игры обычно запускаются в режиме реального времени, что отличается их от настольных игр, где ходят по очереди[8]. Как в одной из самых популярных RTS игр, в Starcraft огромная база игроков и многожество профессиональных соревнований, требующих различных стратегий, тактик и методов реактивного управления. Для исследования игрового ИИ StarCraft предоставляет идеальную среду для изучения управления несколькими юнитами с различными уровнями сложности [9].

In recent years, the study of StarCraft AI has an impressive progress, driven by some StarCraft AI competitions and Brood War Application Programming Interface (BWAPI) [10]. Recently, researchers have developed more efficient platforms to promote the development of this field, including TorchCraft, ELF and PySC2. StarCraft AI aims at solving a series of challenges, such as spatial and temporal reasoning, multi-agent collaboration, opponent modeling and adversarial planning [8].

В последние годы исследования ИИ для StarCraft достигли впечатляющих успехов благодаря соревнованиям ИИ для StarCraft и интерфейсу для прикладного программирования для Starcraft Brood War (BWAPI) [10]. Недавно исследователи разработали более эффективные платформы для содействия развитию этой области, включая TorchCraft, ELF и PySC2. StarCraft ИИ направлен на решение ряда задач, таких как пространственные и временные закономерности(рассуждения), совместная работа нескольких агентов, моделирование оппонентов и планирование состязаний [8].

At present, designing a game AI for the full StarCraft game based on machine learning method is out-of- reach. Many researchers focus on micromanagement as the first step to study AI in StarCraft [11]. In combat scenarios, units have to navigate in highly dynamic environment and attack enemies within fire range. There are many methods for StarCraft micromanagement, including potential fields for spatial navigation and obstacle avoidance [12], [13], bayesian modeling to deal with incompleteness and uncertainty in the game [14], heuristic game-tree search to handle both build order planning and units control [15], and neuroevolution to control individual unit with hand-craft features [16]

В настоящее время не разработано ни одного игрового ИИ для полноценной игры в StarCraft на основе методов машинного обучения. Многие исследователи фокусируются на микроуправлении как на первом этапе изучения ИИ в StarCraft [11]. В боевых сценариях юниты должны перемещаться в высоко динамичной среде и атаковать врагов в радиусе атаки. Существует много методов для микроуправления StarCraft, включая потенциальные поля для пространственной навигации и обхода препятствий [12], [13], байесовское моделирование для решения проблем неполной информации и неопределенности в игре [14], эвристический поиск по дереву игр для обработки последовательности заказов юнитов и зданий, и управления юнитами [15], а также нейроэволюция для управления отдельными юнитами с рукотворными улучшениями [16]

. As an intelligent learning method, reinforcement learning (RL) is very suitable for sequential decision-making tasks. In StarCraft micromanagement, there are some interesting appli- cations with RL methods. Shantia et al. use online Sarsa and neural-fitted Sarsa with a short term memory reward function to control units’ attack and retreat [17]. They use vision grids to obtain the terrain information. This method needs a hand- craft design, and the number of input nodes has to change with the number of units. Besides, they apply an incremental learning method to scale the task to a larger scenario with 6 units. However, the win rate with incremental learning is still below 50%. Wender et al. use different RL algorithms in micromanagement, including Q learning and Sarsa [18]. They control one powerful unit to play against multiple weak units, without cooperation and teamwork between own units.

Как интеллектуальный метод обучения, обучение с подкреплением (RL) очень подходит для последовательных задач принятия решений. В микроуправлении StarCraft есть несколько интересных приложений с методами RL. Шантия и соавт. Используют алгоритм онлайн Сарса и нейронную версию Сарсы с функцией кратковременного поощрения, чтобы контролировать нападение и отступление юнитов [17]. Они используют сетку видимости для получения информации о местности. Этот метод требует ручной разработки, а количество входных узлов должно меняться в зависимости от количества единиц. С другой стороны, они применяют метод инкрементного обучения, чтобы масштабировать задачу до более крупного сценария с 6 юнитами. Тем не менее, процент выигрышей при инкриментном обучении все еще ниже 50%. Вендер и соавт. используют различные RL-алгоритмы в микроуправлении, в том числе Q-обучение и Сарса [18]. Они управляют одним мощным юнитом, чтобы играть против нескольких слабых юнитов, без сотрудничества и совместной работы со своими юнитами.

In the last few years, deep learning has achieved a remarkable performance in many complex problems [19], and has dramatically improved the generalization and scalability of traditional RL algorithms [5]. Deep reinforcement learning (DRL) can teach agents to make decisions in high-dimension state space by an end-to-end method. Usunier et al. propose an RL method to tackle micromanagement with deep neural network [20]. They use a greedy MDP to choose actions for units sequentially at each time step, with zero-order optimiza- tion to update the model. This method is able to control all units owned by the player, and observe the full state of the game.

В последние несколько лет глубокое обучение достигло замечательных результатов во многих сложных задачах [19] и значительно улучшило обобщение и масштабируемость традиционных RL-алгоритмов[5]. Глубокое обучение с подкреплением (DRL) может научить агентов принимать решения в пространстве состояний высокой размерности сквозным/непосредственным методом. Усиниер и соавторы предлагают RL-метод для решения проблемы микроуправления с помощью глубокой нейронной сети [20]. Они используют жадный MDP для последовательного выбора действий для юнитов на каждом временном шаге с оптимизацией нулевого порядка для обновления модели. Этот метод способен контролировать всех юнитов, принадлежащих игроку, и наблюдать за полным состоянием игры.

Peng et al. use an actor-critic method and recurrent neural networks (RNNs) to play StarCraft combat games [21]. The dependency of units is modeled by bi-directional RNNs in hidden layers, and its gradient update is efficiently propagated through the entire networks. Different from Usunier’s and Peng’s work that design centralized controllers, Foerster et al. propose a multi-agent actor-critic method to tackle decen- tralized micromanagement tasks, which significantly improves the performance over centralized RL controllers [22].

Пэнг и соавт. используют метод критики агентов и рекуррентные нейронные сети (RNN), для военных игр в StarCraft [21]. Зависимость юнитов моделируется двунаправленной рекуррентной нейронной сетью в скрытом слое, и обновление её градиента эффективно распространяется по всей сети. Отличие работ Усиниер и Пэнг в том как реализуется централизация контроллеров, Фоэстер и соавторы предлагают многоагентный метод с критикой агентов для решения децентрализованных задач микроуправления, который значительно повышает производительность по сравнению с централизованными RL -контроллерами[22].

For StarCraft micromanagement, traditional methods have difficulties in handling complicated state and action space, and learning cooperative tactics. Modern methods rely on strong compute capability introduced by deep learning. Be- sides, learning micromanagement with model-free RL methods usually needs a lot of training time, which is even more serious in large scale scenarios. In this paper, we dedicate to explore more efficient state representation to break down the complexity caused by the large state space, and propose appropriate RL algorithm to solve the problem of multi-agent decision making in StarCraft micromanagement. In addition, we introduce curriculum transfer learning to extend the RL model to various scenarios and improve the sample efficiency.

Для микроуправления StarCraft традиционные методы испытывают трудности при обработке сложного пространства состояний и действий, а также в обучении тактике сотрудничества. Современные методы основаны на сильных вычислительных возможностях, которые дает глубокое обучение. Кроме того, для изучения микроуправления с помощью безмодельных RL методов, как правило, требуется много времени на обучение, что является еще более серьезной проблемой в крупномасштабных сценариях. В этой статье мы рассматриваем более эффективное представление состояний, чтобы убрать сложность, вызванную большим пространством состояний, и предлагаем соответствующий RL-алгоритм для решения проблемы многоагентного принятия решений в микроуправлении StarCraft. К тому же мы вводим постепенное трансферное обучение, чтобы расширить RL-модель для различных сценариев и повысить эффективность выборки.

The main contributions emphasize in three parts. First, we propose an efficient state representation method to deal with the large state space in StarCraft micromanagement. This method takes units’ attributes and distances into consideration, allowing an arbitrary number of units on both sides.

Основные новшества показаны в трех частях. Во-первых, мы предлагаем эффективный метод представления состояний для работы с большим пространством состояний в микроуправлении StarCraft. Этот метод учитывает атрибуты юнитов и расстояния между ними, что позволяет иметь произвольное количество юнитов обеим сторонам.

Compared with related work, our state representation is more concise and more efficient. Second, we present a parameter sharing multi-agent gradient-descent Sarsa(λ) (PS- MAGDS) algorithm to train our units. Using a neural network as a function approximator, agents share the parameters of a centralized policy, and update the policy with their own experiences simultaneously.

По сравнению с аналогичными работами наше представление состояния значительно меньше и эффективнее. Во-вторых, мы представляем алгоритм с разделением параметров между агентами с градиентным спуском Сарса (λ лямбда) (PS-MAGDS) для обучения наших юнитов. Используя нейронную сеть в качестве аппроксиматора функций, агенты обмениваются параметрами централизованной политики и одновременно обновляют политику в соответствии с собственным опытом.

This method trains homogeneous agents efficiently, and encourages cooperative behaviors. To solve the problem of sparse and delayed rewards, we introduce a reward function including small intermediate rewards in the RL model. This reward function improves the training process, and serves as an intrinsic motivation to help units collaborate with each other.

Этот метод эффективно обучает однородных агентов и поощряет командное поведение. Чтобы решить проблему редких и отсроченных вознаграждений, мы вводим функцию вознаграждения, включающую небольшие промежуточные вознаграждения в RL-модели. Эта функция вознаграждения улучшает процесс обучения и служит внутренней мотивацией, которая помогает юнитам сотрудничать друг с другом.

Third, we propose a transfer learning method to extend our model to various scenarios. Compared with learning from scratch, this method accelerates the training process and improves the learning performance to a great extent. In large scale scenarios, we apply curriculum transfer learning method to successfully train a group of units. In term of win rates, our proposed method is superior to some baseline methods in target scenarios.

В-третьих, мы предлагаем метод трансферного обучения, чтобы расширить нашу модель для различных сценариев. По сравнению с обучением с нуля этот метод ускоряет процесс обучения и в значительной степени повышает эффективность обучения. В крупномасштабных сценариях мы применяем метод постепенного трансферного обучения, чтобы успешно обучать группу юнитов. Что касается процента побед, предлагаемый нами метод превосходит некоторые базовые методы в целевых сценариях.

The rest of the paper is organized as follows. In Section II, we describe the problem formulation of StarCraft micro- management, as well as backgrounds of reinforcement learning and transfer curriculum learning. In Section III, we present the reinforcement learning model for micromanagement, including state representation method, network architecture and action definition.

Оставшаяся часть статьи организована следующим образом. В разделе II мы описываем формулировку проблемы микроуправления в StarCraft, а также основы обучения с подкреплением и постепенное трансферное обучение. В Разделе III мы представляем модель обучения с подкреплением для микроуправления, включая метод представления состояния, архитектуру нейросети и определение действий.

And in Section IV, we introduce the parameter sharing multi-agent gradient-descent Sarsa(λ) algorithm and the reward function. In Section V, we introduce the StarCraft micromanagement scenarios used in our paper and the training details. In Section VI, we make an analysis of experimental results and discuss the learned strategies. In the end, we draw a conclusion of the paper and propose some future work.

А в Разделе IV мы представляем алгоритм с разделением параметром между агентами с градиентным спуском Сарса(λ) и функцию вознаграждения. В разделе V мы представляем сценарии микроуправления StarCraft, используемые в нашей статье, и подробности обучения. В разделе VI мы проводим анализ результатов эксперимента и обсуждаем изученные стратегии. В конце мы подведем итоги работы и предложим некоторые будущие работы.

# II. PROBLEM FORMULATION AND BACKGROUNDS

## A. Problem Formulation

In StarCraft micromanagement, we need to control a group of units to destroy the enemies under certain terrain conditions. The combat scenario with multiple units is approximated as a Markov game, a multi-agent extension of Markov decision processes (MDPs) [21], [22], [23]. In a Markov game with N agents, a set of states S are used to describe the properties of all agents and the environment, as well as a set of actions A 1 , ..., A N and observations O 1 , ..., O N for each agent.

В микроуправлении StarCraft нам нужно управлять группой юнитов, чтобы уничтожать врагов при определенных условиях местности. Боевой сценарий с несколькими юнитами аппроксимируется как игра Маркова, мультиагентным расширением марковских процессов принятия решений (MDP) [21], [22], [23]. В марковской игре с N агентами, набор состояний S используется для описания свойств всех агентов и среды, а также набора действий A1, ..., AN и наблюдений O1, ..., O N для каждого агента.

In the combat, units in each side need to cooperate with each other. Developing a learning model for multiple units is challenging in micromanagement. In order to maintain a flexible framework and allow an arbitrary number of units, we consider that our units have access to the state space S from its own observation of the current combat by treating other units as part of the environment S → O i . Each unit interacts in the combat environment with its own observation and action. S × A 1 × ... × A N → S 0 denotes the transition from state S to the successive state S 0 with actions of all the units, and R 1 ...R N are the generated rewards of each unit. For the sake of multi-agent cooperation, the policy is shared among our units. The goal of each unit is to maximize its total expected return.

В бою юниты c каждой стороны должны кооперироваться друг с другом. Разработка модели обучения для нескольких юнитов является сложной задачей микроуправления. Чтобы поддерживать гибкую структуру и допускать произвольное количество юнитов, мы считаем, что наши юниты должны иметь доступ к пространству состояний S из собственного наблюдения за текущим боем, рассматривая других юнитов как часть среды S → O i. Каждый юнит взаимодействует в боевой обстановке исходя из своих наблюдений и действий. S × A 1 × ... × A N → S 0 обозначает переход из состояния S в следуещее состояние S0 с действиями всех подразделений, а R1 ... RN - сгенерированные вознаграждения каждого юнита. Что касается много-агентной кооперации, политика распространяется среди наших юнитов. Цель каждого юнита - максимизировать его общий ожидаемый результат.

## B. Reinforcement Learning

To solve the multi-agent control problem in StarCraft micromanagement, we can resort to reinforcement learning. Reinforcement learning is a type of machine learning algorithms in which agents learn by trial and error and determine the ideal behavior from its own experience with the environment [24]. We draw the classic RL diagram in Fig. 1. It shows the process that an RL agent interacts with the environment. The agent-environment interaction process in RL is formulated as a Markov decision process. The agent in state s makes an action A according to the policy π.

This behavior causes a reward r, and transfers to a new s 0 . We define the future discounted return at time t as ****, where T is the terminal time step and γ *∈* [0, 1] is a discount factor that determines the importance of future rewards. The aim of an RL model is to learn an optimal policy π, which defines the probability of selecting action a in state s, so that the sum of the overall discounted rewards is maximized, as demonstrated by

Чтобы решить проблему мультиагентного управления в микроменджменте StarCraft, мы можем прибегнуть к обучению с подкреплением. Обучение с подкреплением - это тип алгоритмов машинного обучения, в котором агенты учатся методом проб и ошибок и определяют идеальное поведение на основе своего собственного опыта работы с окружающей средой [24]. Мы рисуем классическую RL-диаграмму на рис. 1. Она показывает процесс взаимодействия RL-агента с окружающей средой. Процесс взаимодействия агент-среда в RL сформулирован как процесс принятия решений Маркова. Агент в состоянии s выполняет действие A согласно политике π.

Такое поведение вызывает вознаграждение r и переводит в новое состояние s0. Мы определяем будущий дисконтированный результат в момент времени t как ****, где T - конечный временной шаг, а γ *∈* [0, 1](гамма) - коэффициент дисконтирования, который определяет важность будущих вознаграждений. Цель RL-модели состоит в том, чтобы изучить оптимальную политику π, которая определяет вероятность выбора действия a в состоянии s, чтобы сумма всех дисконтированных вознаграждений была максимизирована, как показано

 (1)

As one of the most popular RL algorithms, temporal difference (TD) learning is a combination of Monte Carlo method and dynamic programming method. TD method can learn from raw experience without a model of the environment, and update estimates based on part of the sequence, without waiting for a final outcome [25]. The most widely known TD learning algorithms are Q-learning and Sarsa. Q-learning estimates the value of making an action in a given state and iteratively updates the Q-value estimate towards the observed reward. The TD error δ t in Q-learning is computed as



Как один из самых популярных алгоритмов RL, обучение с разницей во времени (TD) представляет собой комбинацию метода Монте-Карло и метода динамического программирования. Метод TD может учиться на основе необработанного опыта без модели среды и обновлять оценки на основе части последовательности, не дожидаясь окончательного результата [25]. Наиболее широко известными алгоритмами обучения TD являются Q-learning и Sarsa. Q-learning оценивает ценность совершения действия в данном состоянии и итеративно обновляет оценку Q-значения в соответствии с наблюдаемым вознаграждением. Ошибка TD δ(дельта) t в Q-обучении вычисляется как



Q-learning is an off-policy learning method, which means it learns a different policy compared with the one choosing actions. Different from Q-learning’s off-policy mechanism, Sarsa is an on-policy method, which means the policy is used both for selecting actions and updating previous Q-value [24]. The Sarsa update rule is demonstrated as



Q-learning - это метод обучения вне политики, что означает, что он изучает различные политики сравнивая с одним выбранным действем. В отличие от механизма вне-политичного Q-learning, Sarsa - это метод, основанный на политике, что означает, что политика используется как для выбора действий, так и для обновления предыдущего значения Q [24]. Правило обновления Sarsa демонстрируется как



where α is the learning rate. Traditional reinforcement learning methods have some successful applications, including TD in Backgammon [26] and adaptive dynamic programming (ADP) in control [27], [28], [29].

где α - скорость обучения. Традиционные методы обучения с подкреплением имеют несколько успешных применений, в том числе TD в нардах [26] и адаптивное динамическое программирование (ADP) в управлении [27], [28], [29].

Reinforcement learning with deep neural networks function approximators has received great attentions in recent years. DRL provides an opportunity to train a single agent to solve a series of human-level tasks by an end-to-end manner [30] [31]. As the most famous DRL algorithm, deep Q-network (DQN) uses the experience replay technique and a target network to remove the correlations between samples and stabilize the training process [5]. In the last few years, we have witnessed a great number of improvements on DQN, including double DQN [32], prioritised DQN [33], dueling DQN [34], distributed DQN [35] and asynchronous DQN [36].

В последние годы большое внимание уделяется обучению с подкреплением с глубокой нейронно-сетевой(deep neural networks, DRL) функцией апроксиматора. DRL предоставляет возможность обучить агентов для решения ряда задач на уровне человека непосредственным способом [30] [31]. Как самый известный алгоритм DRL, глубокая Q-сеть (DQN) использует технику воспроизведения опыта и целевую сеть для удаления корреляций между выборками и стабилизации процесса обучения [5]. За последние несколько лет мы стали свидетелями большого числа улучшений DQN, включая двойной DQN [32], приоритетной DQN [33], дуэльной DQN [34], распределенной DQN [35] и асинхронной DQN [36].

Apart from value-based DRL methods like DQN and its variants, policy- based DRL methods use deep networks to parameterize and optimize the policy directly [37]. Deep deterministic policy gradient (DDPG) is the continuous analogue of DQN, which uses a critic to estimate the value of current policy and an actor to update the policy [38]. Policy-based DRL methods play important roles in continuous control, including asynchronous advantage actor-critic (A3C) [36], trust region policy optimization (TRPO) [39], proximal policy optimization (PPO) [40], and so on.

Помимо основанных на значениях методов DRL, таких как DQN и его вариантов, методы DRL на основе политики используют глубокие сети для параметризации и прямой оптимизации политики [37]. Глубокий детерминированный градиент политики (Deep deterministic policy gradient, DDPG) - это доработанный аналог DQN, который использует критику для оценки ценности текущей политики и агента для обновления политики [38]. Методы DRL на основе политик играют важную роль в непрерывном контроле, включая асинхронно преимущественную критику агента (asynchronous advantage actor-critic (A3C)) [36], оптимизацию политики в области доверия (trust region policy optimization , TRPO) [39], оптимизацию проксимальной политики (proximal policy optimization, PPO) [40] и так далее.

The sample complexity of traditional DRL methods tends to be high, which limits these methods to real-world applications. While model-based DRL approaches learn value function and policy in a data-efficient way, and have been widely used in sensorimotor control. Guided policy search (GPS) uses a supervised learning algorithm to train policy and an RL algorithm to generate guiding distributions, allowing to train deep policies efficiently [41]. Researchers have also proposed some other model-based DRL methods, like normalized advantage functions (NAF) [42] and embed to control (E2C) [43].

*Образец* сложности традиционных методов DRL, как правило, высок, что ограничивает эти методы для реальных приложений. В то время как основанные на модели подходы DRL изучают значение функции и политику эффективных с точки зрения данных способов, они широко используются в сенсомоторном управлении. Поиск руководствующих политик (Guided policy search, GPS) использует контролирующий алгоритм обучения для тренировки политики и алгоритм RL генерирует направляющие распределения, что позволяет эффективно обучать глубоким политикам [41]. Исследователи также предложили некоторые другие основанные на модели методы DRL, такие как нормализованные функции преимущества (normalized advantage functions, NAF) [42] и встраивание в контроль (embed to control, E2C) [43].

Multi-agent reinforcement learning is a closely related area to our work [44]. A multi-agent system includes a number of agents interacting in one environment [45] [46]. Recently, some multi-agent reinforcement learning algorithms with deep neural network are proposed to learn communication [47], cooperative-competitive behaviors [23] and imperfect infor- mation [48]. In our work, we use a multi-agent reinforcement learning method with policy sharing among agents to learn cooperative behaviors. Agents share the parameters of a cen- tralized policy, and update the policy with their own expe- riences simultaneously. This method can train homogeneous agents more efficiently [49].

Многоагентное обучение с подкреплением тесно связанно с нашей областью работы[44]. Многоагентная система включает несколько агентов, взаимодействующих в одной среде [45] [46]. Недавно были предложены некоторые алгоритмы обучения с многоагентным подкреплением с глубокой нейронной сетью для обучения коммуникации [47], кооперативно-конкурентному поведению [23] и несовершенной информации [48]. В нашей работе мы используем многоагентный метод обучения с подкреплением с политикой разделения между агентами для изучения кооперативного поведения. Агенты разделяют параметры централизованной политики и одновременно обновляют политику в соответствии со своим опытом. Этот метод может обучать гомогенных агентов более эффективно [49].

## C. Curriculum Transfer Learning

Generally speaking, model-free reinforcement learning methods need plenty of samples to learn an optimal policy. However, many challenging tasks are difficult for traditional RL methods to learn admissible policies in large state and action space. In StarCraft micromanagement, there are numer- ous scenarios with different units and terrain conditions.

Вообще говоря, безмодельные методы обучения с подкреплением нуждаются в большом количестве примеров, чтобы изучить оптимальную политику. Тем не менее, многие сложные задачи сложны для традиционных RL-методов, чтобы изучить допустимые политики в большом пространстве состояний и действий. В микроуправлении StarCraft существует множество сценариев с различными юнитами и условиями местности.

It will take a lot of time to learn useful strategies in different scenarios from scratch. A number of researchers focus on improving the learning speed and performance by exploiting domain knowledge across various but related tasks. The most widely used approach is transfer learning (TL) [50] [51].

Это займет много времени, чтобы изучить полезные стратегии в различных сценариях с нуля. Ряд исследователей сосредоточены на повышении скорости и производительности обучения, используя знания предметной области для решения различных, но связанных задач. Наиболее широко используемым подходом является трансферное обучение (TL) [50] [51].

To some extent, transfer learning is a kind of generalization across tasks, transferring knowledge from source tasks to target tasks. Besides, transfer learning can be extended to RL problems by using the model parameters in the same model architecture [52]. The procedure of using transfer learning in our experiments is training the model with RL method in a source scenario first. And then, we can use the well-trained model as a starting point to learn micromanagement in target scenarios.

В некоторой степени, трансферное обучение является своего рода обобщением между задачами, передачей знаний от исходных задач к целевым задачам. Кроме того, трансферное обучение может быть распространено на RL-задачи с использованием параметров модели в той же архитектуре модели [52]. Процедура использования трансферного обучения в наших экспериментах - это сначала обучение модели RL-методом в исходном сценарии. И затем мы можем использовать хорошо обученную модель в качестве отправной точки для изучения микроуправления в целевых сценариях.

As a special form of transfer learning, curriculum learning involves a set of tasks organized by increasing level of difficul- ties. The initial tasks are used to guide the learner so that it can perform better on the final task [53]. Combining curriculum learning and transfer learning, curriculum transfer learning (CTL) method has shown good performance to help the learning process converge faster and towards better optimum in recent work [54], [55], [56].

Как особая форма трансферного обучения, обучение по плану включает в себя набор задач, организованных в соответствии с возрастающим уровнем сложности. Начальные задания используются для того, чтобы направлять учащегося, чтобы он смог лучше выполнять заключительное задание [53]. Комбинируя обучение по плану и трансферное обучение, метод плановоого трансферного обучения (CTL) показал хорошие результаты, помогая процессу обучения быстрее сходиться и достигать оптимума в недавних работах[54], [55], [56].

For micromanagement, a feasible method of using CTL is mastering a simple scenario first, and then solving difficult scenarios based on this knowledge. By changing the number and type of units, we could control the difficulty of micromanagement. In this way, we can use CTL to train our units with a sequence of progressively difficult micromanagement scenarios, as shown in Fig. 2.

Для микроуправления, наиболее удобный метод использования CTL - это сначала освоение простого сценария, а затем решение сложных сценариев на основе этих знаний. Изменяя количество и тип юнитов, мы можем контролировать сложность микроуправления. Таким образом, мы можем использовать CTL для обучения наших юнитов с последовательностью постепенно усложняющихся сценариев микроуправления, как показано на рис. 2.

# III. LEARNING MODEL FOR MICROMANAGEMENT

## A. Representation of High-Dimension State

State representation of StarCraft is still an open problem with no universal solution. We construct a state representation with inputs from the game engine, which have different data types and dimensions, as depicted in Table I. The proposed state representation method is efficient and independent of the number of units in the combat. In summary, the state representation is composed of three parts: the current step state information, the last step state information and the last step action, as shown in Fig. 3.

Представление состояний в StarCraft остается открытой проблемой без универсального решения. Мы строим представление состояний с помощью входных данных из игрового движка, которые имеют разные типы данных и измерения, как показано в таблице I. Предложенный метод представления состояния эффективен и не зависит от количества юнитов в бою. Таким образом, представление состояния состоит из трех частей: информация о состоянии текущего шага, информация о состоянии последнего шага и действия последнего шага, как показано на рис. 3.

Таблица 1

R это вещественное число, cat. это категория действия закодированное одним состоянием

The current step state information includes own weapon’s cooldown time, own unit’s hitpoint, distances information of own units, distances information of enemy units and distances information of terrain. The last step state information is the same with the current step. We take the last step action into consideration, which has been proven to be helpful for the learning process in the RL domain [57], [58].

Информация о состоянии текущем шага включает в себя время перезарядки собственного оружия, количество очков здоровья, информацию о расстояниях до своих юнитов, информацию о расстоянии до вражеских юнитов и информацию о расстоянии до объектов местности. Информация о состоянии последнего шага совпадает с текущим шагом. Мы принимаем к сведению последнее действие, которое оказалось полезным для процесса обучения в области RL [57], [58].

The proposed state representation method also has good generalization and can be used in other combat games, which need to take agents’ property and distance information into consideration.

Предложенный метод представления состояния также имеет хорошее обобщение и может использоваться в других боевых играх, в которых необходимо учитывать свойства юнитов и информацию о расстоянии между ними.

All inputs with real type are normalized by their maximum values. Among them, CoolDown and HitPoint have 1 dimen- sion for each. We divide the combat map into 8 sector areas on average, and compute the distances information in each area. Units’ distance information is listed as follows:

Все входы с вещественным типом нормированы по их максимальным значениям. Среди них CoolDown и HitPoint имеют 1 измерение для каждого. Мы делим боевую карту на 8 секторов в среднем и вычисляем информацию о расстояниях в каждой области. Информация о расстоянии между единицами указана следующим образом:

* OwnSumInfo: own units’ distances are summed in each area;
* OwnMaxInfo: own units’ distances are maximized in each area;
* EnemySumInfo: enemy units’ distances are summed in each area;
* EnemyMaxInfo: enemy units’ distances are maximized in each area.
* • OwnSumInfo: расстояния между собственными юнитами суммируются в каждой области;  
  • OwnMaxInfo: расстояния между собственными юнитами максимизированы в каждой области;  
  • EnemySumInfo: расстояния между вражескими юнитами суммируются в каждой области;  
  • EnemyMaxInfo: расстояния между вражескими юнитами максимизированы в каждой области.

If a unit is out of the central unit’s sight range D, the unit’s distance value dis unit is set to 0.05. Otherwise, the value is linear with d, the distance to the central unit, as demonstrated in equation (4).

Если юнит находится вне зоны видимости центрального юнита D, значение расстояния юнита устанавливается на 0,05. В противном случае значение имеет линейный вид с коэффициентом d, расстояние до центрального юнита, как показано в уравнении (4).

(

0.05, d > D

dis unit(d) =

(4)

1 − 0.95(d/D), d ≤ D

In addition, terrain distance value dis terrain is also computed in 8 sector areas. If the obstacle is out of the central unit’s sight range, the value is set to 0. Otherwise, the value is also linear with the distance to the central unit, as shown in equation (5).

Кроме того, значение расстояния между объектами местности рассчитывается также в 8 секторах. Если препятствие находится вне зоны видимости центрального юнита, значение устанавливается равным 0. В противном случае значение также будет линейным с расстоянием до центрального юнита, как показано в уравнении (5).

(

0, d > D

dis terrain(d) =

(5)

1 − d/D, d ≤ D

In this way, the current step state information has 42 dimen- sions. The last step action has 9 dimensions, with selected action setting to 1 and the other actions setting to 0. In total, the state representation in our model is embedded to 93 dimensions.

Таким образом, информация о состоянии текущего шага имеет 42 измерения. Действие последнего шага имеет 9 измерений, причем для выбранного действия задано значение 1, а для других действий - 0. В общем, в представление состояния в нашей модели встроено 93 измерения

## B. Action Definition

In StarCraft micromanagement scenarios, the original action space is very large. At each time step, each unit can move to arbitrary directions with arbitrary distances in the map. When the unit decides to attack, it can choose any enemy units in the weapon’s fire range. In order to simplify the action space, we choose 8 move directions with a fixed distance and attacking the weakest as the available actions for each unit.

В сценариях микроуправления StarCraft исходное пространство действий очень велико. На каждом временном шаге каждый юнит может перемещаться в произвольных направлениях с произвольными расстояниями на карте. Когда юнит решает атаковать, он может выбирать любого вражеского юнита в радиусе действия оружия. Чтобы упростить пространство действий, мы выбираем 8 направлений движения с фиксированным расстоянием и выбираем действие «атакуем самых слабых» в качестве доступного действия для каждого юнита.

When the chosen action is move, our units will turn to one of the 8 directions, Up, Down, Left, Right, Upper-left, Upper-right, Lower-left, Lower-right, and move a fixed distance. When the chosen action is attack, our units will stay at the current position and focus fire on enemy units. Currently, we select the enemy with the lowest hitpoint in our weapon’s attack range as the target. According to the experimental results, these actions are enough to control our units in the game.

Когда выбранное действие - движение, наши юниты повернутся в одно из 8 направлений: вверх, вниз, влево, вправо, влево вверх, право вверх, влево вниз, право вниз и переместятся на фиксированное расстояние. Когда выбранное действие является атакой, наши юниты останутся на текущей позиции и будут фокусировать огонь на вражеских юнитах. В настоящее время в качестве цели мы выбираем противника с самым низким количеством здоровья в радиусе атаки нашего оружия. Согласно результатам эксперимента, этих действий достаточно, чтобы контролировать наших юнитов в игре.

Fig. 4. The PS-MAGDS reinforcement learning diagram in the StarCraft micromanagement scenarios.

Рис. 4. Диаграмма обучения с подкреплением PS-MAGDS в сценариях микроуправления StarCraft.

(12м)

## C. Network Architecture

Because our units’ experience has a limited subset of the large state space and most test states will never been explored before, it will be difficult to apply table reinforcement learning to learn an optimal policy. To solve this problem, we use a neural network parameterized by vector θ to approximate the state-action values to improve our RL model’s generalization.

Поскольку опыт наших юнитов является ограниченным подмножеством большого пространства состояний, и большинство тестовых состояний никогда не будет исследована заранее, будет трудно применить таблицу обучение с подкреплением для изучения оптимальной политики. Чтобы решить эту проблему, мы используем нейронную сеть, параметризованную вектором θ(тета), для аппроксимации значений действия состояния для улучшения обобщения нашей RL-модели.

The input of the network is the 93 dimensions tensor from the state representation. We has 100 neurons in the hidden layer, and use the rectified linear unit (ReLU) activation function for the network nonlinearity, as demonstrated by f (z) = max(0, z), (6)

Ввод сети - тензор с 93 измерениями для представления состояния. У нас есть 100 нейронов в скрытом слое, и мы используем функцию усечённую линейную функцию(ReLU) для нелинейности сети, что демонстрируется f (z) = max (0, z), (6)

(8м)

where z is the output of hidden layer. We use ReLU function rather than Sigmoid or tanh function, because ReLU function does not have the problem of gradient descent, which can guarantee the effective training of the model [59]. Different from these saturating nonlinearities functions such as Sigmoid or tanh, ReLU function is a non-saturating nonlinearity func- tion. In terms of training time with gradient descent, the non- saturating nonlinearity is much faster [60].

где z - выход скрытого слоя. Мы используем функцию ReLU, а не функцию Sigmoid или tanh, потому что функция ReLU не имеет проблемы градиентного спуска, что может гарантировать эффективное обучение модели [59]. В отличие от этих насыщающенных функций нелинейности, таких как сигмоид или тан, функция ReLU является ненасыщенной нелинейной функцией. С точки зрения времени обучения с градиентным спуском ненасыщенная нелинейная намного быстрее [60].

The output layer of the neural network has 9 neurons, giving the probabilities of moving to 8 directions and attack. The learning model of one unit in StarCraft micromanagement scenarios, including state representation, neural network architecture and output actions, is depicted in Fig. 3.

Выходной слой нейронной сети имеет 9 нейронов, что дает вероятность движения в 8 направлениях и атаки. Модель обучения одного юнита в сценариях микроуправления StarCraft, включает представление состояния, архитектуру нейронной сети и выходные действия, изображена на рис. 3.

# IV. LEARNING METHOD FOR MICROMANAGEMENT

In this paper, we formulate StarCraft micromanagement as a multi-agent reinforcement learning model. We propose a parameter sharing multi-agent gradient-descent Sarsa(λ) (PS-MAGDS) method to train the model, and design a reward function as intrinsic motivations to promote the learning process. The whole PS-MAGDS reinforcement learning diagram is depicted in Fig. 4.

В этой статье мы формулируем микроуправление StarCraft как многоагентную модель обучения с подкреплением. Мы предлагаем метод многоагентного градиентно-спуска Сарса (λ) (PS-MAGDS) с разделением параметров для обучения модели, и конструируем функцию вознаграждения в качестве внутренней мотивации для продвижения процесса обучения. Вся схема PS-MAGDS обучения с подкреплением изображена на рис. 4.

## A. Parameter Sharing Multi-agent Gradient-Descent Sarsa(λ)

We propose a multi-agent RL algorithm that extend the traditional Sarsa(λ) to multiple units by sharing the parameters of the policy network among our units. To accelerate the learning process and tackle the problem of delayed rewards, we use eligibility traces in reinforcement learning. As a basic mechanism in RL, eligibility traces are used to assign temporal credit, which consider a set of previously experienced transitions [61].

Мы предлагаем RL-алгоритм с несколькими агентами, который расширяет традиционный Сарса алгоритм (λ) для управления нескольких юнитов, разделяя параметры сети политики между нашими юнитами. Чтобы ускорить процесс обучения и решить проблему отсроченных вознаграждений, мы используем приемлемые **протоколы** **возможного выбора** черты(eligibility traces) в обучении с подкреплением. В качестве основного механизма в RL, приемлемые **протоколы** **возможного выбора** используются для назначения временного вознаграждения, которые учитывает набор ранее испытанных переходов [61].

This means it not only considers the value of the last state-action pair but also the visited ones. With this method, we can solve the problem of delayed reward in the game environment. Sarsa with eligibility traces, termed as Sarsa(λ), is one way of averaging backups made after multiple steps. λ is a factor that determines the weight of each backup.

Это означает, что он учитывает не только значение последней пары состояние-действие, но и посещенные. С помощью этого метода мы можем решить проблему отсроченного вознаграждения в игровой среде. Сарса с приемлемыми чертами, называемая Сарса (λ лямбда), является одним из способов усреднения резервных копий(backups), сделанных после нескольких шагов. λ является фактором, который определяет вес каждой резервной копии.

In our implementation of Sarsa(λ) for multiple units combat, we use a neural network as the function approximator and share network parameters among all our units. Although we have only one network to train, the units can still behave differently because each one receives different observations and actions as its input.

В нашей реализации Sarsa (λ) для боя нескольких юнитов, мы используем нейронную сеть в качестве аппроксиматора функции и разделяем параметры сети между всеми нашими юнитами. Хотя у нас есть только одна сеть для обучения, подразделения могут вести себя по-разному, потому что каждый юнит получает разные наблюдения и действия в качестве входных данных.

To update the policy network efficiently, we use the gradient-descent method to train the Sarsa(λ) reinforcement learning model. The gradient-descent learning update is demonstrated in equation (7),

Для эффективного обновления сети политик мы используем метод градиентного спуска для обучения модели обучения с подкреплением Сарса (λ). Обновление модели с градиентным спуском продемонстрировано в уравнении (7),

δt = rt+1 + γQ(st+1, at+1; θt) − Q(st, at; θt) (7a)

θt+1 = θt + αδtet (7b)

et = γλet−1 + ∇θtQ(st, at; θt), e0 = 0 (7c)

where **et** is the eligibility traces at time step t.

Где **et** приемлемые **протоколы** **возможного выбора** на шаге t.

One of the challenging issues in reinforcement learning is the trade-off between exploration and exploitation. If we choose the best action every step according to current policy, we are likely to trap in local optimum. On the contrary, if we tend to explore in the large state space, the model will have difficulty in converging. In the experiment, we use the #-greedy method to choose actions during training, which selects the current best action with probability 1 − #, and takes a random exploratory action with probability #,

Одним из сложных вопросов в обучении с подкреплением является компромисс между разведкой и эксплуатацией. Если мы выберем лучшее действие на каждом этапе в соответствии с текущей политикой, мы, вероятно, попадем в локальный оптимум. Напротив, если мы склонны исследовать в большом пространстве состояний, модели будет трудно сходиться. В эксперименте мы используем метод e-жадный для выбора действий во время тренировки, который выбирает текущее наилучшее действие с вероятностью 1 - e и выполняет случайное исследовательское действие с вероятностью e,

(8)

where N equals to 9 in the experiment.

Где N равно 9 в эксперименте

We use exponentially e decay to implement the e -greedy method. The e is initialized to 0.5 and anneals schedule with an exponential smoothing window of the episode number episode num, as demonstrated by

Мы используем экспоненциально e распад для реализации метода e-жадный. e инициализируется со значением 0,5 и **нормалует** график закалки/отжига с показателем экспоненциального сглаживания окна в эпизоде с номером эпизода *episode\_num*, как показано

(9)

The overall parameter sharing multi-agent gradient-descent Sarsa(λ) method is presented in Algorithm 1.

Целиком метод разделения общих параметров мультиагентного градиентного спуска Сарса (λ) представлен в алгоритме 1.

Algorithm 1 Parameter Sharing Multi-Agent GradientDescent Sarsa(λ)

1: Initialize policy parameters θ shared among our units

2: Repeat (for each episode):

3: e0 = 0

4: Initialize st, at

5: Repeat (for each step of episode):

6: Repeat (for each unit):

7: Take action at, receive rt+1, next state st+1

8: Choose at+1 from st+1 using -greedy

9: If random(0, 1) <

10: at+1 = randint(N)

11: else

12: at+1 = arg maxa Q(st+1, a; θt)

13: Repeat (for each unit):

14: Update TD error, weights and eligibility traces

15: δt = rt+1 + γQ(st+1, at+1; θt) − Q(st, at; θt)

16: θt+1 = θt + αδtet

17: et+1 = γλet + ∇θt+1Q(st+1, at+1; θt+1)

18: t ← t + 1

19: until st is terminal

(1ч)

## B. Reward Function

The reward function provides useful feedbacks for RL agents, which has a significant impact on the learning results [62]. The goal of StarCraft micromanagement is to destroy all of the enemy units in the combat. If the reward is only based on the final result, the reward function will be extremely sparse. Moreover, units usually get a positive reward after many steps. The delay in rewards makes it difficult to learn which set of actions is responsible for the corresponding

rewards.

Функция вознаграждения обеспечивает полезную обратную связь для RL-агентов, которая оказывает значительное влияние на результаты обучения [62]. Целью микроуправления StarCraft является уничтожение всех вражеских .юнитов в бою. Если вознаграждение основано только на конечном результате, функция вознаграждения будет крайне скудной. Более того, юниты обычно получают положительное вознаграждение после множества шагов. Отсрачивание вознаграждения затрудняет выяснение того, какой набор действий отвечает за соответствующие вознаграждения.

To tackle the problem of sparse and delayed rewards in micromanagement, we design a reward function to include small intermediate rewards. In our experiment, all agents receive the main reward caused by their attack action at each time step, equalling to the damage that the enemy units received minus the hitpoint loss of our units.

Чтобы решить проблему редких и отсроченных вознаграждений в микроуправлении, мы разработали функцию вознаграждения, включающую небольшие промежуточные вознаграждения. В нашем эксперименте все агенты получают основное вознаграждение, получаемое в случае атаки на каждом временном шаге, равное урону, который получили вражеские юниты минус количество потерянного здоровья наших юнитов.

*rt = (damage\_amountt × damage\_factor − ρ×(unit hitpointt−1 − unit hitpointt))/10* (10)

where damage amount is the amount of damage caused by our units’ attack, *damage\_factor* is our units’ attack force and unit hitpoint is our unit’s hitpoint. We divide the reward by a constant to resize it to a more suitable range, which is set to 10 in our experiment. ρ is a normalized factor to balance the total hitpoint of our units and enemy units,

где *damage\_amount* - это количество урона, нанесенного атакой наших юнитов, *damage\_factor* - сила атаки наших юнитов, а количество *хитпоинтов* юнита — это количество очков здоровья нашего юнита. Мы делим вознаграждение на константу, чтобы изменить коэффициент до более подходящего диапазона, который в нашем эксперименте установлен на 10. ρ является нормализованным фактором для баланса общего количества очков здоровья наших юнитов и юнитов противника,



where H is the number of enemy units, and N is the numbe of our units. Generally speaking, this normalized factor is necessary in StarCraft micromanagement with different numbers and types of units. Without proper normalization, polic network will have difficulty in converging, and our units nee much more episodes to learn useful behaviors

где H - количество вражеских юнитов, а N - количество наших юнитов. Вообще говоря, этот нормализованный фактор необходим в микроуправлении StarCraft с различным количеством и типом юнитов. Без надлежащей нормализации, политика сети будет хуже сходиться, и нашим юнитам нужно гораздо больше времени для изучения полезного поведения

Apart from the basic attack reward, we consider some extra rewards as the intrinsic motivation to speed up the training process. When a unit is destroyed, we introduce an extra negative reward, and set it to -10 in our experiment. We would like to punish this behavior in consideration that the decrease of own units has a bad influence on the combat result. Besides in order to encourage our units to work as a team and make cooperative actions, we introduce a reward for units’ move. If there are no our units or enemy units in the move direction we give this move action a small negative reward, which i set to -0.5. According to the experiment, this reward has a impressive effect on the learning performance, as shown i Fig. 6.

Помимо основной награды за атаку, мы рассматриваем некоторые дополнительные награды как внутреннюю мотивацию для ускорения тренировочного процесса. Когда юнит уничтожается, мы вводим дополнительное отрицательное вознаграждение и присваеваем значение -10 в нашем эксперименте. Мы хотели бы наказать такое поведение, потому-что уменьшение количества собственных юнитов плохо влияет на результат боя. Кроме того, чтобы побудить наших юнитов работать в команде и предпринять совместные действия, мы вводим вознаграждение за их перемещение. Если в направлении движения нет наших юнитов или вражеских юнитов, мы даем этому движению небольшое отрицательное вознаграждение, которое равно -0,5. Согласно эксперименту, это вознаграждение оказывает впечатляющее влияние на эффективность обучения, как показано на рис. 6.

## C. Frame Skip

When applying reinforcement learning to video games, we should pay attention to the continuity of actions. Because of the real-time property of StarCraft micromanagement, it is impractical to make a action every game frame. One feasible method is using frame skip technology, which executes a training step every fixed number of frames. However, small frame skip will introduce strong correlation in the training data, while large frame skip will reduce effective training samples. We refer to related work in [20], and try several frame skips (8, 10, 12) in a small scale micromanagement scenario. At last, we set the frame skip to 10 in our experiment, which takes an action every 10 frames for each unit.

Применяя обучающее подкрепление к видеоиграм, мы должны обратить внимание на непрерывность действий. Так как микроуправление в StarCraft происходит в реальном то нецелесообразно совершать действия в каждом игровом кадре. Одним из возможных способов является использование технологии пропуска кадров, которая выполняет этап обучения для каждого фиксированного числа кадров. Тем не менее, небольшой пропуск кадров привнесет сильную корреляцию в обучающие данные, в то время как большой пропуск кадров уменьшит эффективные обучающие выборки. Мы ссылаемся на соответствующую работу в [20] и пробуем несколько пропусков кадров (8, 10, 12) в небольшом сценарии микроуправления. Наконец, мы установили пропуск кадра равным 10 в нашем эксперименте, который выполняет действие каждые 10 кадров для каждого юнита.

# V. EXPERIMENT SETTINGS

## A. StarCraft Micromanagement Scenarios

We consider several StarCraft micromanagement scenarios with various units, including Goliaths vs. Zealots, Goliaths vs. Zerglings and Marines vs. Zerglings, as shown in Fig. 5.

Мы рассматриваем несколько сценариев микроуправления StarCraft с различными юнитами, включая Голиафов против Зилотов, Голиафов против Зерглингов и Морпехов против Зерглингов, как показано на Рис. 5.

1) In the first scenario, we will control 3 Goliaths to fight against 6 Zealots. From Table II, we can see that the enemy units have advantage on the number of units, hitpoint and damage factor. By contrast, our units’ fire range is much wider.

1) В первом сценарии мы будем управлять тремя Голиафами, чтобы сражаться с 6 Зилотами. Из таблицы II видно, что вражеские юниты имеют преимущество по количеству юнитов, здоровья и коэффициенту урона. В отличие от них, радиус атаки наших юнитов намного шире.

2) In the second scenario, the enemies have 20 Zerglings. Our Goliaths units have advantage on hitpoint, damage factor and fire range, while the enemies have much more units and less cooldown time.

2) Во втором сценарии у врагов есть 20 зерглингов. Наши Голиафы имеют преимущество по количеству здоровья, коэффициенту урона и дальности огня, в то время как у врагов гораздо больше юнитов и меньше времени перезарядки оружия(то есть выше скорость атаки).

3) In the third scenario, we will control up to 20 Marines to fight against 30 Zerglings. The enemy units have advantage on speed and amount, while our units have advantage on fire range and damage factor.

3) В третьем сценарии мы будем управлять 20-ю морпехами для борьбы с 30 зерглингами. Вражеские юниты имеют преимущество в скорости и количестве, в то время как наши юниты имеют преимущество в дальности огня и коэффициенте повреждения.

We divide these scenarios into two groups. The first and the second are small scale micromanagements and the last is the large scale micromanagement. In these scenarios, the enemy units are controlled by the built-in AI, which is hard-coded with game inputs. An episode terminates when either side of the units are destroyed. A human beginner of StarCraft can’t beat the built-in AI in these scenarios. Platinum-level players have average win rates of below 50% with 100 games for each scenario. Our RL agents is expected to exploit their advantages and avoid their disadvantages to win these combats.

Мы делим эти сценарии на две группы. Первый и второй являются мелкомасштабными микроуправлениями, а последний - крупномасштабным микроуправлениям. В этих сценариях вражеские юниты управляются встроенным ИИ, который жестко запрограммирован в игре. Эпизод заканчивается, когда любая из сторон потеряла всех своих юнитов. Человек-новичок в StarCraft не может победить встроенный ИИ в этих сценариях. Игроки с платиновым уровнем имеют средний процент выигрыша ниже 50% из 100 игр для каждого сценария. Ожидается, что наши RL-агенты будут использовать их преимущества и избегать их недостатков, чтобы выиграть эти бои.

## B. Training

In the training process, we set the discount factor γ to 0.9, the learning rate α to 0.001, and the eligibility traces factor λ to 0.8 in all scenarios. Moreover, the maximum steps of each episode are set to 1000. In order to accelerate the learning process, the game runs at full speed by setting gameSpeed to 0 in BWAPI. The experiment is applied on a computer with an Intel i7-6700 CPU and 16GB of memory.

В процессе обучения мы устанавливаем коэффициент дисконтирования γ равным 0,9, коэффициент обучения α равным 0,001, а коэффициент приемлимых черт λ - 0,8 во всех сценариях. Кроме того, максимальное количество шагов в каждом эпизоде равен 1000. Чтобы ускорить процесс обучения, игра запускается на полной скорости, установив gameSpeed в 0 в BWAPI. Эксперимент проводится на компьютере с процессором Intel i7-6700 и 16 ГБ памяти.

(1ч)

# VI. RESULTS AND DISCUSSIONS

In this section, we analyze the results in different microman- agement scenarios and discuss our RL model’s performance. In small scale scenarios, we use the first scenario as a starting point to train our units. In the remaining scenarios, we introduce transfer learning method to scale the combat to large scenarios. The object of StarCraft micromanagement is defeating the enemies and increasing the win rates in these given scenarios. For a better comprehension, we analyze the win rates, episode steps and average rewards during training, as well as the learned strategies. Our code and results are open-source at <https://github.com/nanxintin/StarCraft-AI>.

В этом разделе мы анализируем результаты различных сценариев микроуправления и обсуждаем производительность нашей RL-модели. В небольших сценариях мы используем первый сценарий в качестве отправной точки для обучения наших юнитов. В оставшихся сценариях мы используем трасферный метод обучения, чтобы масштабировать бои до больших сценариев. Цель микроуправления StarCraft - победить врагов и увеличить выигрыш в данных сценариях. Для лучшего понимания мы анализируем коэффициенты выигрыша, количество этапов и среднее вознаграждение во время тренировки, а также изученные стратегии. Наш код и результаты с открытым исходным кодом на https://github.com/nanxintin/StarCraft-AI.

## A. Small Scale Micromanagement

In small scale micromanagement scenarios, we will train Goliaths against enemy units with different amounts and types. In the second scenario, we will also use transfer learning method to train Goliaths based on the well-trained model of the first scenario. Both of the two scenarios are trained with 4000 episodes and over 1 million steps.

В мелкомасштабных сценариях микроуправления мы будем тренировать Голиафов на вражеских юнитах различающихся по типу и количеств. Во втором сценарии мы также будем использовать метод трансферного обучения для обучения Голиафов на основе хорошо обученной модели первого сценария. Оба сценария обучены на 4000 эпизодах и более 1 миллиона шагов.

1) Goliaths vs. Zealots: In this scenario, we train our Goliaths units from scratch and analyze the results.

1) Голиафы против зилотов: в этом сценарии мы обучаем отряд наших Голиафов с нуля и анализируем результаты.

* Win Rates: At first, we will analyze the learning perfor- mance of our RL method with moveReward. To evaluate the win rates, we test our model after every 200 episodes’ training for 100 combats, and depict the results in Fig. 6. We can see that our Goliaths units can’t win any combats before 1400 episodes. With the progress of training, units start to win several games and the curve of win rates has an impressive increase after 2000 episodes. After 3000 episodes’ training, our units can reach win rates of 100% at last.
* Процент побед: Сначала мы проанализируем эффективность обучения нашего метода RL с помощью moveReward. Чтобы оценить количество побед, мы тестируем нашу модель после каждых 200 эпизодов тренировок для 100 боев и показываем результаты на рис. 6. Мы видим, что наши отряды Голиафа не могут выиграть ни одного боя до 1400 эпизодов. С прогрессом обучения юниты начинают выигрывать в нескольких играх, а кривая показателей выигрыша впечатляет после 2000 эпизодов. После 3000 тренировок наши юниты могут наконец достичь 100% выигрыша.
* Episode Steps: We depict the average episode steps and standard deviations of our three Goliaths units during training in Fig. 7. It is apparent to see that the curve of average episode steps has four stages. In the opening, episode steps are extremely few because Goliaths have learned nothing and are destroyed quickly. After that, Goliaths start to realize that the hitpoint damage causes a negative reward. They learn to run away from enemies and the episode steps increase to a high level. And then, episode steps start to decrease because Goliaths learn to attack to get positive rewards, rather than just running away. In the end, Goliaths have learned an appropriate policy to balance move and attack, and they are able to destroy enemies in almost 300 steps.
* Количество шагов в эпизоде(тренировке): мы показываем средние количество шагов в эпизоде и стандартные отклонения наших трех Голиафов во время обучения на Рис. 7. Очевидно, что кривая средних шагов эпизода имеет четыре стадии. В начале, шагов в эпизодах очень мало, потому что Голиафы ничему не научились и быстро уничтожаются. После этого Голиафы начинают понимать, что урон здоровью вызывает отрицательное вознаграждение. Они учатся убегать от врагов, и количество шагов в эпизоде увеличиваются до высокого уровня. И затем, шаги эпизода начинают уменьшаться, потому что Голиафы учатся атаковать, чтобы получить положительные награды, а не просто убегать. В конце, Голиафы выучили соответствующую политику, чтобы уравновесить маневрённость и атаку, и они способны уничтожать врагов почти за 300 шагов.
* Average Rewards: Generally speaking, a powerful game AI in micromanagement scenarios should defeat the ene- mies as soon as possible. Here we introduce the average rewards, dividing the total rewards by episode steps in the combat. The curve of our Goliaths units’ average rewards is depicted in Fig. 8. The average rewards have an obvious increase in the opening, grow steadily during training and stay smooth after almost 3000 episodes.
* Среднее вознаграждение: Вообще говоря, мощный игровой ИИ в сценариях микроуправления должен побеждать врагов как можно скорее. Здесь мы вводим среднее вознаграждение, деля общее вознаграждение на количество шагов в бою. Кривая среднего вознаграждения наших Голиафов изображена на рис. 8. Средние вознаграждения имеют сильно возрастают в начале, и неуклонно растёт во время тренировок и остаются плавными после почти 3000 эпизодов.

2) Goliaths vs. Zerglings: In this scenario, the enemy units are a group of Zerglings, and we reuse the well-trained model from the first scenario to initialize the policy network. In comparison with learning from scratch, we have a better understanding of transfer learning.

2) Голиафы против зерглингов: в этом сценарии вражеские отряды состоят из группы зерглингов, и мы повторно используем хорошо обученную модель из первого сценария для инициализации сети политик. По сравнению с обучением с нуля, у нас есть лучшее понимание трансферного обучения.

Win Rates: We draw the win rates in Fig. 9. When training from scratch, the learning process is extremely slow and our units can’t win a game until 1800 episodes. Without transfer learning, the win rates are below 60% after 4000 episodes. When training based on the model of the first scenario, the learning process is much faster. Even in the opening, our units win several games, and the win rates reach 100% in the end.

Коэффициенты выигрыша. На рис. 9 показаны коэффициенты выигрыша. При обучении с нуля процесс обучения очень медленный, и наши юниты не могут выиграть игру до 1800 эпизодов. Без трансферного обучения процент выигрышей ниже 60% после 4000 эпизодов. Когда обучение основано на модели первого сценария, процесс обучения происходит намного быстрее. Даже в дебюте наши юниты выигрывают несколько игр, и в итоге процент выигрышей достигает 100%.

(35m)

Episode Steps: In Fig. 10, we draw the average episode steps of our three Goliaths during training. Without transfer learning, the curve has the similar trend with that in the first scenario. The average episode steps have a obvious increase in the opening and drop gradually during training. When training with transfer learning, the average episode steps keep stable in the whole training process, within the range of 200 to 400. A possible explanation is that our units have learned some basic move and attack skills from the well-trained model, and they use these skills to speed up the training process.

Шаги эпизода: На рис. 10 мы показываем средние количество шагов эпизода для наших трех Голиафов во время тренировки. Без трансферного обучения кривая имеет сходную тенденцию с таковой в первом сценарии. Средние количество шагов эпизода имеет очевидное увеличение открытия и постепенно синжается во время тренировки. При обучении с трансферным обучением к среднее количество шагов эпизода остаётся стабильными в течение всего тренировочного процесса, в диапазоне от 200 до 400. Возможное объяснение состоит в том, что наши юниты изучили некоторые базовые навыки движения и атаки из хорошо обученной модели, и они используюте эти навыки, чтобы ускорить процесс обучения.

Average Rewards: We draw the average rewards of our three Goliaths in Fig. 11. When training from scratch, our units have difficulty in winning the combat in the opening and the average rewards are in a low level before 1000 episodes. The average rewards with transfer learning, by comparison, are much higher from the beginning and behave better in the whole training process.

Средняя награда. На рис. 11. мы показываем среднюю награду для наших трех Голиафов. При обучении с нуля наши юниты испытывают трудности в победе в бою в дебюте, а средние награды находятся на низком уровне до 1000 эпизодов. Средние вознаграждения с трансферным обучением, для сравнения, намного выше с самого начала и ведут себя лучше во всем процессе обучения.

## B. Large Scale Micromanagement

In large scale micromanagement scenarios, we use curricu- lum transfer learning to train our Marines to play against Zer- glings, and compare the results with some baseline methods.

В крупномасштабных сценариях микроуправления мы используем постепенное трансферное обучение, чтобы обучить наших морпехов играть против зерглингов и сравнить результаты с некоторыми базовыми методами.

1) Marines vs. Zerglings: In this section, we design a curriculum with 3 classes to train the units, as shown in Table III. After training, we test the performance in two target scenarios: M10 vs. Z13 and M20 vs. Z30. In addition, we use some baseline methods as a comparison, which consist of rule-based approaches and DRL approaches.

1) Морпехи против зерглингов: в этом разделе мы разрабатываем учебную программу с 3 классами для обучения юнитов, как показано в таблице III. После обучения мы тестируем производительность в двух целевых сценариях: M10 против Z13 и M20 против Z30. Кроме того, мы используем некоторые базовые методы для сравнения, которые состоят из подходов основанных на правилах и подходов DRL.

• Weakest: A rule-based method, attacking weakest in the fire range.

• Слабейшие: метод, основанный на правиле, атаковать слабейших в радиусе действия оружия.

• Closest: A rule-based method, attacking closest in the fire range.

• Ближайший: метод, основанный на правиле, атакуовать ближайшего в радиусе действия оружия.

• GMEZO: A DRL method, based on the zero-order op- timization, having impressive results over traditional RL methods [20].

• GMEZO: метод DRL, основанный на оптимизации нулевого порядка, имеющий впечатляющие результаты по сравнению с традиционными RL-методами [20].

• BicNet: A DRL method, based on the actor-critic archi- tecture, having the best performance in most StarCraftmicromanagement scenarios [21].

• BicNet: метод DRL, основанный на архитектуре актер-критика, имеющий наилучшую производительность в большинстве сценариев микроуправления StarCraft [21].

In Table IV, we present the win rates of the PS-MAGDS method and baseline methods. In each scenario, we measure our model’s average win rates in 100 test games for 5 times. In M10 vs. Z13, PS-MAGDS achieves a win rate of 97%, which is much higher than other methods, including the recently proposed GMEZO and BicNet. In M20 vs. Z30, PS-MAGDS has the second best performance, which is very close to the best one.

В таблице IV мы представляем процент выигрышей по методу PS-MAGDS и базовым методам. В каждом сценарии мы 5 раз измеряем средние показатели выигрышей в нашей модели в 100 тестовых играх. В M10 против Z13 PS-MAGDS достигает выигрыша 97%, что намного выше, чем другие методы, включая недавно предложенные GMEZO и BicNet. В M20 против Z30, PS-MAGDS имеет вторую лучшую производительность, которая очень близка к лучшей.

(39м)

We also test our well-trained models in curricular scenarios and unseen scenarios, and present the results in Table V. We can see that PS-MAGDS has outstanding performances in these curricular scenarios. In unseen scenarios with more units, PS-MAGDS also has acceptable results.

Мы также проверяем наши хорошо обученные модели в учебных сценариях и не исследуемых ранее сценариях, и представляем результаты в Таблице V. Мы видим, что у PS-MAGDS есть выдающиеся производительность в этих учебных сценариях. В не исследованных ранее сценариях с большим количеством единиц у PS-MAGDS также есть приемлемые результаты.

## C. Strategies Analysis

In StarCraft micromanagement, there are different types of units with different skills and properties. Players need to learn how to move and attack with a group of units in real time. If we design a rule-based AI to solve this problem, we have to consider a large amount of conditions, and agent’s ability is also limited.

В микроуправлении StarCraft есть различные типы юнитов с различными навыками и свойствами. Игроки должны изучить, как двигаются и атакуют группы юнитов в режиме реального времени. Если мы проектируем основанный на правилах AI, чтобы решить эту проблему, мы должны рассмотреть большое количество условий, и способность агентов также ограничена.

Beginners of StarCraft could not win any of these combats presented in our paper. So these behaviors are highly complex and difficult to learn. With reinforcement learning and curriculum transfer learning, our units are able to master several useful strategies in these scenarios. In this section, we will make a brief analysis on these strategies that our units have learned.

Новички StarCraft не могут выиграть ни один из этих боев, представленных в нашей статье. Таким образом, это поведение очень сложное и трудно в усвоениии. При обучение с подкреплением и постепенным обучением, наши юниты в состоянии справиться с несколькими полезными стратегиями в этих сценариях. В этом разделе мы сделаем краткий анализ этих стратегий, которые изучили наши юниты.

1) Disperse Enemies: In small scale micromanagement scenarios, our Goliaths units have to fight against the opponent with a larger amount and more total hitpoints. If our units stay together and fight against a group of units face-to-face, they will be destroyed quickly and lose the combat. The appropriate strategy is dispersing enemies, and destroying them one by one.

1) Разделение врагов: В мелкомасштабных сценариях микроуправления наши Голиафы должны бороться против противника с большим количеством противников и большим количеством общего количества здоровья. Если наши отделения останутся вместе и будут бороться с противником «лицом к лицу», они будут быстро уничтожены и проиграют бой. Правильная стратегия в разделении врагов и уничтожении их по одному.

In the first scenario, our Goliaths units have learned dispersing Zealots after training. In the opening, our units disperse enemies into several parts and destroy it in one part first. After that, the winning Goliath moves to other Goliaths and helps to fight against the enemies. Finally, our units focus fire on the remaining enemies and destroy them. For a better understanding, we choose some frames of game replay in the combat and draw units’ move and attack directions in Fig. 12. The white lines stand for the move directions and the red lines stand for the attack directions.

В первом сценарии наши отделения Голиафов изучили рассеивающихся зилотов после обучения. В открытии наши отделения рассеивают врагов в несколько частей и разрушают его в одной части сначала. После этого победа Голиаф двигается к другим Голиафам и помогает бороться против врагов. Наконец, наш центр единиц стреляют в остающихся врагов и разрушают их. Для лучшего понимания мы выбираем некоторые структуры переигровки игры в бою и привлекаем направления движения и нападения единиц на Рис. 12. Белые линии обозначают направления движения, и красные линии обозначают направления нападения.

The similar strategy occurs in the second scenario. The opponent has much more units, and Zerglings Rush has great damage power, which is frequently used in StarCraft games. Our Goliaths units disperse Zerglings into several groups and keep a suitable distance with them. When units’ weapons are in a valid cooldown state, they stop moving and attack the enemies, as shown in Fig. 13.

Подобная стратегия происходит во втором сценарии. У противника есть намного больше единиц, и ранняя атака Зерглингами наносит большой ущерб противнику, она часто используется в StarCraft. Наши Голиафы разделяют Зерглингов на несколько групп и сохраняют подходящую дистанцию. Когда оружие юнита находится в состоянии готовым к стрельбе, они прекращают двигаться и нападают на врагов, как показано на Рис. 13.

2) Keep the Team: In large scale micromanagement sce- narios, each side has a mass of units. Marines are small size ground units with low hitpoints. If they combat in several small groups, they are unable to resist the enemies. A suitable strategy is keeping our Marines in a team, moving with the same direction and attacking the same target, as demonstrated in Fig. 14. From these figures, we can see that our Marines have learned to move forward and retreat in a queue.

2) Сохранение команды. В масштабных сценариях микроуправления каждая сторона имеет массу юнитов. Морпехи - это небольшие наземные подразделения с низкими количеством здоровья. Если они сражаются в нескольких небольших группах, они не могут противостоять врагам. Подходящей стратегией является удержание наших морпехов в команде, движение в том же направлении и атака на одну и ту же цель, как показано на рис. 14. Из этих рисунков мы видим, что наши морпехи научились двигаться вперед и отступать по очереди.

3) Hit and Run: Apart from the global strategies discussed above, our units have also learned some local strategies during training. Among them, Hit and Run is the most widely used tactic in StarCraft micromanagement. Our units rapidly learn the Hit and Run tactic in all scenarios, including the single unit’s Hit and Run in Fig. 12 and Fig. 13, and a group of units’ Hit and Run in Fig. 14.

3) Бей и беги: кроме глобальных стратегий, о которых говорилось выше, наши юниты также изучили некоторые локальные стратегии во время обучения. Среди них «Бей и беги» - наиболее широко используемая тактика в микроуправлении StarCraft. Наши юниты быстро изучают тактику «Беги и беги» во всех сценариях, включая «Бей и беги» одного юнита на рис. 12 и рис. 13, а также группу юнитов использующих тактику «Бей и беги»на рис. 14

4) Existing Problems: Although our units have learned useful strategies after training, there are still some problems in combats. For instance, Goliaths move forward and backward now and then and don’t join in combats to help other units in time. In addition, units prefer moving to the boundary of the map, so as to avoid the enemies.

4) Существующие проблемы: Хотя наши подразделения выучили полезные стратегии после обучения, в сражениях все еще есть некоторые проблемы. Например, Голиафы время от времени двигаются вперед и назад и не участвуют в битвах, чтобы вовремя помогать другим юнитам. Кроме того, юниты предпочитают перемещаться к границе карты, чтобы избежать врагов.

# VII. CONCLUSION AND FUTURE WORK

This paper focuses on the multiple units control in StarCraft micromanagement scenarios. We present several contributions, including an efficient state representation, the parameter shar- ing multi-agent gradient-descent Sarsa(λ), the effective reward function and the curriculum transfer learning method used to extend our model to various scenarios.

Эта статья посвящена управлению несколькими юнитами в сценариях микроуправления StarCraft. Мы представляем несколько вкладов, в том числе эффективное представление состояния, совместное использование параметров многоагентного градиентного спуска Сарса (λ), эффективную функцию вознаграждения и метод трансферного постепенного обучения, используемый для расширения нашей модели в различных сценариях.

We demonstrate the effectiveness of our approach in both small scale and large scale scenarios, and the superior performance over some baseline methods in two target scenarios. It is remarkable that our proposed method is able to learn appropriate strategies and defeat the built-in AI in various scenarios. In addition, there are still some areas for future work.

Мы демонстрируем эффективность нашего подхода как в мелкомасштабных, так и в крупномасштабных сценариях, а также превосходную производительность по сравнению с некоторыми базовыми методами в двух целевых сценариях. Примечательно, что предлагаемый нами метод способен выучить соответствующие стратегии и победить встроенный ИИ в различных сценариях. Кроме того, еще есть области для будущей работы.

The cooperative behaviors of multiple units are learned by sharing the policy network, constructing an efficient state representation method including other units’ information and the proposed intrinsic motivated reward function. Although our units can successfully master some effective coordina- tion strategies, we will explore more intelligent methods for multi-agent collaboration.

Совместное поведение нескольких юнитов изучается путем совместного использования сети политик, построения эффективного метода представления состояния, включающего в себя информацию других юнитов и предлагаемую встроенную функцию мотивированного вознаграждения. Хотя наши подразделения могут успешно освоить некоторые эффективные стратегии координации, мы будем исследовать более интеллектуальные методы для совместной работы нескольких агентов.

To solve the delayed reward problem in StarCraft micromanagement, we use a simple, straight and efficient reward shaping method. Nevertheless, there are also some other methods solving the sparse and delayed rewards, such as hierarchical reinforcement learning. Hierarchical RL integrates hierarchical action-value functions, operating at different temporal scales [63]. Compared with the reward shaping method, hierarchical RL has the capacity to learn temporally-abstracted exploration, and gives agents more flexibility.

Чтобы решить проблему отложенного вознаграждения в микроуправлении StarCraft, мы используем простой, прямой и эффективный метод формирования вознаграждения. Тем не менее, есть также некоторые другие методы решения редких и отсроченных вознаграждений, такие как иерархическое обучение с подкреплением. Иерархический RL объединяет иерархические функции действия-значения, работающие в разных временных масштабах [63]. По сравнению с методом формирования вознаграждения, иерархический RL обладает способностью изучать абстрагированное от времени исследование, и дает агентам большую гибкость.

But its framework is also much more complicated, and automatically subgoals extraction is still an open problem. In the future, we will make an in-depth study on applying hierarchical RL to StarCraft. At present, we can only train ranged ground units with the same type, while training melee ground units using RL methods is still an open problem. We will improve our method for more types of units and more complex scenarios in the future. Finally, we will also consider to use our micromanagement model in the StarCraft bot to play full the game.

Но его структура также намного сложнее, и автоматическое извлечение подзадач остается открытой проблемой. В будущем мы проведем углубленное исследование по применению иерархической RL к StarCraft. В настоящее время мы можем обучать только наземных юнитов дальнего боя того же типа, в то время как тренировка наземных юнитов ближнего боя с использованием методов RL остается открытой проблемой. Мы улучшим наш метод для большего количества типов юнитов и более сложных сценариев в будущем. Наконец, мы также рассмотрим возможность использования нашей модели микроуправления для StarCraft бота в полноценной игре.

(1ч)

# ACKNOWLEDGMENT

We would like to thank Qichao Zhang, Yaran Chen, Dong Li, Zhentao Tang and Nannan Li for the helpful comments and discussions about this work and paper writing. We also thank the BWAPI and StarCraft group for their meaningful work.