Оглавление

[1. Введение 3](#_Toc19449905)

[1.1 Недостатки существующих решений и предлагаемые решения 3](#_Toc19449906)

[1.2 Постановка задачи 4](#_Toc19449907)

[1.3 Научная новизна 4](#_Toc19449908)

[2. Аналитическая часть 6](#_Toc19449909)

[2.1 Обзор исследований 6](#_Toc19449910)

[2.1.1 Бот компании OpenAI игры Dota 2 6](#_Toc19449911)

[2.1.2 Бот компании OpenAI играющий в сумо и футбол 6](#_Toc19449912)

[2.1.3 Per-Arne Andersen, Morten Goodwin, Ole-Christoffer Granmo «Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games» 7](#_Toc19449913)

[2.1.4 Kun Shao, Yuanheng Zhu, Member, IEEE and Dongbin Zhao, Senior Member «StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning» 10](#_Toc19449914)

[2.1.5 Hendrik Baier, Peter I. Cowling «Evolutionary MCTS for Multi-Action Adversarial Games» 28](#_Toc19449915)

[2.1.6 AlphaStar: Mastering the Real-Time Strategy Game StarCraft II 41](#_Toc19449916)

[Список литературы 47](#_Toc19449917)

# Введение

Адекватный игровой искусственный интеллект имеет большое значение в современных военных стратегиях, т.к. зачастую игроку очень сложно найти другого игрока для долгой совместной игры.

Ещё большую сложность представляет разработка ИИ, способного адекватно управлять армией из разнородных юнитов, передвигающихся по графу реальных дорог, учитывая множество условий: рельеф местности, ландшафт, время суток и время года, тактико-технические характеристики техники и вооружений, снабжение по дорогам и т. д. Программирование поведения армии, способной нанести поражение игроку-человеку без численного перевеса на основе жёстко заданных алгоритмов сложная задача, кроме того к такому виду ИИ гораздо проще подобрать стратегию, с помощью которой можно всегда выигрывать, такие стратегии называются доминантными.   
 Перспективным видится использование ИНС, способных к «самообучению». Искусственная нейронная сеть(ИНС, в тексте используется сокращение нейросеть) - математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. ИНС представляет из себя набор нейронов, которые представляют из себя функции с несколькими входными переменными, которые передают полученный результат следующему нейрону или наружу. ИНС является ИИ, который представляет из себя самообучающуюся программу. Программы такого типа разрабатываются в рамках машинного обучения.

Реализация игрового интеллекта на базе такой технологии в теории позволит обучить программу не только на играх с человеком (что очень медленно), а на множестве ускоренных по времени играх с самим собой. Также при достаточно длительном обучении можно получить ИИ к которому не удастся подобрать доминантную стратегию и поэтому игроку придётся лучше думать, но это также не означает что будет создан непобедимый ИИ.

## 1.1 Недостатки существующих решений и предлагаемые решения

Рассмотренные далее, решения хорошо справляются с задачей управления на графах похожих на «шахматную доску», но они не могут работать с графом дорог(картой дорог). Графы вида «шахматная доска» похожи на графы дорог, но главное отличие в том, что они не такие плотные и разветвлённые, и в качестве вершин у них клетки вместо мест стыковок дорог.

## 1.2 Постановка задачи

Целью данной работы является разработка нейронной сети, которая будет управлять различными видами войск на дорожном графе в игре «WarOnMap», ранее созданной студентами ПГТУ.

Цель нейросети: победить игрока (человека или ИИ). Для этого она будет управлять следующими типами войск: пехота, артиллерия, мотострелковые подразделения, танки и поезда для переброски на дальние расстояния. У каждого подразделения имеется запас сил, боеприпасов, еды, воды, кроме этого при ведении боя необходимо учитывать боевые характеристики: боевой дух, боевой опыт, дальность атаки, скорость передвижения. Все эти характеристики и особенности графа дорог(множество путей, разветвлённость и плотность дорог) должны приниматься во внимание нейронной сетью при достижении цели.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Исследовать применение технологий машинного обучения в играх
2. Добавить возможность играть против ИИ в игру «WarOnMap»
3. Создать ИИ-сервер, который будет создавать и управлять экземплярами нейросетей связанных с играми сокетами.
4. Сформировать критерии эффективности для нейронной сети.
5. Спроектировать нейросеть, которой на вход будет подаваться список юнитов(войск), на выходе команды для игрового сервера, которые будут применены к юнитам.
6. Создать нейросеть
7. Обучить нейросеть на самой себе или реальных игроках
8. Протестировать нейросеть на реальных игроках

## 1.3 Научная новизна

Разработанная нейронная сеть будет первой нейронной сетью решающая задачу на дорожном графе. Результаты эффективности полученной нейронной сети позволят понять возможности «обучения с подкреплением» в задачах на дорожном графе, что поможет использовать «обучения с подкреплением» в других, возможно более сложных, задачах на дорожных графах.

# 2. Аналитическая часть

## 2.1 Обзор исследований

В рамках ВКР был проведён обзор последних исследований в применении машинного обучения в играх, в том числе и игр жанра стратегий реального времени.

### 2.1.1 Бот компании OpenAI игры Dota 2

Компания OpenAI в 2019 году создала бота для игры Dota 2. Представленный бот обучался с нуля на самом себе, без имитационного обучения и поиска по дереву. [id\_2\_1\_1\_\_1]

Dota 2 — компьютерная многопользовательская командная игра в жанре многопользовательская онлайн арена(multiplayer online battle arena, сокр. MOBA), разработанная Valve Corporation. Игра представляет из себя сражение на карте, где в каждом матче участвуют две команды по пять игроков, управляющих «героями» — персонажами с различными наборами способностей. Для победы в матче команда должна уничтожить особый объект-«крепость», принадлежащий вражеской стороне, и защитить от уничтожения собственную «крепость».[id\_2\_1\_1\_\_2]

Dota 2 в режиме 1 на 1 — это сложная игра со скрытой информацией. В процессе обучения бот смог научиться планировать, атаковать, устраивать засады для своих противников, благодаря чему смог обыграть профессиональных игроков.

### 2.1.2 Бот компании OpenAI играющий в сумо и футбол

Также компания OpenAI разработала бота играющего в сумо и футбол. При разработке бота использовалось трансферное обучение(transfer learning). Суть трансферного обучения заключается в том, что для создания нового слоя нейронов мы берём копию другого слоя, который выполнял похожую задачу, и обучаем его.

Применение этого метода позволило быстрее обучить бота выполнению похожих задач. Авторы данного исследования сначала обучили сумо-бота, затем поставили ему вместо реального противника «ветер», который «дул» со случайной стороны. Даже несмотря на то, что с ветром сумо-бот никогда до этого не сталкивался, он хорошо справился со своей новой задачей.[id\_2\_1\_2\_\_1]

Также исследователями было придумано решение проблемы переобучения. Чтобы бот не переобучился в процессе игры с одним видом противников, ему давали разных противников с различными стратегиями. Каждый из этих ботов противников обучался параллельно. Получившийся бот вырабатывал общие стратегии поведения, не заточенные под конкретного противника.

### 2.1.3 Per-Arne Andersen, Morten Goodwin, Ole-Christoffer Granmo «Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games»

В статье обозреваются среды для исследовании технологий машинного обучения. Были рассмотрены следующие среды(в скобках будет указана игра, на которой тестировались технологии машинного обучения)[id\_2\_1\_3\_\_1]:

* Stratagus(Warcraft 2)
* Arcade Learning Environment(Atari 2600)
* microRTS(собственная игра)
* TorchCraft(Starcraft)
* Malmo(Minecraft)
* ViZDoom(Doom)
* DeepMind Lab(головоломки и 3D-навигация)
* OpenAI Gym – набор инструментов для развития и сравнения алгоритмов машинного обучения.
* OpenAI Universe – платформа для исследования алгоритмов машинного обучения, имеет ограниченное количество сред для тестирования
* Extensive Lightweight Flexible (ELF) – первая из сред которая предназначена для игр жанра стратегий в реальном времени.
* StarCraft II Learning Environment(SC2LE)

Все перечисленные среды используют только обучение с подкреплением в качестве основной технологии машинного обучения, так же как и Deep RTS. Авторы также рассмотрели несколько наиболее успешных разработок в этой области:

* TD-Gammon – ИИ, который играет в нарды. Он был разработан Джеральдом Тесоро в 1992 в Научно-исследовательском центре Томаса Дж. Уотсона IBM[id\_2\_1\_3\_\_2, id\_2\_1\_3\_\_3]. TD-Gammon состоит из искусственной нейронной сети (ИНС) с тремя слоями и обучается с помощью метода обучения с подкреплением под названием TD-лямбда. TD-лямбда является алгоритмом учитывающим разницу во времени(TDLambda is a temporal difference learning algorithm), который был изобретен Ричардом С. Саттоном [id\_2\_1\_3\_\_4]. ИНС перебирает все возможные шаги и оценивает вознаграждение за конкретное движение, затем выбирает действие с самым высоким вознаграждением. TD-Gammon - первый ИИ, который обучается на самом себе.
* AlphaGO – первый ИИ, который победил профессионального игрока в Go. AlphaGO является фреймворком обучения с подкреплением, который использует дерево поиска Монте-Карло и две глубоких нейронных сети стоимости и стратегическую оценку.[id\_2\_1\_3\_\_5]
* DeepStack - ИИ, который играет в Техасский Холдем покер. Этот ИИ использует дерево поиска вместе с нейронными сетями для решения проблем с неполной информацией(imperfect information)[id\_2\_1\_3\_\_6]
* Бот для Dota 2. Он рассматривался ранее (см. главу «2.1.1 Бот компании OpenAI игры Dota 2»)

Игровая среда Deep RTS позволяет проводить исследования на разных уровнях сложности в планировании, реагирования и контроле. Deep RTS имеет несколько конфигурации, при этом они являются детерминированными и недолговечными. Действия в недолговечной конфигурации непосредственно применяются к среде в течение нескольких игровых кадров, что делает корреляцию между действием и вознаграждением более наблюдаемой. Авторы решили не использовать долговременные конфигурации, так как они значительно усложняют пространство состояний и время обучения.

Цель игры в Deep RTS состоит в следующем: уничтожить базу другого игрока. В распоряжении игрока один рабочий, который может построить ратушу, казарму и ферму. Ратуша является местом доставки ресурсов и позволяет создавать новых рабочих. В казармах создаются воины, которые более эффективны в бою, чем рабочие. Фермы нужны для повышения максимального количества юнитов.

Все действия в игре сводятся к трём задачам: сбор ресурсов, нападение и оборона. Это позволяет сократить количество игровых состояний, что ускоряет разработку и обучение ИИ. Deep RTS имеет несколько сценариев и позволяет создавать собственные. Представленные сценарии отличаются друг от друга количеством игроков, используемой стратегией поведения и размером карты.

Также Deep RTS имеет настраиваемый игровой таймер, который позволяет ИИ обучаться с минимальной задержкой. Это возможно благодаря тому что, можно настраивать количество тиков в секунде и каждое действие выполняется за определённое количество тиков, а не секунд. Например, для перемещения на 1 клетку нужно 10 тиков, для постройки здания 300 тиков. Также в игру был добавлен специальный алгоритм поиска путей, который называется jump-point-search, с его помощью удалось достичь почти мгновенного нахождения пути.

Поведение юнитов в Deep RTS определяется конечным автоматом. После создания юнита, он находится в состоянии ожидания, и из этого состояния он может перейти в нужное игроку или ИИ состояние.

Deep RTS управляет боевыми единицами с помощью действий, множество которых разделено на 2 абстрактных уровня:

* Первый уровень - это действия, которые непосредственно влияют на среду, например, щелчок правой кнопкой мыши, щелчок левой кнопкой мыши, перемещение влево или выбор юнита.
* Второй уровень абстракции - это действия, объединяющие действия предыдущего уровня, например, select-unit → right-click → right-click → move-left. Преимущество этой абстракции состоит в том, что алгоритмы могут фокусироваться на определенных областях внутри игрового состояния и позволяют создавать иерархические модели, каждая из которых специализируется на определённом виде задач (постройка новой базы, оборона, нападение).

Deep RTS, на момент написания статьи, была одной из самых производительных сред. Deep RTS могла обновлять игру 7 000 000 раз в секунду (на самой маленькой карте 10 на 10 клеток), тогда как ELF только 36000[id\_2\_1\_3\_\_6] и microRTS 11500[id\_2\_1\_3\_\_7]. Таких выдающихся результатов в производительности удалось достичь благодаря настраиваемому таймеру и специальному алгоритму поиска путей jump-point-search.

Среда Deep RTS хорошо справляется со своей задачей, но у неё есть недостаток – не может работать с дорожным графом.

### 2.1.4 Kun Shao, Yuanheng Zhu, Member, IEEE and Dongbin Zhao, Senior Member «StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning»

В данной статье авторы используют технологии машинного обучения для управления армией в игре Starcraft. Разработанный ИИ управлял группой одинаковых юнитов на определённой местности.

#### Постановка задачи и предпосылки

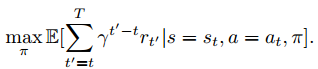
Боевой сценарий с несколькими юнитами аппроксимируется как игра Маркова, мультиагентным расширением марковских процессов принятия решений (Markov decision processes, MDP) [Id\_2\_1\_4\_\_21], [Id\_2\_1\_4\_\_2], [Id\_2\_1\_4\_\_3]. В марковской игре с N агентами, набор состояний S используется для описания свойств всех агентов, и среды, а также набора действий A1, ..., AN и наблюдений O1, ..., ON для каждого агента.

В бою юниты c каждой стороны должны взаимодействовать друг с другом. Разработка модели обучения для нескольких подразделений является сложной задачей в микроуправлении. Чтобы поддерживать гибкую структуру и управлять произвольным количеством юнитов, авторы сделали так, что каждый юнит имеют доступ к пространству состояний S из собственного наблюдения за текущим боем, рассматривая других юнитов как часть среды S → Oi(где i – индекс текущего юнита). Каждый юнит взаимодействует в бою исходя из своих наблюдений и действий. S×A1 ×, ..., × AN → S′ обозначает переход из состояния S в следующее состояние S′ с действиями всех подразделений, а R1 ... RN - сгенерированные вознаграждения каждого юнита. Что касается много-агентной кооперации, эта политика распространяется среди всех юнитов. Цель каждого юнита - максимизировать его общий ожидаемый результат.

Чтобы решить проблему мультиагентного управления в микроменджменте StarCraft, авторы использовали обучение с подкреплением(reinforcement learning, RL). 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 [Id\_2\_1\_4\_\_24].

Процесс взаимодействия агент-среда в RL сформулирован как процесс принятия решений Маркова. Агент в состоянии **S** выполняет действие **A** согласно политике **π**.

Такое поведение применяет вознаграждение **r** и переводит в новое состояние **S0**. Далее определяется будущий дисконтированный результат в момент времени **t** как , где **T** - конечный временной шаг, а **γ** ∈ [0, 1] - коэффициент дисконтирования, который определяет важность будущих вознаграждений. Цель модели RL 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)

Далее авторы провели обзор имеющихся на тот момент технологий обучения с подкреплением:

1. RL алгоритм «обучение с разницей во времени» ( temporal difference (TD) learning), который представляет из себя комбинацию метода Монте-Карло и метода динамического программирования. TD алгоритм может учиться на основе необработанного опыта без модели среды и обновлять оценки на основе части последовательности, не дожидаясь окончательного результата [Id\_2\_1\_4\_\_25]. Наиболее широко известными алгоритмами обучения TD являются Q-learning и Sarsa:
   1. Q-learning. Он оценивает ценность совершения действия в данном состоянии и итеративно обновляет оценку Q-значения в соответствии с наблюдаемым вознаграждением. Ошибка TD δt в Q-обучении вычисляется как δt = r t+1 + γ max Q (st+1, a) - Q (st, at) (2). Q-learning - это метод обучения вне политики, это означает, что он изучает различные политики сравнивая с одним выбранным действием.
   2. Sarsa - это метод, основанный на политике, что означает, что политика используется как для выбора действий, так и для обновления предыдущего значения Q [Id\_2\_1\_4\_\_24]. Правило обновления Sarsa демонстрируется как

δt = rt+1 + γQ(s t+1, a t+1) − Q(st , at) , (3a)

Q(st+1 , at+1) = Q(st , at) + αδt , (3b)

где α - скорость обучения.

1. Обучению с подкреплением с глубокой нейронно-сетевой(deep neural networks, DRL) функцией апроксиматора. DRL предоставляет возможность обучить одного агента для решения ряда задач на уровне человека сквозным способом [Id\_2\_1\_4\_\_30] [Id\_2\_1\_4\_\_31]. Как самый известный алгоритм DRL, глубокая Q-сеть (deep Q-network(DQN)) использует технику воспроизведения опыта и целевую сеть для удаления корреляций между выборками и стабилизации процесса обучения [Id\_2\_1\_4\_\_5]. За последние несколько лет мы стали свидетелями большого числа улучшений DQN, включая двойное DQN [Id\_2\_1\_4\_\_32], приоритетное DQN [Id\_2\_1\_4\_\_33], дуэльное DQN [Id\_2\_1\_4\_\_34], распределенное DQN [Id\_2\_1\_4\_\_35] и асинхронное DQN [Id\_2\_1\_4\_\_36].

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 [Id\_2\_1\_4\_\_37]. Глубокий детерминированный градиент политики (Deep deterministic policy gradient, DDPG) - это непрерывный аналог DQN, который использует критику для оценки ценности текущей политики и актёра для обновления политики [Id\_2\_1\_4\_\_38]. Методы DRL на основе политик играют важную роль в непрерывном контроле, включая асинхронно преимущественная критика актёра (asynchronous advantage actor-critic (A3C)) [Id\_2\_1\_4\_\_36], оптимизацию политики в области доверия (trust region policy optimization , TRPO) [Id\_2\_1\_4\_\_39], оптимизацию проксимальной политики (proximal policy optimization, PPO) [Id\_2\_1\_4\_\_40] и так далее.

Традиционные DRL методы имеют слишком высокую сложность, из-за чего они не применимы в реальных приложениях. Тем не менее они широко используются в сенсомоторном управлении.

1. Поиск руководствующих политик (GPS) использует контролируемый алгоритм обучения для тренировки политике и алгоритм RL генерирует руководящие распределения(guiding distributions), что позволяет эффективно обучать глубоким политикам [Id\_2\_1\_4\_\_41]. Исследователи также предложили некоторые другие основанные на модели методы DRL, такие как нормализованные функции (normalized advantage functions, NAF) [Id\_2\_1\_4\_\_42] и встраивание в контроль (embed to control, E2C) [Id\_2\_1\_4\_\_43].

Многоагентное обучение с подкреплением тесно связанно с областью работы авторов исследования[Id\_2\_1\_4\_\_44]. Многоагентная система включает несколько агентов, взаимодействующих в одной среде [Id\_2\_1\_4\_\_45] [Id\_2\_1\_4\_\_46]. Недавно были предложены некоторые алгоритмы обучения с многоагентным подкреплением с глубокой нейронной сетью для обучения коммуникации [Id\_2\_1\_4\_\_47], кооперативно-конкурентному поведению [Id\_2\_1\_4\_\_23] и несовершенной информации [Id\_2\_1\_4\_\_48]. Авторы используют 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 с собственным опытом. Этот метод может обучать гомогенных агентов более эффективно [Id\_2\_1\_4\_\_49].

Для более быстрого обучения похожим задачам, авторы использовали трансферное обучение(transfer learning, TL). Суть трансферного обучения заключается в том, берётся слой нейронов выполняющий определённую задачу, этот слой копируется (вместе со всеми весами), далее копия будет выполнять другую, но похожую задачу. Применение этого метода позволяет гораздо быстрее обучать нейросеть новым задачам чем, если бы нейросеть обучалась с нуля.

Также авторами был использован метод обучения по плану (curriculum learning). Согласно этому методу нейросеть сначала обучается более простым задачам, которые необходимо выполнить для достижения конечной цели[Id\_2\_1\_4\_\_53]. Авторы статьи приняли решение использовать комбинацию двух предыдущих методов под названием плановое трансферное обучение(curriculum transfer learning , CTL), который хорошо себя показал на практике[Id\_2\_1\_4\_\_54], [Id\_2\_1\_4\_\_55], [Id\_2\_1\_4\_\_56]

Наиболее удобный метод использования CTL - это сначала освоение простых сценариев, а затем решение сложных сценариев на основе полученных знаний. Изменяя количество и типы юнитов, можно контролировать сложность микроуправления. Таким образом, авторы исследования использовали CTL для обучения юнитов с помощью последовательности постепенно усложняющихся сценариев.

#### Представление состояния игры

Представление состояний игры в StarCraft остается открытой проблемой без универсального решения. Оно строится из входных данных из игрового движка, которые имеют разные типы данных и разные степени свободы, как показано в таблице 1. Чтобы решить эту проблему авторы предлагают собственный метод решения. 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.(см. рис. 1)

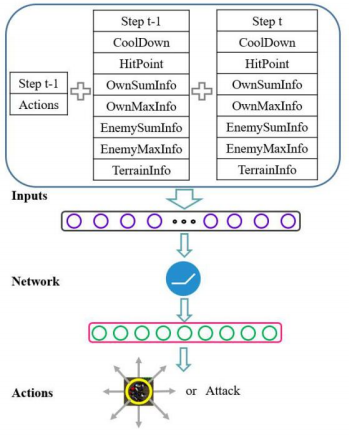


Рисунок 1. Представление модели обучения одного юнита в сценарии микроуправления StarCraft. Представление состояния состоит из трех частей и нейронная сеть используется как аппроксиматор функции. Сеть выводит вероятности движения в 8 направлениях и атаки.

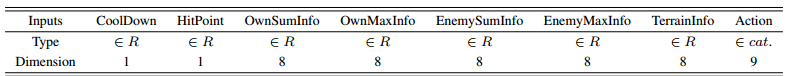


Таблица 1. Типы данных и размерность модели. R – число с плавающей запятой, cat – категория действия, является одним из 8 направлений движения или атакой

Информация о текущем шаге состояния включает в себя: время перезарядки оружия(CoolDown), количество очков здоровья(HitPoint), информацию о расстояниях до своих юнитов, информацию о расстоянии до вражеских юнитов и информацию о расстоянии объектов местности. Информация о состоянии последнего шага совпадает с текущим шагом(если обучение только началось). К сведению принимается последнее действие, которое оказалось полезным для процесса обучения.

Все входы с типом «число с плавающей точкой» нормированы по их максимальным значениям. Среди них CoolDown и HitPoint имеют 1 степень свободы для каждого. Авторы делят боевую карту на 8 секторов в среднем и вычисляют информацию о расстояниях в каждой области. Информация о расстоянии между единицами указана следующим образом:

* OwnSumInfo: расстояния между собственными юнитами суммируются в каждой области;
* OwnMaxInfo: расстояния между собственными подразделениями максимизированы в каждой области;
* EnemySumInfo: расстояния между вражескими подразделениями суммируются в каждой области;
* EnemyMaxInfo: расстояния между вражескими подразделениями максимизированы в каждой области.

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



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



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.

#### Определение действия

В сценариях микроуправления StarCraft исходное пространство действий очень велико. На каждом временном шаге каждый юнит может перемещаться в произвольных направлениях и с произвольными расстояниями на карте. Когда юнит атакует, он может выбрать любого вражесого юнита в радиусе действия своего оружия. Чтобы упростить пространство действий, авторы решили ограничить передвижение юнита 8-ю направлениями с фиксированным расстоянием и при атаке выбирать только самых слабых Когда выбрано передвижение в определённом направлении, юнит повернётся в одно из 8 направлений: вверх, вниз, влево, вправо, вверху слева, вверху справа, внизу слева, внизу справа и переместится на фиксированное расстояние. Когда выбрана атака, юнит остановится и будет атаковать врагов. Авторы решили в качестве цели выбирать цель с самым низким уровнем здоровья в радиусе действия оружия. Согласно результатам эксперимента, этих действий достаточно, чтобы эффективно управлять юнитами в игре.

#### Архитектура нейросети

Поскольку опыт наших юнитов имеет ограниченное подмножество большого пространства состояний, и большинство тестовых состояний никогда не исследовались заранее, будет трудно применить таблицу обучения с подкреплением для изучения оптимальной политики. Чтобы решить эту проблему, мы используем нейронную сеть, параметризованную вектором θ, для аппроксимации значений действия состояния для улучшения обобщения нашей модели RL.

Ввод сети - тензор с 93 измерениями из представления состояния игры. Нейросеть авторов имеет 100 нейронов в скрытом слое, где используется функция активации «усеченное линейное преобразование» (rectified linear unit, ReLU) для нелинейности сети, которая выражается формулой

f(z) = max (0, z), (6)

где z - выход скрытого слоя. Авторы исследования решили использовать функцию ReLU, а не функцию sigmoid или tanh, потому что функция ReLU не имеет проблемы градиентного спуска, что может гарантировать эффективное обучение модели [Id\_2\_1\_4\_\_59]. В отличие от этих насыщающих функций нелинейности, таких как sigmoid или tanh, функция ReLU является ненасыщенной нелинейной функцией. С точки зрения времени обучения с градиентным спуском ненасыщенная нелинейная намного быстрее [Id\_2\_1\_4\_\_60].

Выходной слой нейронной сети имеет 9 нейронов, который выдаёт вероятность движения в 8 направлениях и атаки. Модель обучения одного юнита в сценариях микроуправления StarCraft, включает представление состояния, архитектуру нейронной сети и выходные действия, изображенные на рис. 1.

#### Метод обучения в микроменеджменте

В этой рассматриваемой статье авторы формулируют задачу микроуправления как многоагентную модель обучения с подкреплением. Предлагается метод многоагентного градиентно-спуска Сарса (λ)с разделением параметров(parameter sharing multi-agent gradient-descent Sarsa, PS-MAGDS) для обучения модели, и конструируется функцию вознаграждения в качестве внутренней мотивации для продвижения процесса обучения. Вся схема PS-MAGDS обучения с подкреплением изображена на рис. 2.

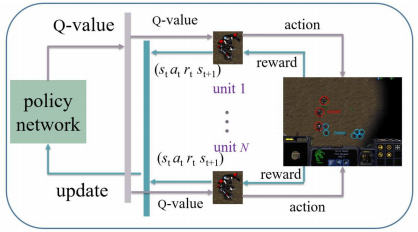


Рисунок 4. Диаграмма обучения с подкреплением PS-MAGDS в сценариях микроуправления StarCraft.

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

Авторы предлагают RL-алгоритм с несколькими агентами, который расширяет традиционный Сарса алгоритм (λ) до нескольких единиц, разделяя параметры сети политики между нашими юнитами. Чтобы ускорить процесс обучения и решить проблему отсроченных вознаграждений, используются приемлемые черты(eligibility traces) в обучении с подкреплением. В качестве основного механизма в RL, приемлемые черты используются для назначения временного вознаграждения, который учитывает набор ранее испытанных переходов [Id\_2\_1\_4\_\_61].

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

В реализации Sarsa (λ) для битв нескольких юнитов, используется нейронная сеть в качестве аппроксиматора функции, и параметры сети разделяются между всеми юнитами. Хотя 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.

Для эффективного обновления политик сети авторы статьи используют 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)

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-greedy для выбора действий во время тренировки, который выбирает текущее наилучшее действие с вероятностью 1 - e и выполняет случайное исследовательское действие с вероятностью e,

(8)

Мы используем экспоненциально # decay для реализации метода e-greedy. e инициализируется со значением 0,5 и график отжига с показателем экспоненциального сглаживания окна в эпизоде с номером эпизода эпизод num, как показано

(9)

Общий 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

#### Функция вознаграждения

Функция вознаграждения обеспечивает полезную обратную связь для RL-агентов, которая оказывает значительное влияние на результаты обучения [Id\_2\_1\_4\_\_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. Задержка вознаграждения затрудняет выяснение того, какой набор действий отвечает за соответствующие вознаграждения.

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 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)

где damage\_amount - это количество урона, нанесенного атакой наших юнитов, коэффициент урона - сила атаки наших юнитов, а количество хитпоинтов юнита — это количество очков здоровья нашего юнита. Мы делим вознаграждение на константу, чтобы изменить его размер 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,

(11)

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.

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 это за то, 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 я установил на -0,5. Согласно эксперименту, это вознаграждение оказывает впечатляющее влияние на эффективность обучения, как показано на рис. 6.

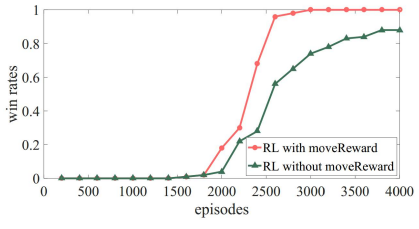


Рисунок 6. Процент побед подразделений в сценарии «3 голиафа против 6 зилотов» через каждые 200 тренировок эпизодов.

#### Пропуск кадров

Применяя обучающее подкрепление к видеоиграм, мы должны обратить внимание на непрерывность действий. Из-за свойства микроуправления StarCraft в реальном времени 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 кадра привнесет сильную корреляцию в обучающие данные, в то время как большой пропуск кадра уменьшит эффективные обучающие выборки. Мы ссылаемся на соответствующую работу в [Id\_2\_1\_4\_\_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.

#### Эксперименты и их результаты

Все сценарии разделены на две группы – мелкомасштабные и крупномасштабные, они отличаются количеством юнитов. В этих сценариях вражеские юниты управляются встроенным ИИ, который жестко запрограммирован в игре. Эпизод заканчивается, когда любая из сторон потеряла всех своих юнитов.

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.

В мелкомасштабных сценариях микроуправления мы будем тренировать Голиафов против вражеских юнитов с разным количеством и типом. 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) Голиафы против зилотов: в этом сценарии мы обучаем отряд наших Голиафов с нуля и анализируем результаты.

* 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 эпизодических тренировок наши отряды могут наконец достичь 100% выигрыша.
* Шаги эпизодов: 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, эпизодов 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 концов, Голиафы выучили соответствующую политику, чтобы уравновесить маневрённость и атаку, и они способны уничтожать врагов почти за 300 шагов.
* 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 шаги эпизода в бою. Кривая среднего вознаграждения наших Голиафов изображена на рис. 8. Средние вознаграждения имеют очевидное увеличение при открытии, и неуклонно растёт во время тренировок и остаются плавными после почти 3000 эпизодов.

2) Голиафы против зерглингов: в этом сценарии вражеские отряды представляют собой группу 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.

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 4000% 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%.

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.

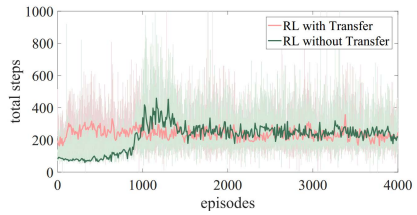


Рисунок. 10. Среднее количество шагов в эпизоде для юнитов в сценарии «3 голиафа против 20 зерлингов» во время обучения.

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.

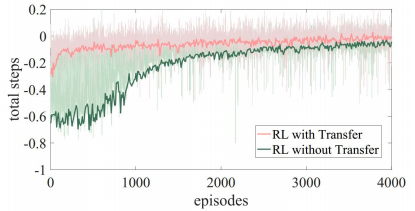


Рисунок 11. Средняя награда наших юнитов в сценарии «3 голиафа против 20 зерлингов» во время обучения.

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

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 основанных на правилах подходах и подходов DRL.

* Слабейшие: метод, основанный на правиле, атаковать слабейших в радиусе действия оружия.
* Ближайший: метод, основанный на правиле, атакуовать ближайшего в радиусе действия оружия.
* GMEZO: метод DRL, основанный на оптимизации нулевого порядка, имеющий впечатляющие результаты по сравнению с традиционными RL-методами [Id\_2\_1\_4\_\_20].
* BicNet: метод DRL, основанный на архитектуре актер-критика, имеющий наилучшую производительность в большинстве сценариев микроуправления StarCraft [Id\_2\_1\_4\_\_21].

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.

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.

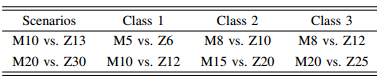


Таблица 3. Классы планового обучения для сценария «Морпехи против Зерглингов». M: морпехи, Z: зерглинги

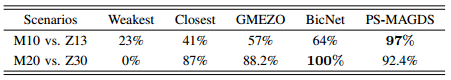


Таблица 4. Сравнение производительности модели с основными методами в двух крупномасштабных сценариях. M: морпехи, Z: зерглинги

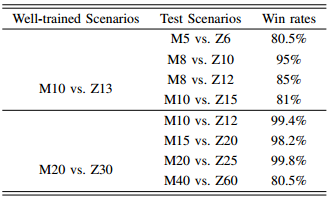


Таблица 5. Вероятность победы в различных сценариях планового обучения с знакомыми и не знакомыми сценариями

#### Анализ стратегий

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.

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) 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.

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.

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) 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) 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) 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.

#### Результаты и будущие работы

Чтобы решить проблему отложенного вознаграждения в микроуправлении StarCraft, авторы использовали простой 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 [Id\_2\_1\_4\_\_63]. По сравнению с методом формирования вознаграждения, иерархический RL обладает способностью изучать абстрагированные от времени исследования, что даёт агентам большую гибкость.

Но его структура также намного сложнее, и автоматическое извлечение подзадач остается открытой проблемой. В будущем мы проведем углубленное исследование по применению иерархического RL к StarCraft. В настоящее время авторы рассматриваемого исследования могут обучать только наземных юнитов дальнего боя, в то время как тренировка наземных юнитов ближнего боя с использованием методов RL остается открытой проблемой. Предложенный метод будет улучшен и использован для большего количества типов юнитов и более сложных сценариев в будущем. Наконец, авторы также рассмотрят возможность использования модели микроуправления для StarCraft бота в полноценной игре.

### 2.1.5 Hendrik Baier, Peter I. Cowling «Evolutionary MCTS for Multi-Action Adversarial Games»

Авторы исследования обозревают различные методы поиска дерева решений и сравнивают их с эволюционным MCTS(метод поиска дерева решений Монте Карло).

#### Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [Id\_2\_1\_5\_\_1], [Id\_2\_1\_5\_\_2] является первым и лучшим алгоритмом поиска дерева на основе стохастических симуляций для оценки состояния, которая была успешно применена к большому множеству игр и других задач [Id\_2\_1\_5\_\_4]. Алгоритм, как правило, строит дерево поиска с узлами, представляющими состояния игры, и рёбра представляющие действия, ведущие от одного состояния до другого. В детерминированной игре и игнорировании перемещений, это может также рассматриваться как дерево, в котором узлы представляют список действий, которые были применены из корневого состояния для достижения их соответствующего состояния – это представление будет полезно позже. MCTS начинает свой поиск с корневого узла, соответствующем текущему состоянию игры. Это тогда повторяет следующий четырехфазовый цикл, пока время вычисления не заканчивается:

1. В фазе выбора, политика выбора используется, чтобы пересечь дерево, пока нерасширенное действие не выбрано. Политика выбора должна уравновесить эксплуатацию состоянии с высокими оценками стоимости и исследованием состояний с неопределенными оценками стоимости. В данной статье используется популярная политика UCB [Id\_2\_1\_5\_\_10].
2. На этапе расширения ранее нерасширенное действие и узел, представляющий его преемника, добавляются в дерево.
3. На этапе развертывания, *политика развертывания* используется для воспроизведения оставшейся части моделируемой игры, начиная с состояния, представленного вновь добавленным узлом. Эта политика развертывания может быть равномерно случайной, но может также извлечь выгоду из знаний эвристических игр. В этой статье мы используем жадное развертывание, которое выбирает случайное действие с вероятностью, а в остальном следуют простой жадной эвристике.
4. Наконец, на этапе обратного распространения оценки. значения всех состояний, пройденных во время симуляции, обновляются в соответствии с результатом законченной игры. Со временем было предложено несколько вариантов и улучшений MCTS, чтобы применить MCTS к играм с более высокими коэффициентами ветвления.

Срочность в первую очередь [Id\_2\_1\_5\_\_11] поощряет эксплуатацию, предоставляя значение для непосещенных дочерних узлов, устраняя необходимость для MCTS посещать каждый дочерний узел, прежде чем можно будет применять политику выбора, такую как UCB. Прогрессивное расширение [Id\_2\_1\_5\_\_12] и расщепление [Id\_2\_1\_5\_\_13] приближаются к проблеме фактора ветвления в Go, сначала ограничивая число действий, развертываемых в новом узле MCTS, а затем увеличивая его с течением времени, чтобы улучшить оценки значений и при этом гарантировать сходимость в пределе.

Для игр с гораздо более высокими факторами ветвления, таких как стратегии в реальном времени (RTS), подходы основанные на скриптах разработаны для поиска небольшого числа рукодельных скриптов вместо большего числа элементарных действий: Поиск по иерархическому портфолио[14] и основанные на скриптах UCT [Id\_2\_1\_5\_\_15] попадают в эту категорию, а также не-MCTS подход Portfolio Greedy Search [Id\_2\_1\_5\_\_16]. В некоторых предыдущих работах варианты MCTS применялись к доменам с очень большими или непрерывными пространствами действий, делая сильно упрощающие предположения, такие как независимость юнитов в игре RTS [Id\_2\_1\_5\_\_17] или сходство «близких» действий в области, основанной на физике [Id\_2\_1\_5\_\_18].

Часто делается предположение, что каждый юнит может выполнять одно действие за временной шаг, как это типично для игр RTS. В этой статье мы не предполагаем независимость юнитов, не связываем действия с юнитами и не предполагаем существование предопределенных политик или сценариев. Тем не менее, мы используем эвристическую функцию оценки, которая кодируется вручную в нашей тестовой области, но в будущем может быть автоматически изучена [Id\_2\_1\_5\_\_3].

Авторы используют два специально адаптированных варианта MCTS в качестве базовых в наших экспериментах, описанных в подразделе III-B. Предложенный EMCTS похож на ванильный MCTS в том смысле, что он использует ту же древовидную структуру поиска выбора, расширения, развертывания и обратного распространения, работая над новым пространством поиска, основанным на эволюции.

#### Online Evolutionary Planning

Эволюционные алгоритмы (EAs) - это класс алгоритмов оптимизации, основанный на естественном отборе, который широко использовался для развития и обучения агентов ИИ для игр [19], [20]. В классическом автономном эволюционном подходе параметры ИИ развиваются с использованием его производительности при игре в игру в качестве фитнес-функции. Эволюция не применяется после окончания обучения и ИИ добавляется в игру [Id\_2\_1\_5\_\_21], [Id\_2\_1\_5\_\_22], [Id\_2\_1\_5\_\_23], [Id\_2\_1\_5\_\_24].  
 Онлайн-эволюция - это новый подход, в котором эволюционные алгоритмы применяются во время игры. Это может принимать форму изменения параметров ИИ во время игры [Id\_2\_1\_5\_\_25]. Тем не менее, также возможно развить следующее действие в текущей запущенной игре. Эволюционный алгоритм со скользящим горизонтом(RHEA) [Id\_2\_1\_5\_\_26], [Id\_2\_1\_5\_\_27], например, развивает будущие последовательности действий фиксированной длины в однопользовательской игре, которые сравниваются путем их моделирования и оценки результирующих игровых состояний. При достижении ограничения по времени алгоритм выполняет первое наилучшее действие в найденной последовательности и продолжает поиск последовательностей действий, начиная со следующего временного шага («скользящий» горизонт поиска).  
 Онлайн-эволюционное планирование (OEP) [Id\_2\_1\_5\_\_28], [Id\_2\_1\_5\_\_7] - это недавний эволюционный подход, применимый к состязательным играм с множеством действий. Он оптимизирует только последовательность действий текущего хода, не обращая внимания на будущие ходы игрока или противника. Поэтому его можно рассматривать как выполнение одной итерации RHEA в начале каждого хода и с горизонтом поиска в один ход. Наилучшая найденная последовательность действий затем выполняется без «накатывания» действия горизонта вперед действием.

OEP начинает свой поиск с создания начальной популяции геномов, каждый из которых представляет полный оборот (последовательность действий с фиксированной длиной). Классический OEP выбирает каждый из этих геномов путем многократного выбора случайных действий, начиная с текущего игрового состояния. Затем эта популяция улучшается из поколения в поколение, пока не истечет заданное время вычислений. Каждое поколение состоит из следующих четырех этапов:

1. Все геномы переводятся в соответствующие фенотипы, игровые состояния возникают в результате применения их последовательности действий к текущему игровому состоянию. Пригодность этих фенотипов затем оценивается с помощью статической эвристической оценки.
2. Геномы с наименьшей пригодностью удаляются из популяции. Доля удаляемых геномов - это параметр, называемый скоростью уничтожения.
3. Каждый из выживших геномов соединяется со случайно выбранным различным геномом и создает потомство посредством равномерного кроссовера. Если этот оператор кроссовера приводит к недопустимому действию в потомстве, он восстанавливается заменой действием другого родителя или иным образом случайным корректным действием.
4. Доля потомства, определяемая параметром, называется скоростью мутации, подвергается мутации. Одно случайно выбранное действие последовательности заменяется другим действием, случайно выбранным из всех разрешенных действий. Если это приводит к некорректным действиям позже в последовательности, они также заменяются случайными корректными действиями.

Когда бюджет времени исчерпан, OEP возвращает последовательность действий, представленную текущим лучшим геномом, которая затем выполняет действие за действием. По словам Wang et al. «Проблема выбора действий рассматривается скорее как проблема оптимизации, а не как проблема планирования» [Id\_2\_1\_5\_\_29]. В настоящее время это наилучший подход к пошаговым многопользовательским состязательным играм, в частности к тестовой области данной статьи: Hero Academy [Id\_2\_1\_5\_\_7]. Он также был применен к другим проблемам, таким как микробитвы [Id\_2\_1\_5\_\_29] или онлайн адаптация порядка действий [Id\_2\_1\_5\_\_30] в RTS играх.

Мы используем исходный OEP, а также новый улучшенный вариант, в качестве базы в наших экспериментах. Предложенный EMCTS аналогичен OEP в том смысле, что в многопользовательских состязательных играх он также ищет пространство полных ходов, которые связаны друг с другом через один и тот же оператор мутации. Он отличается тем, что является алгоритмом поиска по дереву.

#### Hybrids of tree search and evolution

Было опубликовано несколько других методов, которые объединяют идеи из древовидных алгоритмов поиска и эволюционных алгоритмов.

Gaina et al. [Id\_2\_1\_5\_\_31] экспериментировали в General Video Game AI (GVGAI) с разделением общего времени поиска на две части, используя MCTS в первой половине, чтобы сгенирировать первоначальное решение, которое затем уточняется RHEA во второй половине. Это могло превзойти RHEA, но не MCTS. Хорн и соавт. [Id\_2\_1\_5\_\_32] гибридизировали MCTS и RHEA двумя различными способами: используя симуляции Монте-Карло с ограниченной глубиной при оценке геномов RHEA и запуская RHEA и MCTS по отдельности и выбирая лучшее решение, найденное любым из них для выполнения.

EMCTS, с другой стороны, использует единый алгоритм поиска и поиск по дереву со статической оценкой состояния вместо эволюционного поиска с развертыванием для оценки. Лукас и соавт. [Id\_2\_1\_5\_\_33] использовали эволюционный алгоритм для улучшения политики развертывания MCTS во время поиска. Перес-Либана и соавт. [Id\_2\_1\_5\_\_34] адаптировали подобный метод для GVGAI, объединив его с базой знаний, чтобы улучшить расчеты вознаграждения данных состояний. Несмотря на повышение производительности MCTS или RHEA в различных однопользовательских играх, алгоритмы, разработанные для среды GVGAI, не могут напрямую применяться в многопользовательских состязательных играх.

Для состязательных игр Hong et al. [Id\_2\_1\_5\_\_35] предложили стратегию эволюции путей через игровое дерево с помощью эволюционного алгоритма. Хотя их подход предполагает наличие одинаковых действий, доступных во всех состояниях на одной и той же глубине поиска, чего нет в большинстве реальных игр, в том числе в нашей проверяемой игры Hero Academy, он дает интересное указание на возможную будущую работу, которая может позволить учитывать действия противника.

#### Игра для тестирования: Hero Academy

**Правила**. Наша тестовая область - это упрощенный Java-клон [Id\_2\_1\_5\_\_36] Hero Academy [Id\_2\_1\_5\_\_37], пошаговая тактическая игра для двух игроков. Игроки могут использовать различных боевых юнитов, предметы и заклинания, сначала вытягивая их из колоды карт, а затем ставить, применяя заклинания или перемещая юнито их на поле битвы 9 × 5 клеток. Специальные квадраты на этом поле битвы позволяют поставить юнитов, повысить характеристики отдельных юнитов или представить два кристалла игрока. Игра будет выиграна первым игроком, который либо уничтожит всех вражеских юнитов, либо уничтожит оба вражеских кристалла. Более подробную информацию о реализации и правилах можно найти в [Id\_2\_1\_5\_\_28].

Центральная механика игры - очки действия (AP). За каждый ход игрок получает количество очков действия - пять в стандартной форме игры. Каждое очко действия может использоваться для любого простого действия, такого как размещение юнита игрока на поле битвы, перемещение юнита на поле битвы, атака вражеского юнита, лечение дружественного юнита и так далее. Игрок может потратить любое количество очков действия на одного юнита, например, перемещая его несколько раз. При среднем числе 30-60 действий, доступных для каждого игрового состояния, в зависимости от стиля игры, полный коэффициент ветвления за ход может быть приблизительно оценен как 305 ≈ 2,4 × 106 до 605 ≈ 7,8 × 108. Поиск наилучшей последовательности действий для любого хода будет следовательно сам по себе сложной поисковой задачей.

Порядок карт в колоде, а также карты противника неизвестны игроку “Академии героев”. Тем не менее, эта статья посвящена проблеме множества ходов, игнорируя аспекты скрытой информации и индетерминизма, как в [Id\_2\_1\_5\_\_7]. В соответствии с предыдущей работой Justesen и соавторов над Hero Academy мы используем игровые знания для оценки состояния, а также для обрезки и упорядочения действий:

**Оценка состояния**. Все алгоритмы, сравниваемые в этой статье, используют одну и ту же эвристическую функцию оценки. Эта функция представляет собой линейную комбинацию функций, таких как текущее состояние здоровья отдельных юнитов, оснащены ли они определенными предметами и стоят ли они на специальных клетках. Усовершенствование этой вручную написанной функции с помощью машинного обучения и тестирования, если наши выводы все еще верны, они стоят дальнейшей работы.  
 **Сокращение действий и порядок**. Все алгоритмы, сравниваемые в этой статье, используют форму жесткого сокращения, удаляя ряд избыточных или доказанных неоптимальных действий из набора доступных действий, рассматриваемых в любом выданном состоянии. В двух вариантах MCTS, рассматриваемых в качестве базовых, также используется статическое упорядочение действий, что дает приоритет наиболее перспективным действиям на этапах их расширения и просмотра. Используемая для этого эвристика проще и быстрее, чем функция оценки. Заинтересованный читатель может обратиться к [28] для полного определения эвристической функции оценки и стратегий сокращения и упорядочения.

#### Базовые подходы

Чтобы сделать наши результаты прямо сопоставимыми с литературой, мы проверяем наш подход на основе пяти алгоритмов, описанных в [Id\_2\_1\_5\_\_7]. Четыре из них - это методы поиска по дереву, а один - онлайн-эволюционное планирование, представляющее современное состояние “Академии героев”.  
 **Жадное Действие**. AI с жадным поиском действия выбирает первое действие своего хода простым однослойным поиском всех корректных действий, максимизируя эвристическую оценку в немедленно возникающим состоянии. Это повторяется для каждого очка действия, то есть для всех пяти действий хода.

**Жадный ход**. AI с жадным ходом выбирает свои действия, пытаясь выполнить пятислойный поиск в глубину всего хода, максимизируя эвристическую оценку листовых состояний, полученных в результате полных ходов. Он использует таблицу транспонирования, чтобы избежать повторного посещения состояний. Действия упорядочены для поиска с помощью функции оценки, что особенно важно, поскольку «Жадный ход» обычно не может осуществлять полный поиск по всему ходу в заданный срок.

**Не исследующий MCTS**. Этот AI является первым вариантом MCTS, адаптированным для многопользовательских состязательных игр в [Id\_2\_1\_5\_\_7]. Он ищет игровое дерево, как показано на рисунке 2, в котором каждое ребро представляет дополнительное действие для рассматриваемого хода (или его применения). Следующий ход противника может быть достигнут деревом глубже, чем пять слоев, количество очков действия. Политика выбора этого варианта MCTS – это UCB, и политика развертывания определенно следует эвристике упорядочивания действий. Было обнаружено, что производительность улучшается, когда выкаты достаточно длинные, чтобы завершить текущий ход игрока, чтобы действовать в листовом узле, вызывая эвристическое средство оценки состояния в конце хода для получения результата выкатывания. Коэффициент разведки MCTS установлен в C = 0 в попытке вырастить достаточно глубокое дерево (чистая эксплуатация).

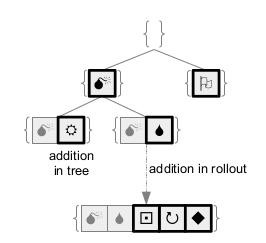
****

Рисунок. 2: Древовидная структура в поисках классическом MCTS и ее вариантов (не объяснено MCTS, BB-MCTS). Узлы представляют собой частичные последовательности действий или состояния, возникающие из них. Края представляют собой добавление атомарного действия к последовательности действий или применение атомарного действия к состоянию. После каждого расширения узла выполняется прокатка для оценки. (Мы используем символы для представления различных атомарных действий.)

**MCTS “Сжигание моста”(Bridge-burning MCTS, BB-MCTS).** Этот вариант MCTS ищет дерево того же типа, показанное на рисунке 2. Вместо детерминированных прокаток он использует -жадную прокатку с = 0.5, которая также достигают только конца текущего хода конечного узла. Его разведочный коэффициент C = 1 / √2. Однако для того, чтобы вырастить достаточно глубокое дерево для ходо с несколькими действиями, он использует технику, называемую «сжигание моста» в [Id\_2\_1\_5\_\_7] - переизобретение поиска по шагам [Id\_2\_1\_5\_\_8]. Мы сохраняем здесь термин «сжигание мостов», так как термин «движение» в “Академии героев” неоднозначен, а также потому, что мы собираемся обобщить концепцию “сжигания мостов” для другого вида дерева в следующем подразделе.

Идея BB-MCTS состоит в том, чтобы разделить бюджет времени для текущего поиска хода на пять этапов, равных количеству действий за ход. На каждой фазе поиск MCTS продолжается в обычном режиме, но в конце каждой фазы выполняет наиболее многообещающее действие из корня, ведущее к корневому состоянию для следующей фазы. Это может быть реализовано как стратегия жесткого сокращения, показанная на рисунке 3.

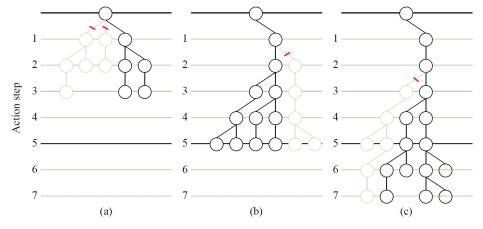


Рис. 3: Стратегия поиска «сжигания моста» (иллюстрация адаптирована из [Id\_2\_1\_5\_\_7]). (а) После фазы 1 все ветви, кроме лучшей, обрезаются в корне. (б, в) После фаз 2, 3,. , , n, обрезка применяется на глубине 2, 3,. , , п. Частичное дерево ниже лучшей ветви сохраняется.

**Эволюционное онлайн-планирование**. Основа OEP соответствует описанию в подразделе II-B. В наших экспериментах мы использовали те же настройки параметров, которые были предложены в [Id\_2\_1\_5\_\_7]: размер популяции 100, коэффициент уничтожения 0,5, коэффициент мутации 0,1 и операторы равномерного кроссовера и мутации.

Этот алгоритм в настоящее время является наиболее эффективным подходом для многошаговых пошаговых игр, таких как “Академии героев”. Хотя [Id\_2\_1\_5\_\_7] показывает, что он имеет силу, аналогичную неисследованным MCTS и BB-MCTS в стандартной форме игры с 5 очками действия за ход, было показано, что OEP лучше масштабируется для более сложных задач “Академии героев” с использованием 10 AP или больше. Наши эксперименты включают в себя и экспоненциально более сложные варианты.

#### Evolutionary MCTS

В этом подразделе предлагается наш новый алгоритм поиска, Evolutionary MCTS или EMCTS, применительно к игре в многопользовательские пошаговые состязательные игры. Он объединяет поиск по дереву MCTS с подходом эволюционных алгоритмов на основе генома.

Вместо обычного дерева MCTS, показанного на рисунке 2, EMCTS строит дерево, как показано на рисунке 4. Вместо того, чтобы начинать с пустого хода в корне, EMCTS запускается с полной последовательности из пяти (или более, в зависимости от предметной области) действий. - так же, как геномы ОЭП. Вместо того чтобы выращивать дерево, которое добавляет одно действие к текущей последовательности с каждым ребром, EMCTS выращивает дерево, которое мутирует текущую последовательность с каждым ребром, используя тот же оператор мутации, что и OEP. И вместо того, чтобы использовать прокатку для завершения текущего хода, и затем оценивать его, как это делают наши исходный показатели MCTS, мы просто оцениваем решения на конечных узлах2. Обратное распространение без изменений.

EMCTS не применяет мутации случайным образом, но может точно выбратьдействие в последовательности которое должно мутировать и какие другие корректные действия мутируют в 3. В то время как OEP превратил планирование последовательности действий в задачу оптимизации, EMCTS, таким образом, берет эволюционную оптимизацию последовательности и превращает ее обратно в проблему планирования. Это можно рассматривать как поиск по дереву, но это также можно рассматривать как систематическое исследование мутационного ландшафта OEP, что дает эволюции преимущество прогнозирования.

Чтобы полностью раскрыть EMCTS, необходимо ответить на два вопроса. **Во-первых, откуда берется корневая последовательность?** EMCTS нуждается в исходном решении для поиска, так же ЭА, такие как ОЭП нуждаются в стартовой попуояции решения. Возможны разные подходы - в этой статье мы используем алгоритм Greedy Action, описанный выше, для быстрой и жадной инициализации корня. **Во-вторых, что происходит, когда мутация приводит к недопустимой последовательности действий?** Мы могли бы отфильтровать их, заранее смоделировав каждую возможную мутацию, но это было бы вычислительно дорого. Вместо этого, подобно OEP, мы применяем классический эволюционный алгоритм с использованием стратегии восстановления - в этой статье мы используем AI с жадным поиском действий также для ремонта, когда это необходимо.

Обратите внимание, что использование Greedy Action не вводит дополнительных эвристических знаний, так как все алгоритмы, сравниваемые в этой статье, работают с одной и той же функцией оценки. Однако мы заметили, что, как и EMCTS, OEP также можно значительно улучшить, **используя политику восстановления Greedy Action вместо политики случайного восстановления**. Это приводит к более высокому качеству ремонта в среднем случае. И точно так же, как EMCTS получает прибыль от генома жадного корня, OEP может получить прибыль от заполнения 20% начальной популяции последовательностями Greedy Action вместо случайных4. Это запускает поиск с более качественными стартовыми решениями. Мы называем этот новый вариант жадным OEP здесь, в отличие от ванильного OEP со случайным ремонтом и чисто случайным начальным населением, как описано в [Id\_2\_1\_5\_\_28], [Id\_2\_1\_5\_\_9], [Id\_2\_1\_5\_\_7], и включаем его в наши эксперименты для справедливого сравнения.

Наконец, EMCTS приводит к еще большему коэффициенту ветвления, чем классические варианты MCTS. В то время как коэффициент ветвления в игре “Академия Героев” между базовыми уровнями MCTS составлял от 30 до 40, коэффициент ветвления дерева мутаций EMCTS составляет около 30 на точку действия, т.е. около 150 для стандартных настроек игры с пятью очков действий. Мы обнаружили, что эффективным способом борьбы с этим является «прожигание моста», так же, как оно применяется к обычному дереву MCTS BB-MCTS. Вместо выполнения наиболее многообещающего действия в корне после каждой фазы поиска, такой как BB-MCTS, EMCTS выполняет наиболее многообещающую мутацию в корне после каждой фазы. Количество фаз горения моста, последовательных поисков и сокращений и мутаций, является единственным параметром EMCTS, который мы настроили (см. Следующий раздел). Коэффициент исследования MCTS был установлен на C = 0. Политика выбора - UCB, как и в других вариантах MCTS.

#### Эксперименты и их результаты

Мы протестировали EMCTS в “Академии героев” против Greedy Action, Greedy Turn, не исследующего MCTS, BB-MCTS и классического OEP, как предложено в [Id\_2\_1\_5\_\_7], а также улучшенной жадной OEP, как было предложено в предыдущем разделе. Все сравнения проводились на стандартных настройках игры с 5 очками действия за ход, но также и с измененными правилами, позволяющими 10 AP или даже 15 AP за ход 5. Это увеличивает сложность одного хода в геометрической прогрессии, но дает более сильный показатель обобщения для других игр, которые могут иметь большее количество возможных действий за ход. Кроме того, все сравнения выполнялись при разных временных бюджетах: 200 мс на ход, 1 секунда на ход и 5 секунд на ход. Каждое сравнение состояло из 400 игр, при этом EMCTS играл 200 игр в качестве первого игрока и 200 игр в качестве второго игрока. Используемая карта показана на рисунке 1. Игры, в которых не было победителя после 200 ходов, учитывались как ничьи, то есть половина выигрыша для каждого игрока.

Все алгоритмы использовали настройки параметров, описанные в разделе III. Количество фаз «горения моста» для EMCTS было определено в предварительных экспериментах и установлено равным 20 в течение 200 мс, 40 в течение 1 секунды и 100 в течение 5 секунд на контроль времени хода. Количество фаз для BB-MCTS было идентичным количеству очков действий на ход, поскольку она ищет тип дерева, показанный на рисунке 2, и не получает выгоды от более глубоких поисков. Поскольку никакой другой алгоритм не был изменен на основе AP(очков действий) за ход, EMCTS также не был специально настроен для разных AP.

Таблица I показывает производительность предлагаемой Evolutionary MCTS в сравнении с пятью базовыми показателями и улучшенным жадным эволюционным онлайн планирование.

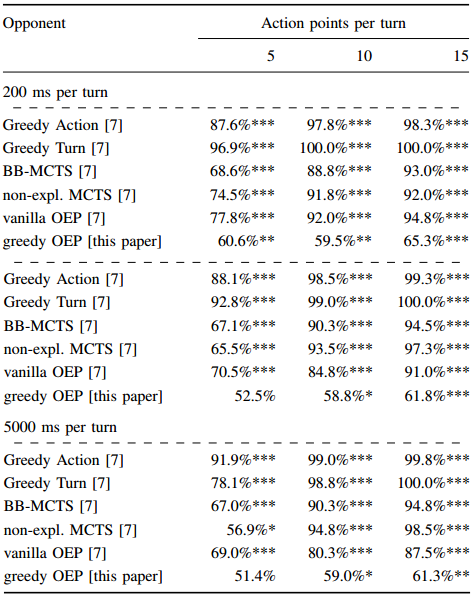


Таблица I: Коэффициенты выигрыша EMCTS в сравнении со всеми базовыми показателями в разное время управления. 400 игр на точку данных. Звездочки указывают на более сильная игру EMCTS: \* р <0,05, \*\* р <0,01, \*\*\* р <0,001

EMCTS значительно сильнее всех базовых алгоритмов (Greedy Action, Greedy Turn, BB-MCTS, не-expl. MCTS и классический OEP) во всех контролях времени и всех количествах очков действий на ход. Его относительная сила увеличивается со сложностью задачи поиска, измеряемой в очках действий на ход. Недавно предложенный жадный OEP является существенным улучшением по сравнению с классическим OEP, как описано в [Id\_2\_1\_5\_\_7], но все еще значительно слабее, чем EMCTS при всех выбранных количествах очков действий при 200 мс на ход и во всех точках действия, за исключением самого низкого значения (5) в 1 с и 5 с за ход, где оба алгоритма работают одинаково. Таким образом, результаты показывают, что Эволюционный MCTS очень эффективен при различных контролях времени и лучше масштабируется в зависимости от сложности предметной области, чем все другие проверенные подходы.

### 2.1.6 AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

AlphaStar — первая система искусственного интеллекта, способная победить лучших профессиональных игроков. В серии матчей, которые состоялись 19 декабря, AlphaStar одержала уверенную победу над Grzegorz Komincz (MaNa) из команды Liquid, одного из сильнейших игроков в мире, со счетом 5:0[Id\_2\_1\_6\_\_1]. Перед этим также был сыгран успешный показательный матч против его товарища по команде Dario Wünsch (TLO). Матчи прошли по всем профессиональным правилам на специальной турнирной карте и без каких-либо ограничений.

AlphaStar играет в полноценную игру (не только управляет армией, но и строит базу, развивает экономику), используя глубокие нейронные сети, которые обучаются на основе необработанных игровых данных, с помощью методов обучения с учителем и обучения с подкреплением.

StarCraft II — это выдуманная фантастическая вселенная с богатым, многоуровневым геймплеем. Существует множество режимов игры, но самый распространенный в киберспорте — это турниры «один-на-один», состоящие из 5 матчей. Сначала игрок выбирает одну из трех рас — зергов, протоссов или терранов, каждая из которых имеет свои особенности и возможности. Каждый игрок в начале игры имеет 16 рабочих, которые добывают ресурсы для постройки зданий, других юнитов или развития технологий, и главное здание. Для победы игрок должен грамотно использовать имеющиеся ресурсы и своевременно вкладывать накопленные ресурсы в экономику, армию или улучшения, в зависимости от ситуации(это называется «макро»-контролем), также от контроля отдельными юнитами(это называется «микро»-контролем) зависит исход отдельных сражений.

Для достижения победы в игре Starcraft 2, ИИ необходимо балансировать краткосрочные и долгосрочные цели, и адаптироваться к непредвиденным ситуациям. Большинство систем часто оказываются совершенно негибкими, а потому не могут справиться с поставленной задачей. Решение этой проблемы требует прорыва в нескольких областях ИИ:

* Теория игр: StarCraft — это игра, где нет одной победной стратегии. На каждую стратегию как в «камень, ножницы, бумага» есть своя контр-стратегия. Поэтому в процессе обучения ИИ должен постоянно исследовать и расширять спектр своих стратегий.
* Неполная информация: В рассматриваемой игре присутствует туман войны, из-за которого не видно юнитов противника, для того чтобы увидеть противника необходимо заниматься разведкой
* Долгосрочное планирование: Как и в реальных задачах, причинно-следственные связи могут не быть мгновенными. Игра также может длиться часа и больше, поэтому действия, совершенные в начале игры, могут не иметь в том числе абсолютно никакого значения в долгосрочной перспективе.
* Реальное время: В StarCraft'е игроки совершают действия непрерывно, наряду с ходом времени, в отличие от пошаговых игр, вроде шахмат.
* Огромное пространство действий: В реальном времени нужно управлять множеством юнитов и зданий, что создает огромное количество комбинаций юнитов и действий. В дополнение к этому, многие действия являются иерархическими и могут изменяться и дополняться по ходу. Параметризация игры дает в среднем примерно от 10 до 26 действий в единицу времени.

Создатели AlphaStar совместно с Blizzard в 2016 и 2017 году, опубликовали набор инструментов PySC2, включающий в себя самый большой, из когда-либо изданных массив, анонимизированных реплеев(записи игр). На полученных реплеях обучался AlphaStar.

#### Как происходит обучение

AlphaStar является нейронной сетью глубокого обучения, которая получает через интерфейс сырые данные (список юнитов и их свойств) и дает на выходе последовательность инструкций, которые являются действиями в игре. Если говорить точнее, архитектура нейронной сети применяет подход преобразование модели юнита, в сочетании с глубоким LSTM ядром, авторегрессионной политикой главного центра с сетью указателей и централизованным базовым значением («transformer torso to the units, combined with a deep LSTM core, an auto-regressive policy head with a pointer network, and a centralised value baseline»). Разработчики AlphaStar предполагают, что полученные модели в дальнейшем помогут справиться с другими задачами машинного обучения, среди которых моделирование долгосрочных последовательностей и большие выходные пространства, такие как перевод, моделирование языков и визуальные представления.

AlphaStar использует новый мультиагентный алгоритм обучения. Изначально представленная нейросеть была обучена с помощью метода обучения с учителем на основе реплеев, о которых говорилось ранее. Благодаря этому AlphaStar смог изучить и сымитировать основные микро- и макро-стратегии, используемые игроками. Представленный агент победил встроенного ИИ уровня «Elite», что эквивалентно уровню игрока золотой лиги, в 95% тестовых игр.

Полученные результаты затем используются для мультиагентного процесса обучения с подкреплением. Для этого была создана лига, где агенты-оппоненты играют против друг друга, подобно тому, как люди получают опыт, играя на турнирах. Новые соперники добавлялись в лигу путем дублирования текущих агентов. Такая новая форма обучения, заимствуя некоторые идеи из метода обучения с подкреплением с элементами генетических алгоритмов, позволяет создать непрерывный процесс исследования огромного пространства стратегий StarCraft'а, и быть уверенным, что агенты в силах противостоять наиболее сильным стратегиям, не забывая при этом старые.

В ходе развития лиги появлялись различные стратегии и контр-стратегии, которые были способны победить предыдущие. В то время когда одни агенты совершенствовали старые стратегии, другие агенты создавали абсолютно новые. Например, на ранней стадии были актуальными так называемые «чизы» — быстрые атаки с помощью фотонных (Photon canon) пушек или темных тамплиеров (Dark templarss). В процессе обучения эти рискованные стратегии были отброшены, уступив место другим. Например, производство избыточного количества рабочих для получения дополнительного притока ресурсов или размен двух оракулов (Oracles) на рабочих противника и подрыва его экономики. Похожим образом игроки-люди открывали для себя новые стратегии, заменяя слишком рискованные и неактуальные более надежными.

Для обеспечения разнообразия, каждый агент наделялся собственной целью обучения. Например, один агент может иметь цель победить конкретного противника, а другой — целый ряд оппонентов, но сделать это только конкретными юнитами. Эти цели менялись по ходу процесса обучения.

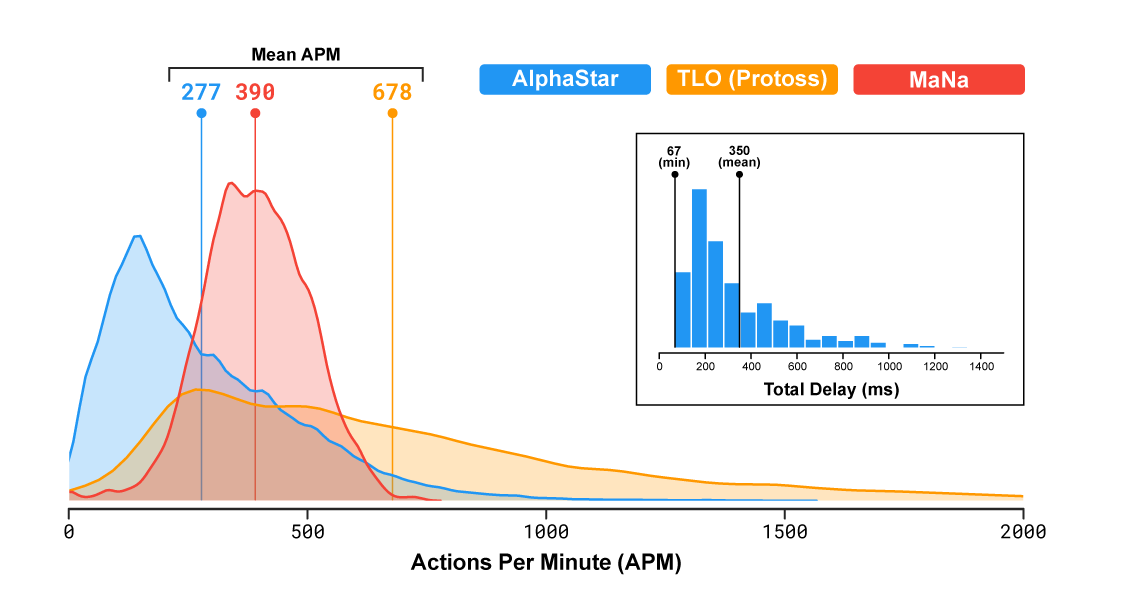
Коэффициенты (веса) нейронной сети каждого агента обновлялись с помощью обучения с подкреплением на основе игр с оппонентами, чтобы оптимизировать их для конкретной цели обучения. Правило обновления весов — это новый эффективный алгоритм обучения вне политичный алгоритм обучения с подкреплением с критикой агентов на основе реплеев, с само-имитационным обучением и политикой дистилляции(off-policy actor-critic reinforcement learning algorithm with experience replay, self-imitation learning and policy distillation)

Для обучения AlphaStar разработчики создали масштабируемую распределенную систему на основе Google TPU 3, которая обеспечивает процесс параллельного обучения целой популяции агентов с тысячами запущенных игр. Лига AlphaStar проработала 14 дней, используя 16 TPU для каждого агента. В ходе обучения каждый агент получил 200 лет опыта игры в StarCraft в реальном времени. Финальная версия агента AlphaStar содержит компоненты распределения Нэша всей лиги[Id\_2\_1\_6\_\_2]. Другими словами, самую эффективную смесь стратегий, которые были обнаружены в ходе игр. И эта конфигурация может быть запущена на одном стандартном настольном GPU. Полное техническое описание готовится к публикации в рецензируемом научном журнале.

#### Как AlphaStar действует и видит игру

Количество действий в минуту у профессиональных игроков в среднем 300 действий в минуту(actions per minute, APM). Это значительно меньше, чем у существующих ботов[id\_2\_1\_5\_\_3], которые могут совершить десятки тысяч действий в секунду и при этом управлять каждым юнитов в отдельности.

В играх против TLO и MaNa, AlphaStar имел 280 среднего APM, что намного меньше, чем у профессиональных игроков, но действия ИИ при этом более точные. Такой низкий APM объясняется в частности и тем, что AlphaStar учился на основе реплеев обычных игроков и пытался подражать манере игроков-людей. Кроме того, AlphaStar реагирует с задержкой между наблюдением и действием в среднем около 350 мс.

Рисунок 12. Распределение APM AlphaStar в матчах против MaNa и TLO, и общая задержка между наблюдением и действием.

В первом противостоянии против TLO и MaNa, AlphaStar взаимодействовал с игровым движком StarCraft'а через базовый (raw) интерфейс, из-за чего он мог видеть атрибуты своих и видимых вражеских юнитов на карте напрямую, без необходимости двигать камеру — эффективно играть с уменьшенным видом всей территории. В это время живые люди должны явно управлять «экономикой внимания». Под «экономикой внимания» подразумевается то, что игрок может управлять только теми юнитами, которые находятся в поля зрения камеры, и поэтому игроку нужно постоянно переключаться между разными участками.

Затем была создана вторая версия AlphaStar с камерой как у игрока человека. Эту версию MaNa смог победить. Хотя вторая версия обучалась 7 дней, а не 14 как первая, она всё равно хорошо себя показала. Разработчики AlphaStar планируют в ближайшее время дообучить вторую версию, и оценить возможности этой версии.

Игры AlphaStar против TLO и MaNa, показывают что успех AlphaStar является в первую очередь следствием грамотного использования стратегий, а не быстрой реакции и большого количества действий.

#### Итоги игры AlphaStar против профессиональных игроков

Разработчики решили, что AlphaStar будет на данный момент специализироваться только на протоссах(одна из трёх рас: терранов, зергов, протоссов). Агенты обучались в StarCraft II версии 4.6.2 в режиме протосс против протосса, на карте CatalystLE. Для оценки производительности AlphaStar, агенты тренировались против TLO — профессионального игрока за зергов и игрока за протоссов уровня «GrandMaster»(высший уровень). AlphaStar выиграл со счетом 5:0 в свою пользу, используя широкий диапазон юнитов и стратегий.

После дополнительной недели обучения, AlphaStar сыграл против MaNa, одного из самых сильных игроков StarCraft II в мире, и входящего в 10 сильнейших игроков за протоссов. AlphaStar и в этот раз победил со счетом 5:0.

#### AlphaStar и другие сложные проблемы

Разработчики AlphaStar считают что техники, лежащие в основе AlphaStar, могут быть полезны в решении других задач. Например, использованный тип архитектуры нейронной сети способен моделировать очень длинные последовательности действий, в играх длящихся около часа и содержащих десятки тысяч действий, основанных при этом на неполной информации. Каждый кадр в StarCraft'е используется как один шаг для нейросети. При этом нейронная сеть каждый такой шаг предсказывает ожидаемую последовательность действий для всей оставшейся игры. Фундаментальная задача составления сложных прогнозов для очень длинных последовательностей данных встречается во многих задачах реального мира, таких как прогноз погоды, моделирование климата, понимание языка и др.

Создатели AlphaStar также рассчитывают, что некоторые из использованных ими методов обучения могут оказаться полезными в изучении безопасности и надежности ИИ. Одна из самых сложных проблем в области ИИ — это большое количество ошибочных вариантов действий. У встроенного ИИ в StarCraft'е есть свои уязвимости, которые профессиональные игроки быстро находят. Инновационный подход AlphaStar, основанный на обучении в лиге, находит такие подходы и делает общий процесс более надежным и защищенным от подобных ошибок. Такой подход может помочь в улучшении безопасности и надежности ИИ-систем в целом. В особенности, в таких критичных областях, как энергетика, где крайне важно правильно реагировать в сложных ситуациях.

# Список литературы

1. OpenAI and Dota 2 [Электронный ресурс] - Режим доступа: <https://openai.com/blog/dota-2/> , свободный
2. Competitive Self-Play [Электронный ресурс] - Режим доступа: <https://openai.com/blog/competitive-self-play/> , свободный
3. Per-Arne Andersen, Morten Goodwin, Ole-Christoffer Granmo «Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games» [Электронный ресурс] - Режим доступа: <https://arxiv.org/abs/1808.05032>, свободный
4. Kun Shao, Yuanheng Zhu, Member, IEEE and Dongbin Zhao, Senior Member «StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning» [Электронный ресурс] - Режим доступа: https://arxiv.org/pdf/1804.00810.pdf, свободный
5. Hendrik Baier, Peter I. Cowling «Evolutionary MCTS for Multi-Action Adversarial Games» [Электронный ресурс] - Режим доступа: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjqpb-r-qbgAhWI8ywKHdj5B5YQFjAAegQICRAC&url=https%3A%2F%2Fhendrikbaier.jimdo.com%2Fapp%2Fdownload%2F13604570727%2Fevolutionary%2520mcts%2520for%2520multi-action%2520adversarial%2520games.pdf%3Ft%3D1532703212&usg=AOvVaw3M9xX0AbJl4h5uYAy8GBBh, свободный
6. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II [Электронный ресурс] - Режим доступа: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ , свободный
7. G. Tesauro, “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play,” Neural Computation, vol. 6, no. 2, pp. 215–219, 1994. [Online]. Available: <http://www.mitpressjournals.org/doi/10.1162/neco.1994.6.2.215>
8. G. Tesauro, “Temporal difference learning and TD-Gammon,”*Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=203330.203343>
9. R. S. Sutton and A. G. Barto, “Chapter 12: Introductions,” Acta Physiologica Scandinavica, vol. 48, no. Mowrer 1960, pp. 57–63, 1960. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, 2016.
10. M. Moravc ˇ´ık, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, ´ T. Davis, K. Waugh, M. Johanson, and M. Bowling, “DeepStack: Expert-level artificial intelligence in heads-up no-limit poker,” Science, vol. 356, no. 6337, pp. 508–513, jan 2017. [Online]. Available: <http://arxiv.org/abs/1701.01724http://dx.doi.org/10.1126/science.aam6960>
11. Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, “ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games,” Advances in Neural Information Processing Systems, pp. 2656–2666, jul 2017. [Online]. Available: <http://arxiv.org/abs/1707.01067>
12. S. Ontanon, “The combinatorial multi-armed bandit problem and its application to real-time strategy games,” in Ninth Artificial Intelligence and Interactive Digital . . ., 2013, pp. 58–64. [Online]. Available: <http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377>
13. id\_2\_1\_5\_\_1 David Balduzzi, Karl Tuyls, Julien Perolat, Thore Graepel “Re-evaluating Evaluation” 2018, [Электронный ресурс] - Режим доступа: https://papers.nips.cc/paper/7588-re-evaluating-evaluation.pdf, свободный
14. id\_2\_1\_5\_\_2 Starcraft AI Competion, [Электронный ресурс] - Режим доступа: http://www.cs.mun.ca/~dchurchill/starcraftaicomp/results.shtml, свободный
15. id\_2\_1\_1\_\_1 Википедия. Dota 2[Электронный ресурс] - Режим доступа: <https://ru.wikipedia.org/wiki/Dota_2> , свободный
16. id\_2\_1\_1\_\_2 OpenAI and Dota 2 [Электронный ресурс] - Режим доступа: https://openai.com/blog/dota-2/ , свободный
17. id\_2\_1\_2\_\_1 Competitive Self-Play [Электронный ресурс] - Режим доступа: <https://openai.com/blog/competitive-self-play/> , свободный
18. id\_2\_1\_3\_\_1 Per-Arne Andersen, Morten Goodwin, Ole-Christoffer Granmo «Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games» [Электронный ресурс] - Режим доступа: https://arxiv.org/abs/1808.05032, свободный4
19. id\_2\_1\_3\_\_2 G. Tesauro, “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play,” Neural Computation, vol. 6, no. 2, pp. 215–219, 1994. [Online]. Available: <http://www.mitpressjournals.org/doi/10.1162/neco.1994.6.2.215>
20. id\_2\_1\_3\_\_3 G. Tesauro, “Temporal difference learning and TD-Gammon,”*Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=203330.203343>
21. id\_2\_1\_3\_\_4 R. S. Sutton and A. G. Barto, “Chapter 12: Introductions,” Acta Physiologica Scandinavica, vol. 48, no. Mowrer 1960, pp. 57–63, 1960. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, 2016.
22. id\_2\_1\_3\_\_5 M. Moravc ˇ´ık, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, ´ T. Davis, K. Waugh, M. Johanson, and M. Bowling, “DeepStack: Expert-level artificial intelligence in heads-up no-limit poker,” Science, vol. 356, no. 6337, pp. 508–513, jan 2017. [Online]. Available: <http://arxiv.org/abs/1701.01724http://dx.doi.org/10.1126/science.aam6960>
23. id\_2\_1\_3\_\_6 Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, “ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games,” Advances in Neural Information Processing Systems, pp. 2656–2666, jul 2017. [Online]. Available: <http://arxiv.org/abs/1707.01067>
24. id\_2\_1\_3\_\_7 S. Ontanon, “The combinatorial multi-armed bandit problem and its application to real-time strategy games,” in Ninth Artificial Intelligence and Interactive Digital . . ., 2013, pp. 58–64. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377
25. Id\_2\_1\_4\_\_21 [21] I. Fathy, M. Aref, O. Enayet, and A. Al-Ogail, “Intelligent online case-based planning agent model for real-time strategy games,” in Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA’10. IEEE, nov 2010, pp. 445– 450. [Online]. Available: http://ieeexplore.ieee.org/document/5687225/
26. Id\_2\_1\_4\_\_2 [22] U. Jaidee and H. Munoz-Avila, “CLASSQ-L: A Q-Learning ˜ Algorithm for Adversarial Real-Time Strategy Games,” pp. 8–13, 2012. [Online]. Available: <http://www.aaai.org/ocs/index.php/AIIDE/> AIIDE12/paper/viewFile/5515/5734
27. Id\_2\_1\_4\_\_3 [23] B. Lindstrom, I. Selbing, T. Molapour, and A. Olsson, “Racial ¨ Bias Shapes Social Reinforcement Learning,” Psychological Science, vol. 25, no. 3, pp. 711–719, feb 2014. [Online]. Available: <http://arxiv.org/abs/1602.01783>
28. 214 20 24 25 30 31 5 32-49 53-56 59-63
29. Id\_2\_1\_4\_\_20
30. Id\_2\_1\_4\_\_24
31. Id\_2\_1\_4\_\_25
32. Id\_2\_1\_4\_\_30
33. Id\_2\_1\_4\_\_31
34. Id\_2\_1\_4\_\_5
35. Id\_2\_1\_4\_\_32
36. Id\_2\_1\_4\_\_33
37. Id\_2\_1\_4\_\_34
38. Id\_2\_1\_4\_\_33