Оглавление

[1. Введение 4](#_Toc43660824)

[1.1. Недостатки существующих решений и предлагаемые решения 4](#_Toc43660825)

[1.2. Научная новизна 5](#_Toc43660826)

[1.3. Постановка задачи 5](#_Toc43660827)

[2. Аналитическая часть 7](#_Toc43660828)

[2.1. Обзор исследований 7](#_Toc43660829)

[2.1.1. Бот компании OpenAI игры Dota 2 7](#_Toc43660830)

[2.1.2. Бот компании OpenAI играющий в сумо и футбол 7](#_Toc43660831)

[2.1.3. Игровая среда Deep RTS 8](#_Toc43660832)

[2.1.4. Управление армией в StarCraft 12](#_Toc43660833)

[2.1.4.1. Представление состояния игры 14](#_Toc43660834)

[2.1.4.2. Определение действий 16](#_Toc43660835)

[2.1.4.3. Архитектура нейросети 17](#_Toc43660836)

[2.1.4.4. Метод обучения армии 17](#_Toc43660837)

[2.1.4.5. Многоагентный градиентный-спуск Сарса (λ)с разделением параметров 18](#_Toc43660838)

[2.1.4.6. Функция вознаграждения 19](#_Toc43660839)

[2.1.4.7. Пропуск кадров 20](#_Toc43660840)

[2.1.4.8. Эксперименты и их результаты 20](#_Toc43660841)

[2.1.4.9. Недостатки работы 23](#_Toc43660842)

[2.1.5. Эволюционный алгоритм поиска дерева Монте-Карло для игры Hero Academy 25](#_Toc43660843)

[2.1.5.1. Метод поиска дерева Монте-Карло 25](#_Toc43660844)

[2.1.5.2. Эволюционное онлайн планирование 27](#_Toc43660845)

[2.1.5.3. Гибриды алгоритмов поиска по дереву и эволюционных алгоритмов 28](#_Toc43660846)

[2.1.5.4. Тестовая площадка «Hero Academy» 29](#_Toc43660847)

[2.1.5.5. Существующие подходы 30](#_Toc43660848)

[2.1.5.6. Evolutionary MCTS 32](#_Toc43660849)

[2.1.5.7. Эксперименты и результаты 34](#_Toc43660850)

[2.1.6. AlphaStar – полноценный ИИ для StarCraft II 36](#_Toc43660851)

[2.1.6.1. Как происходит обучение 36](#_Toc43660852)

[2.1.6.2. Как AlphaStar действует и видит игру 38](#_Toc43660853)

[2.1.6.3. Итоги игры AlphaStar против профессиональных игроков 39](#_Toc43660854)

[2.1.6.4. AlphaStar и другие сложные проблемы 40](#_Toc43660855)

[3. Конструкторская часть 41](#_Toc43660856)

[3.1. Структура проекта 41](#_Toc43660857)

[3.2. Обновление игры «WarOnMap» 42](#_Toc43660858)

[3.2.1. Создание ИИ-игрока 42](#_Toc43660859)

[3.2.2. Тиковый таймер 43](#_Toc43660860)

[3.3. ИИ-сервер 44](#_Toc43660861)

[3.4. Модельный граф 49](#_Toc43660862)

[3.4.1. Место графа в игре WarOnMap 49](#_Toc43660863)

[3.4.2. Как генерируется карта дорог 51](#_Toc43660864)

[3.4.3. Поиск пути 51](#_Toc43660865)

[3.5. Скриптовый бот 53](#_Toc43660866)

[3.6. Нейросеть 54](#_Toc43660873)

[3.6.1. Общая структура 54](#_Toc43660892)

[3.6.2. Представление состояния игры 55](#_Toc43660912)

[3.6.3. Слой стратегий 58](#_Toc43660923)

[3.6.4. Слой выбора стратегий 59](#_Toc43660924)

[3.6.5. Метод обучения 59](#_Toc43660938)

[3.6.6. Функция вознаграждения 60](#_Toc43660939)

[4. Эксперименты и итоги 63](#_Toc43660954)

[4.1. Процесс обучения 63](#_Toc43660957)

[4.2. Результаты 63](#_Toc43660958)

[5. Заключение 66](#_Toc43660959)

[6. Список литературы 67](#_Toc43660960)

# Введение

Игровой искусственный интеллект с приемлемым уровнем сложности имеет большое значение в современных военных стратегиях, так как игроку не всегда удаётся найти другого игрока соперника.

Разработка ИИ способного управлять армией, состоящей из разных видов войск передвигающихся по графу реальных дорог с учётом множества условий(рельеф местности, ландшафт, время суток и время года, тактико-технические характеристики техники и вооружений, снабжение по дорогам и т. д), и нанести поражение игроку-человеку без численного перевеса является сложной задачей. Под графом дорог подразумевается граф автомобильных дорог. Так как в рамках исследования рассматриваются только автомобильные дороги, поэтому железные дороги не рассматриваются. ИИ можно создать на основе строго заданных алгоритмов, но к такому виду ИИ гораздо проще подобрать стратегию, с помощью которой можно всегда выигрывать, такие стратегии называются доминантными.

## 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). Под графом «шахматная доска» подразумевается граф, у которого в качестве вершин клетки, а в качестве ребёр переходы между клетками как на шахматной доске. 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.

Чтобы решить проблему доминантных стратегий и строго заданного поведения, необходимо использовать нейронную сеть. Реализация игрового интеллекта на базе такой технологии в теории позволит обучить программу не только на играх с человеком (что очень медленно), а также на множестве ускоренных по времени играх с экземпляром другого ИИ(который может быть копией текущего) или тестировочным скриптовым ботом. 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.

## Научная новизна

Разрабатываемая нейросеть является одним из первых известных применений технологий машинного обучения для решения задачи управления разнородной виртуальной армией в стратегии реального времени на графе дорог. Результаты обучения 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.

## 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), уничтожив все войска противника или базы. Для этого она будет управлять подразделениями следующих типов: пехота, артиллерия, мотострелковые подразделения, танки. У каждого их них имеется запас сил, боеприпасов, еды, 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. В игре существуют поезда, которые могут доставлять подразделения или грузы, но так как они передвигаются по железным дорогам, то ими нейросеть управлять не будет, так как было выше сказано, что железные дороги в область исследования не входят.

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» и настраиваемый игровой таймер, который позволит ускорить обучение нейросети за счёт ускорения течения времени в игре
3. Создать ИИ-сервер, который будет создавать, и управлять экземплярами нейросетей, которые отправляют команды игре по сокетам. Пояснение: именно ИИ-сервер, потому-что игра многопользовательская, следовательно необходимо создавать несколько параллельно работающих экземпляров ИИ, каждый из которых будет взаимодействовать со своей игрой.
4. Создать модельный граф. Модельный граф – случайно генерируемая карта дорог, на которой будет обучаться нейросеть. Этот граф будет значительно меньше реального, и он будет находиться в памяти, а не в файле базы данных, что значительно ускорит поиск пути. Случайная генерация позволит избежать проблемы переобучения.
5. Создать скриптового бота, который будет генерировать случайные команды со случайными параметрами. На нём будет обучаться нейросеть. Так как применение технологий машинного обучения в задачах на дорожных графах плохо изучена, то нужны измеряемые результаты, по которым можно будет сказать об эффективности технологии в изучаемой области. Чтобы измерить результат обучения, нейросеть нужно обучить на противнике с постоянным поведением, то есть противник должен быть не нейросетью, поэтому в качестве такого противника был выбран скриптовый бот.
6. Спроектировать нейросеть, которой на вход будет подаваться список подразделений, на выходе команды для игрового сервера, которые будут применены к подразделениям.
7. Создать нейросеть
8. Обучить нейросеть на скриптовом боте.
9. Проанализировать результаты обучения
10. Оценить возможности «обучения с подкреплением» в других задачах на дорожном графе (не относящихся к моей теме)

# Аналитическая часть

## Обзор исследований

В рамках ВКР был проведён обзор последних исследований(2018-2019 г.) в применении машинного обучения в играх, в том числе и игр жанра стратегий реального времени.

### Бот компании OpenAI игры Dota 2

Компания OpenAI в 2019 году создала бота для игры Dota 2. Представленный бот обучался с нуля на своей копии , без имитационного обучения и поиска по дереву. [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».[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 своего противника, благодаря чему смог обыграть профессиональных игроков.

### 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.[3]

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.

### Игровая среда Deep RTS

В статье Per-Arne Andersen, Morten Goodwin, Ole-Christoffer Granmo[4] рассматривается, разработанная ими, игровая среда Deep RTS для изучения алгоритмов обучения с подкреплением.

Также в статье проводится обзор сред для исследования технологий машинного обучения. Были рассмотрены следующие среды(в скобках будет указана игра, на которой тестировались технологии машинного обучения):

* 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 (сокр. ELF) – первая из сред которая предназначена для игр жанра стратегий в реальном времени.
* StarCraft II Learning Environment(сокр. 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[5, 6]. 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 [7]. 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 своей копии .
  + 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 для оценки выходных значений и политики.[8]
* 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)[9]
* Бот для Dota 2. Он рассматривался ранее (см. главу «2.1.1 Бот компании OpenAI игры Dota 2»)

Игровая среда Deep RTS позволяет изучать алгоритмы машинного обучения на разных уровнях сложности в планировании, реагировании 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. Представленный алгоритм не подходит для графа дорог, так как правила «отсечения» из алгоритма применимы только для графа вида «шахматная доска».

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:

1. Первый уровень - это действия, которые непосредственно влияют на среду, например, щелчок правой кнопкой мыши, щелчок левой кнопкой мыши, перемещение влево или выбор юнита.
2. 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[9] и microRTS 11500[10]. 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.

### Управление армией в StarCraft

Kun Shao, Yuanheng Zhu, Dongbin Zhao, Senior Member[11] использовали обучение с подкреплением для управления армией в игре Starcraft. Разработанный ИИ управлял группой одинаковых юнитов на определённой местности.

Также 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 веса нейросети на основе части последовательности, не дожидаясь конца игры[18]. 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 [17].
2. Обучение с подкреплением с глубокой нейронно-сетевой(deep neural networks, DRL) функцией апроксиматора. DRL предоставляет возможность обучить одного агента для решения ряда человеческих задач сквозным способом(end-to-end manner) [19] [20]. 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 [15]. За последние несколько лет появилось множество улучшений DQN, включая двойное DQN [22], приоритетное DQN [23], дуэльное DQN [24], распределенное DQN [25] и асинхронное DQN [26].

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 [27]. Глубокий детерминированный градиент политики (Deep deterministic policy gradient, DDPG) - это непрерывный аналог DQN, который использует критику для оценки ценности текущей политики и актёра(actor) для обновления политики [28]. Методы DRL на основе политик играют важную роль в непрерывном контроле, включая асинхронно преимущественную критику актёра (asynchronous advantage actor-critic (AAAC или A3C)) [26], оптимизацию политики в области доверия (trust region policy optimization , TRPO) [29], оптимизацию проксимальной политики (proximal policy optimization, PPO) [30] 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 политики и алгоритм RL генерирует руководящие распределения(guiding distributions), что позволяет эффективно обучать глубоким политикам [31]. Исследователи также предложили некоторые другие основанные на модели методы DRL, такие как нормализованные функции (normalized advantage functions, NAF) [32] и встраивание в контроль (embed to control, E2C) [33].

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 из собственного наблюдения за текущим боем, рассматривая других юнитов как часть среды. Что касается много-агентной кооперации, эта политика распространяется среди всех юнитов. Цель каждого юнита - максимизировать его общий ожидаемый результат.

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 однотипных агентов более эффективно [38].

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[39]. Авторы статьи приняли решение использовать комбинацию двух предыдущих методов под названием плановое трансферное обучение(curriculum transfer learning , CTL), который хорошо себя показал на практике[40], [41], [42]

#### Представление состояния игры

Представление состояния игры в StarCraft в удобном и компактном виде остается открытой проблемой без универсального решения. Для решения этой проблемы авторы статьи предлагают собственный метод решения. Предложенный метод представления состояния эффективен и не зависит от количества юнитов в бою. Согласно этому методу представление состояния состоит из трех частей: информация о состоянии текущего шага, информация о состоянии последнего шага и действия последнего шага.(см. рис. 1)



Рисунок 2.1.1 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

Информация о текущем шаге состояния включает в себя (см. таблицу 1):

* время перезарядки оружия(CoolDown),
* количество очков здоровья(HitPoint),
* информацию о расстояниях до своих юнитов(OwnSumInfo, OwnMaxInfo)
* информацию о расстоянии до вражеских юнитов(EnemySumInfo, EnemyMaxInfo)
* информацию о расстоянии объектов местности(TerrarianInfo)



Таблица 2.1.4.1-1 Типы данных и размерность модели. R – число с плавающей запятой, cat – категория действия, является одним из 8 направлений движения или атакой

Информация о состоянии последнего шага совпадает с текущим шагом(если обучение только началось). К сведению принимается последнее действие, которое оказалось полезным для процесса обучения.

Авторы делят боевую карту на 8 секторов в среднем и вычисляют информацию о расстояниях в каждой области. Все входы с типом «число с плавающей точкой» нормированы по их максимальным значениям, что позволяет обобщать данные о юнитах в секторах и сократить количество используемых параметров. CoolDown и HitPoint имеют одну степень свободы (может только уменьшаться). 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.

Таким образом, информация о состоянии текущего шага имеет 42 измерения(см рисунок 1 и таблицу 1). Действие последнего шага имеет 9 измерений, причем для выбранного действия задано значение 1, а для других действий - 0. В результате представление состояния игры имеет 93 измерения(42 для текущего шага + 42 предыдущего + стоимости 9-ти действий).

#### Определение действий

В сценариях микроуправления StarCraft пространство действий очень велико. На каждом временном шаге юнит может перемещаться в произвольных направлениях и с произвольными расстояниями на карте, во время атаки, 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 цель с самым низким уровнем здоровья в радиусе действия оружия. Когда выбрано передвижение в определённом направлении, юнит повернётся в одно из 8 направлений и переместится на фиксированное расстояние. Когда выбрана атака, юнит остановится и будет атаковать врагов. Согласно результатам эксперимента, этих действий достаточно, чтобы эффективно управлять юнитами в игре.

#### Архитектура нейросети

Опыт юнитов включает в себя ограниченное подмножество большого пространства состояний, и большинство тестовых состояний никогда не будет исследовано заранее, поэтому будет трудно применить таблицу обучения с подкреплением для изучения оптимальной политики. Чтобы решить эту проблему, авторы статьи используют нейронную сеть, параметризованную вектором θ, для аппроксимации значений действие-состояние(state-action values) для улучшения обобщения RL-модели.

Входные данные сети - это 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)

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 [43]. В отличие от этих насыщенных функций нелинейности(saturating nonlinearities function), таких как sigmoid или tanh, функция ReLU является ненасыщенной нелинейной функцией. С точки зрения времени обучения с градиентным спуском ненасыщенная нелинейная намного быстрее [44].

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 рисунке 1.

#### Метод обучения армии

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, с использованием функцией вознаграждения в качестве внутренней мотивации для продвижения процесса обучения. Вся схема PS-MAGDS обучения с подкреплением изображена на рис. 2.



Рисунок 2.1.2 Диаграмма обучения с подкреплением PS-MAGDS в сценариях микроуправления StarCraft.

#### Многоагентный градиентный-спуск Сарса (λ)с разделением параметров

Сарса(Sarsa)– это RL-алгоритм с обучением с разницей во времени(temporal difference (TD) learning). Он основан на политике, которая используется как для выбора действий, так и для обновления предыдущего значения.

Авторы предлагают RL-алгоритм с несколькими агентами, который расширяет традиционный Сарса алгоритм (λ) до нескольких юнитов, 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 [45].

Алгоритм учитывает не только значение последней пары состояние-действие, но и уже посещенных. С помощью этого метода решается проблема отсроченного вознаграждения. Сарса с приемлемыми чертами, называемый 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 Сарса (λ) для битв нескольких юнитов, используется нейронная сеть в качестве аппроксиматора функции, и параметры сети разделяются между всеми юнитами. Несмотря на то, что есть только одна сеть для обучения, юниты могут вести себя по-разному, потому что каждый юнит получает разные наблюдения и действия в качестве входных данных.

Одним из сложных вопросов в обучении с подкреплением является компромисс между разведкой и эксплуатацией. Для решения этого компромисса используется метод ε-жадный для выбора действий во время тренировки, который выбирает текущее наилучшее действие с вероятностью 1 - ε и выполняет случайное действие с вероятностью ε.

#### Функция вознаграждения

Авторы статьи для решения проблемы редких и отсроченных вознаграждений разработали функцию вознаграждения, включающую небольшие промежуточные вознаграждения. В экспериментах все агенты получали основное вознаграждение при атаке на каждом временном шаге, равное урону, который получили вражеские юниты минус количество потерянного здоровья своих юнитов:

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

где *damage\_amount* - это количество урона, нанесенного своими юнитами, *damage\_factor* - сила атаки своих юнитов, *unit hitpoint* — это количество очков здоровья своих юнитов. Вознаграждение делится на константу, чтобы вписать его в более подходящий диапазон значений, который в эксперименте равен 10. *ρ* является нормализованным фактором для баланса общего количества очков здоровья наших юнитов и юнитов противника,



где H - количество вражеских юнитов, а N - количество наших юнитов. Этот нормализованный фактор необходим в микроуправлении StarCraft для различного количества и типов юнитов. Без надлежащей нормализации, политика сети будет хуже сходиться, и юнитам нужно будет гораздо больше времени для изучения полезного поведения.

Помимо основной награды за атаку, рассматриваются дополнительные награды. Например, когда юнит уничтожается, даётся отрицательное вознаграждение. Для того чтобы побудить юнитов к совместным действиям, были введены вознаграждение за их перемещение. Если в направлении движения нет своих юнитов или вражеских юнитов, то этому движению присваивается небольшое отрицательное вознаграждение. Согласно эксперименту, это вознаграждение оказывает впечатляющее влияние на эффективность обучения, как показано на рис. 3.



Рисунок 2.1.3 Процент побед подразделений в сценарии «3 голиафа против 6 зилотов» через каждые 200 тренировок. moveReward – награда за перемещение.

#### Пропуск кадров

В StarCraft нецелесообразно совершать действия в каждом игровом кадре, так как это будет сильно нагружать компьютер. Авторы статьи решили использовать технологию пропуска кадров, которая выполняет этап обучения через фиксированное число кадров. В процессе была выбрана частота 10 кадров.

#### 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.

Было 3 сценария:

1. Голиафы(юниты дальнего боя с большим запасом здоровья) против Зилотов(юниты ближнего боя с большим уровнем здоровья и наносимого урона)
2. Голиафы против Зерглингов(юниты ближнего боя с низким уровнем здоровья и наносимого урона, но быстрой скоростью атаки)
3. Морпехи(юниты дальнего боя с небольшим запасом здоровья) против Зерглингов

В каждом сценарии было 3 стадии, в первой стадии было небольшое количество юнитов у нейросети и противника. В следующих стадиях количество юнитов у той и другой стороны было больше.

После обучения нейросети первому сценарию, авторы решили проверить эффективность трансферного обучения и создали две нейронные сети, одна из которых с нуля обучалась сценарию «Голиафы против Зерглингов», другая была копией сети обученной на сценарии «Голиафы против Зилотов». В итоге получилось, что без трансферного обучения нейросеть обучается значительно медленнее (см рисунок 4).



Рисунок 2.1.4 Процент побед юнитов в сценарии «3 голиафа против 20 зерглингов» через каждые 200 тренировок.

. Кроме того авторы решили сравнить PS-MAGDS с базовыми методами, которые состоят из подходов основанных на правилах и подходов DRL:

* Слабейший: метод, основанный на правиле, атаковать слабейших в радиусе атаки.
* Ближайший: метод, основанный на правиле, атаковать ближайшего в радиусе действия оружия.
* GMEZO: метод DRL, основанный на оптимизации нулевого порядка, имеющий впечатляющие результаты по сравнению с традиционными RL-методами [66].
* BicNet: метод DRL, основанный на архитектуре актер-критика, имеющий наилучшую производительность в большинстве сценариев микроуправления StarCraft [67].

В таблице 2 показан процент побед метода PS-MAGDS в сравнении с базовыми методами. В каждом сценарии 5 раз измерялся средний показатель побед модели в 100 тестовых играх.



Таблица 2.1.4.8-1 Сравнение эффективности модели с базовыми методами в двух сценариях «Морпехи против зерглингов»

В результате обучения агенты освоили следующие стратегии:

1. Разделение врагов. Суть в том чтобы выманивать противников по одному и сражаться только с одним противником.
2. Сохранение группы. Организация в группы позволяет эффективно бороться с противниками, это выглядит как движение группы в одном направлении и атака на одну и ту же цель, как показано на рисунке 5.
3. Бей и беги. Широко используемая игроками стратегия. Суть её в том чтобы сначала атаковать и пока оружие перезаряжается, бежать от противника.



Рисунок 2.1.5 Пример игры в сценарии «20 морпехов против 30 зерглингов». Морпехи красного цвета слева, Зерглинги синего цвета справа. Белые линии показывают направление движения, красные – направление атаки.

Существующие проблемы:

1. Юниты время от времени двигаются вперед и назад и не участвуют в битвах, чтобы вовремя помогать другим юнитам.
2. Юниты предпочитают перемещаться к границе карты, чтобы избежать врагов.

#### Недостатки работы

Авторы рассматриваемого исследования обучали нейросеть управлению группой однотипных юнитов, кроме того они могут обучать только наземных юнитов дальнего боя, в то время как тренировка наземных юнитов ближнего боя с использованием методов RL остается открытой проблемой.

Также авторы подчеркивают, что есть и 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 [47]. По сравнению с методом формирования вознаграждения, иерархический RL обладает способностью изучать абстрагированное от времени исследование, и дает агентам большую гибкость. Но его структура также намного сложнее, и автоматическое извлечение подзадач остается открытой проблемой, поэтому было принято решение использовать более простой метод.

Карты в Starcraft не являются дорожным графом, поэтому рассматриваемая нейросеть не может с ним работать.

### Эволюционный алгоритм поиска дерева Монте-Карло для игры Hero Academy

Hendrik Baier, Peter I. Cowling в своей работе [48] предлагают новый метод Эволюционный алгоритм поиска дерева Монте-Карло(Evolutionary Monte Carlo Tree Search) и сравнивают его с другими известными алгоритмами, которые применяются в пошаговых стратегиях.

Новый алгоритм объединяет в себе древовидный поиск по дереву и эволюционные алгоритмы. Поэтому сначала были рассмотрены существующие решения: метод поиска дерева Монте-Карло, эволюционное онлайн планирование и несколько гибридных алгоритмов.

#### Метод поиска дерева Монте-Карло

Monte Carlo Tree Search (MCTS) [49, 50] является первым алгоритмом поиска дерева решений на основе стохастических симуляций для оценки состояния, который был успешно применён в большом множестве игр и других задач [52]. Алгоритм, как правило, строит дерево поиска с узлами, представляющими состояния игры, и рёбрами, представляющими действия, ведущие от одного состояния до другого. MCTS начинает свой поиск с корневого узла, соответствующий текущему состоянию игры. Алгоритм работает несколько этапов, пока время вычислений не закончится, в каждом этапе четыре шага:

1. В фазе выбора, **политика выбора** используется, чтобы пройти по дереву, 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 [56].
2. На этапе **расширения** ранее нерасширенное действие и узел, представляющий его преемника, добавляются в дерево.
3. На этапе развёртывания, **политика развёртывания(rollout policy)**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 авторы используют 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 первого хода(First-play urgency) [57] 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 [58] и расщепление [59] 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(script-based approaches) разработаны для поиска небольшого числа самодельных скриптов(hand-coded scripts) вместо большего числа элементарных действий: Поиск по иерархическому портфолио[60] и основанный на скриптах UCT [61] попадают в эту категорию, а также не-MCTS подход Portfolio Greedy Search [62]. 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 [63] или сходство «близких» действий в среде [64].

Часто делается предположение, что каждый юнит может выполнять одно действие за временной шаг, как это типично для игр RTS. В этой статье авторы не предполагают независимость юнитов, не связывают действия с юнитами(так как делается несколько действий за ход и действия могут применяться несколько раз к разным юнитам, о чём будет сказано далее) и не предполагают существование предопределенных политик или сценариев. Тем не менее, используется эвристическая функцию оценки, которая написана вручную в их тестовой области, но в будущем она может быть автоматически изучена [51].

#### Эволюционное онлайн планирование

Эволюционные алгоритмы (Evolutionary algorithms, EAs) - это класс алгоритмов оптимизации, основанный на естественном отборе, который широко использовался для развития и обучения агентов ИИ для игр [65], [66].

Онлайн эволюция - это новый подход, в котором эволюционные алгоритмы применяются во время игры. Это может выглядеть как изменение параметров ИИ во время игры [71]. Тем не менее, возможно улучшить следующее действие в текущей запущенной игре. Эволюционный алгоритм со скользящим горизонтом(Rolling Horizon Evolutionary Algorithm*,* RHEA) [72], [73], например, улучшает будущие последовательности действий фиксированной длины в однопользовательской игре, которые сравниваются путем их моделирования и оценки результирующих игровых состояний. При окончании времени поиска алгоритм выполняет первое наилучшее действие в найденной последовательности и продолжает поиск последовательностей действий, начиная со следующего временного шага («скользящий»(rolling) горизонт поиска).  
 Эволюционное онлайн планирование (OEP) [74], [53] - это 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 «скольжения» горизонта от действия к действию.

OEP начинает свой поиск с создания начальной популяции геномов, каждый из которых представляет законченный ход(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 (совокупность внешних и внутренних признаков организма, приобретённых в результате индивидуального развития), игровые состояния возникают в результате применения 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(kill rate*)*.
3. Каждый из выживших геномов соединяется со случайно выбранным различным геномом и создает потомство посредством равномерного кроссовера(обмен хромосомами). Если после кроссовера появляется недопустимое действие в потомстве, геном восстанавливается заменой действием другого родителя или иным образом случайным корректным действием.
4. Доля потомства, определяемая параметром скорость мутации(mutation rate), 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 для следующих действий. По словам Wang «Проблема выбора действий рассматривается скорее как проблема оптимизации, а не как проблема планирования» [75]. На момент написания статьи это наилучший подход к пошаговым многопользовательским состязательным играм, в частности к тестовой области данной статьи: Hero Academy [53]. Он также был применен к другим проблемам, таким как микробитвы [75] или онлайн адаптация порядка действий [76] в RTS играх.

#### Гибриды алгоритмов поиска по дереву и эволюционных алгоритмов

Gaina и соавт. [77] 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. [78] 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. Lucas и соавт. [79] использовали эволюционный алгоритм для улучшения политики развертывания MCTS во время поиска. Perez-Liebana и соавт. [80] 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 и соавт. [81] предложили стратегию эволюции путей(strategy to evolve paths) 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 Hero Academy, он дает интересное направление на возможную будущую работу, которая может позволить учитывать действия противника.

#### Тестовая площадка «Hero Academy»

Представленный метод тестировали на упрощенном Java-клоне[82] «Hero Academy» [83]. Hero Academy - пошаговая тактическая стратегия для двух игроков. В распоряжении игроков различные боевые юниты, предметы и заклинания, которые сначала достаются их из колоды карт, а затем применяются на карте, состоящей из клеток. В эти клетки можно поставить юнитов, есть также специальные клетки, которые повышают характеристики отдельных юнитов или в них расположены кристаллы игроков (их всего два). Цель игры: уничтожить всех войска врага или два кристалла врага.



Рисунок 2.1.6 Скриншот игры Hero Academy. Круг с пятью синими треугольниками это очки действий, слева от него карты. Цифры под дверями - размеры колод. Один из кристаллов красного игрока уже уничтожен.

В игре есть очки действий, по 5 на каждый ход. Они расходуются на перемещение юнитов, атаку юнитов врага и другие действия. Можно потратить несколько очков на одного юнита, например, переместив его на несколько клеток вперёд. Порядок карт в колоде и сами карты противника неизвестны игроку и наоборот.

#### Существующие подходы

**Жадное Действие(Greedy Action)**. Жадный поиск выбирает первое действие своего хода в процессе простого однослойного поиска всех корректных действий, максимизируя эвристическую оценку в немедленно возникающем состоянии. Это повторяется для каждого очка действия из пяти, дающихся на один ход.

**Жадный ход**. ИИ с жадным ходом выбирает последовательность из пяти ходов с максимальным выигрышем. 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**. Является первым вариантом MCTS, адаптированным для многопользовательских состязательных игр в [53]. Он ищет игровое дерево, как показано на рисунке 2, в котором каждое ребро представляет дополнительное действие для рассматриваемого хода (или его применения). Это вариант MCTS использует политику выбора UCB[56], и политика развертывания следует эвристике упорядочивания действий. Следующий ход противника может быть изучен деревом с количеством очков больше пяти. Было обнаружено, что производительность улучшается, если брать последовательность с количеством шагов, которое близко к числу очков действий в ходе. Коэффициент разведки C = 0 для того чтобы вырастить достаточно глубокое дерево (что является «чистой эксплуатацией»).



Рисунок 2.1.7 Дерево поиска в классическом MCTS и ее вариантов (не исследующий MCTS, BB-MCTS). Узлы представляют собой частичные последовательности действий или состояния, возникающие из них. Ребра представляют собой добавление атомарного действия к последовательности или применение атомарного действия к состоянию. После каждого расширения узла выполняется оценки последовательности.(символы используются для представления различных атомарных действий)

**MCTS “Сжигание моста” (Bridge-burning MCTS, BB-MCTS).** Этот вариант MCTS ищет такое же дерево как на рисунке 2. Вместо проверки после каждого расширения он использует «ε-жадную прокатку»(ε-greedy rollouts) с коэффициентом исследования С = 0.5, которая проходит только в конце текущего хода из конечного узла. Для того чтобы вырастить достаточно глубокое дерево для ходов с несколькими действиями, авторы используют технику, которая называется «сжигание моста» в [53]. Данная техника является переизобретением поиска по шагам [54]. Авторы используют здесь термин «сжигание мостов», так как термин «движение» в “Hero Academy” неоднозначен, а также потому, что они собираются обобщить концепцию “сжигания мостов” для другого вида дерева(о нём будет сказано 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 выгодное действие из корня, ведущее к корневому состоянию для следующей фазы. Это может быть реализовано как стратегия жесткого сокращения, показанная на рисунке 3.



Рисунок 2.1.8 Стратегия поиска «сжигания моста» (иллюстрация адаптирована из [53]). (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. Часть дерева из лучшей ветви сохраняется.

**Эволюционное онлайн-планирование(Online Evolutionary Planning, OEP)**. Этот алгоритм в настоящее время является наиболее эффективным подходом для многошаговых пошаговых игр, таких как “ Hero Academy ”. Хотя [53] показывает, что он имеет силу, аналогичную не исследующему MCTS и BB-MCTS в стандартной форме игры с 5 очками действия за ход, было показано, что OEP лучше масштабируется для более сложных задач “ Hero Academy ” с использованием 10 AP или больше.

#### Evolutionary MCTS

Evolutionary MCTS или EMCTS объединяет поиск по дереву MCTS с подходом эволюционных алгоритмов на основе генома.



Рисунок 2.1.9 Дерево EMCTS. Узлы представляют собой последовательности завершённых действий (геномы) или состояния, возникающие из них. Рёбра представляют собой мутацию простейшего действия в геноме. Ремонт может быть необходимым, если эти мутации могут привести к некорректным геномам. После каждого расширения узла вызывается функция оценки вместо прокатки.

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 узлах(Оценка в конечных узлах - это хорошо известный вариант MCTS, который был успешно использован, например, в AlphaGo Zero и AlphaZero. [51].). Обратное распространение без изменений.

Для представленного алгоритма нужна стартовая последовательность. Возможны разные подходы для её создания - в этой статье использовался алгоритм «жадное действие», описанный ранее. Мутации могут приводить к некорректной последовательности. Чтобы решить эту проблему применяется классический эволюционный алгоритм с использованием стратегии восстановления - в этой статье используется ИИ с «жадным действием» также и для ремонта, когда это необходимо.

Использование «жадного действия» не вводит дополнительных эвристических знаний, так как все алгоритмы, сравниваемые в этой статье, работают с одной и той же функцией оценки. Тем не менее авторы отмечают, что OEP, как и EMCTS, можно значительно улучшить, **используя политику восстановления «жадное действие» вместо политики случайного восстановления**. Это приводит к более высокому качеству ремонта в среднем случае. И точно так же, как EMCTS получает улучшение от генома жадного корня(стартовой последовательности, сгенерированной «жадным действием»), OEP может получить прибыль от заполнения 20% начальной популяции последовательностями «жадного действия» вместо случайных. Это запускает поиск с более качественным стартовым решениям. Авторы называют этот новый вариант жадным OEP, в отличие от стандартного OEP со случайным ремонтом и чисто случайным начальным заполнением, как описано в [74], [55], [53], и включают его в эксперименты для сравнения.

EMCTS имеет более большой коэффициент ветвления, чем классические варианты MCTS. В то время как коэффициент ветвления в игре “ Hero Academy ” 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 очками действий. Мы обнаружили, что эффективным способом борьбы с этим является «сжигание 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», или последовательных поисков и сокращений или мутаций, является единственным параметром EMCTS, который авторы регулировали в экспериментах. Коэффициент исследования MCTS был установлен на C = 0. Политика выбора - UCB[56]., как и в других вариантах MCTS.

#### Эксперименты и результаты

Так как сложность одного хода увеличивается в геометрической прогрессии, сравнения проводились с разными временными рамками: 200 мс на ход для 5 очков действия, 1 секунда для 10 и 5 секунд для 15. Каждое сравнение состояло из 400 игр, при этом EMCTS играл 200 игр с правом первого хода и 200 игр после хода противника. Игры, в которых не было победителя после 200 ходов, учитывались как ничьи, то есть половина выигрыша для каждого игрока. 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 соответственно. Результаты сравнения показаны в таблице 1.



Таблица 2.1.5.7-1 Процент выигранных EMCTS сражений против других алгоритмов. Звездочки указывают на значительное превосходство, р вероятность победы противника: \* р <0,05, \*\* р <0,01, \*\*\* р <0,001

EMCTS значительно сильнее всех базовых алгоритмов во всех бюджетах времени и при любом количестве очков действий на ход. Его относительная сила увеличивается со сложностью задачи поиска, измеряемой в очках действий на ход. Недавно предложенный «жадный OEP» на практике оказался лучше классического OEP, как описано в [53], но все еще значительно слабее, чем EMCTS при всех выбранных количествах очков действий при 200 мс на ход, и при всех выбранных количествах очков действий, за исключением самого низкого количества очков действий в 1 с и 5 с на ход, где оба алгоритма работают одинаково. Таким образом, результаты показывают, что Эволюционный MCTS очень эффективен при различных бюджетах времени и лучше масштабируется в зависимости от сложности предметной области, чем все другие проверенные подходы.

### AlphaStar – полноценный ИИ для StarCraft II

AlphaStar — первая нейросеть, которая способна победить профессиональных игроков в StarCraft II. В серии матчей, которые состоялись 19 декабря, AlphaStar одержал уверенную победу над Grzegorz Komincz (MaNa) из команды Liquid, одного из сильнейших игроков в мире, со счетом 5:0[84]. 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.

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 выдает последовательность действий. При создании архитектуры нейронной сети применялся 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(уровень среднего игрока), 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 это делают люди, играя на турнирах. Новые соперники добавлялись в лигу путем дублирования текущих агентов. Такая форма обучения, с использованием некоторых идей из метода обучения с подкреплением с элементами генетических алгоритмов, позволяет создать непрерывный процесс исследования огромного пространства стратегий StarCraft'а, при котором агенты могут противостоять более сильным стратегиям и не забывать 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. 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[85]. Другими словами, самую эффективную комбинацию стратегий, которые были получены в процессе обучения. 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[86], 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 2.1.10 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 интерфейс, из-за чего он мог видеть атрибуты своих и видимых вражеских юнитов на карте напрямую, без необходимости двигать камеру. 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 карты.

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. Наработки AlphaStar позволят в дальнейшем решать задачи составления сложных прогнозов для очень длинных последовательностей данных, таких как прогноз погоды, моделирование климата, понимание языка и др.

Создатели AlphaStar также рассчитывают на то, 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.

# Конструкторская часть

## Структура проекта

Ранее в пункте 1.3 было сказано о том, что нужно сделать для выполнения поставленной цели:

1. Обновить модуль, создающий игроков, так чтобы в игре «WarOnMap» мог участвовать ИИ, и добавить настраиваемый игровой таймер, подобный тиковому таймеру из пункта «2.1.3 Игровая среда Deep RTS». Представленный модуль находится на игровом сервере, который написан на языке JavaScript.
2. ИИ-сервер, который будет создавать, и управлять экземплярами нейросетей, которые отправляют команды игре по сокетам. Пояснение: именно ИИ-сервер, потому что игра многопользовательская, следовательно необходимо создавать несколько параллельно работающих экземпляров ИИ, каждый из которых будет взаимодействовать со своей игрой. В названии присутствует слово «ИИ», а не «нейросеть», потому что сервер будет создавать не только нейросети, но и тестировочных скриптовых ботов. Используемые технологии: язык Python, Python-библиотека и веб-фрэймворк Flask
3. Модельный граф – случайно генерируемая карта дорог, на которой будет обучаться нейросеть. Этот граф будет значительно меньше реального, и он будет находиться в памяти, а не в файле базы данных, что значительно ускорит поиск пути. Случайная генерация позволит избежать проблемы переобучения. Используемые технологии: язык JasvaScript. Модельный граф является модулем для игры, находится на игровом сервере.
4. Скриптовый бот. Генерирует случайные команды со случайными параметрами. Нейросеть нужно обучить на противнике с постоянным поведением, чтобы наглядно показать прогресс в обучении. Используемые технологии: язык Python. Находится на ИИ-сервере.
5. Нейросеть, которой на вход будет подаваться список подразделений, на выходе команды для игрового сервера, которые будут применены к подразделениям. Она будет находиться на ИИ-сервере. Используемые технологии: язык Python, Tensorflow.

На рисунке 3.1.1 показано устройство системы, для которой разрабатывается нейросеть.



Рисунок 3.1.1 Диаграмма контейнеров игры WarOnMap

## Обновление игры «WarOnMap»

### Создание ИИ-игрока

Процесс создания состоит из следующих шагов:

1. Игроку человеку нужно при создании игры указать, что один из игроков будет ИИ. При этом можно выбрать один из предложенных ИИ, у каждого будет своё название (см рисунок 3.2.1). В рамках ВКР разрабатывается только одна нейросеть, несмотря на это предоставляется список с расчётом на то, что в дальнейшем разработчики WarOnMap добавят другие нейросети или скриптовых ботов. Возможно, в этом списке будут разные версии разрабатываемой нейросети.
2. Далее игрок нажимает на кнопку «Create game», после «Клиент» отправляет GET запрос на «Игровой сервер».
3. На «Игровом сервере» класс GameManagement создаёт игру, из запроса клиента извлекаются данные о карте. Далее создаётся игра, объявляются игроки, сначала ИИ-игроки, затем игроки-люди. При объявлении ИИ-игрока класс AiConnectionCreator создаёт сокет, который подключается к ИИ-серверу.
4. После подключения класс AiUnitPlacer ждёт позиции юнитов ИИ и размещает их.

В игровом цикле будет отправляться запрос на команды от ИИ по ранее созданному сокету. По сокету отправляется JSON-файл с данными о подразделениях, в ответ приходит JSON с командами от ИИ.



Рисунок 3.2.1 Интерфейс опций выбранной карты

### Тиковый таймер

В игру добавлен класс GameTimer, который подобен тиковому таймеру из пункта «2.1.3 Игровая среда Deep RTS».. Чтобы сделать игру независимой от реального времени, в конструктор таймера нужно передать экземпляр класса TickMode. Конструктор TickMode принимает число равное длительности одного тика в секундах. Тик в WarOnMap равен длительности 1 кадра(по умолчанию 0,04, 25 кадров в секунду). Если нужно сделать привязку к реальному времени, как в режимах «игрок против игрока» или «игрок против ИИ», то необходимо передать в конструктор экземпляра класса RealTimeMode. В режиме реального времени таймер выдаёт реально прошедшее время, а в тиковом режиме время одного тика.

## ИИ-сервер

Перед тем как описать устройство ИИ-сервера, сначала будет рассмотрен жизненный цикл ИИ в виде диаграммы последовательности (см. Рисунок 3.3.1).



Рисунок 3.3.1 Диаграмма последовательности для жизненного цикла ИИ

Представленная выше диаграмма позволит лучше понять устройство ИИ-сервера, так как каждый из компонентов контейнера «ИИ-сервер» выполняет какое-либо действие из диаграммы последовательности. На рисунке 3.3.1 указано только то, что относится непосредственно к ИИ, поэтому не показано где создаются игроки-люди. Как было сказано в пункте 3.2.1, сначала создаются игроки-люди, потом ИИ-игроки.

При создании представленного проекта использовалась концепция C4Model, которая является последовательностью диаграмм разного уровня абстракции (ранее была показана диаграмма контейнеров см. рисунок 3.1.1), поэтому устройство ИИ-сервера будет рассмотрено на двух уровнях: на уровне взаимодействия компонентов ИИ-сервера (диаграмма компонентов рисунок 3.3.2) и на уровне взаимодействия конкретных классов (диаграмма классов см. рисунки 3.3.3 и 3.3.4), из которых состоят компоненты ИИ-сервера.

ИИ-сервер состоит из нескольких компонентов, показанных на Рисунок 3.3.3.



Рисунок 3.3.2 Диаграмма компонентов контейнера ИИ-сервера

Компонент «Менеджер ИИ» будет содержать в себе экземпляры ИИ, у каждого из которых будет свой сокет. В этом же компоненте будет храниться тестовый скриптовый бот (подробнее описан в пункте 3.6) на котором будет обучаться нейросеть.

На следующем рисунке показана диаграмма классов компонента «Сервер» из контейнера «ИИ-сервер».



Рисунок 3.3.3 Диаграмма классов компонента «Сервер» из контейнера «ИИ-сервер».

На представленной выше диаграмме показано устройство только рассматриваемого компонента, устройство «Менеджера ИИ» будет рассмотрено дальше (см. рисунок 3.3.4).

Диаграммы классов могут представлять систему на разных уровнях:

1. на уровне анализа класс содержит в себе только набросок общих контуров системы и работает как логическая концепция предметной области или программного продукта.
2. на уровне проектирования класс отражает основные проектные решения касательно распределения информации и планируемой функциональности, объединяя в себе сведения о состоянии и операциях.
3. на уровне реализации класс дорабатывается до такого вида, в каком он максимально удобен для воплощения в выбранной среде разработки.

На рисунке 3.3.3 и 3.3.4 рассмотрен уровень проектирования. Было решено не создавать диаграмму классов на уровне реализации, так как диаграммы такого уровня сложно поддерживать в актуальном состоянии и на их создание требуется значительно больше времени.

На следующем рисунке показана диаграмма классов компонента «Менеджер ИИ» из контейнера «ИИ-сервер».



Рисунок 3.3.4 Диаграмма классов компонента «Менеджер ИИ» из контейнера «ИИ-сервер»

## Модельный граф

### Место графа в игре WarOnMap

На рисунке 3.4.1.1 показана диаграмма последовательности для модельного графа. Модельный граф создаётся перед созданием сокетов для игроков людей и ИИ-игроков, если выбранная карта является модельным графом.



Рисунок 3.4.1 Диаграмма последовательности для модельного графа

На представленной выше диаграмме в процессе «Настройка остальных компонентов игры» подразумевается последовательность процессов “ «создание соединения» «отправка данных о местности», «ожидание позиции подразделений ИИ», «расстановка подразделений ИИ»” из рисунка 3.3.1.

Чтобы протестировать правильность генерации модельного графа, было решено сделать его доступным для игроков-людей, то есть игрок-человек может сыграть на модельном графе, как на обычной карте, это означает, что модельный граф будет рисоваться вместо реальной карты в клиенте. Далее будет описан процесс выбора карты игроком-человеком.

Игру можно запускать в двух режимах: production-режим и режим разработчика. В production-режиме игрок не может сразиться на модельном графе. Модельный граф доступен только в режиме разработчика, его можно выбрать в качестве карты. Нейросеть будет тестироваться при включенном режиме разработчика у игры.



Рисунок 3.4.2 Экран выбора карты

После того как в качестве карты выбран «Модельный граф», игроку нужно настроить параметры графа: количество узлов (по вертикали и горизонтали), расстояние между узлами в км (минимальное и максимальное).

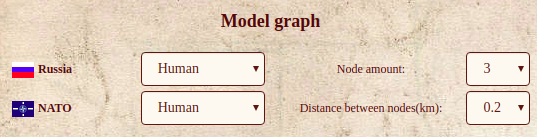


Рисунок 3.4.3 Интерфейс опции «Модельного графа»

При создании игры нужно выбрать место, откуда брать геоданные: геосервер или модельный граф. В игре WarOnMap существует класс MapGeoDataDistributor, который даёт игре геоданные выбранной игроком карты (маршруты, высоту рельефа, погоду). На его основе с таким же интерфейсом(IGeoDataDistributor) создан класс ModelGraphDataDistributor, которому в конструктор передаётся ModelGraph с параметрами от игрока. Когда игрок выбирает модельный граф, то игра будет брать данные от ModelGraphDataDistributor, вместо MapGeoDataDistributor. Подробное описание модельного графа и его генерация описана в следующем разделе .

Модельный граф будет прислан клиенту после создания игры или при перезагрузке страницы. Присланный граф в виде JSON-файла будет преобразован клиентом в список объектов местности, затем компоненты списка будут нарисованы на карте.

Модельный граф не будет генерировать случайные погодные условия и случайную карту высот (овраги, холмы и прочее). Будет статичная погода и ровная поверхность, потому что сейчас важно понять возможности «обучения с подкреплением» в задачах на дорожном графе и для этого не нужно наличие геоданных, а их наличие замедляет обучение .

### Как генерируется карта дорог

Модельный граф хранится в классе ModelGraph. Его генерацией занимается ModelGraphBuider. Используется паттерн проектирования «Строитель», чтобы каждый из шагов генерации можно было проверить, поэтому после выполнения шага возвращается результат. Процесс генерации выглядит следующим образом:

1. В ModelGraphBuider передаются параметры, пришедшие от игрока: количество узлов (по вертикали и горизонтали), расстояние между узлами в км. Генерируется сетка, где дуги – дороги, а узлы – перекрёстки . Длина дуг равна «расстоянию между узлами в км». Сгенерированная сетка имеет отступ от краёв карты равное расстоянию между узлами.
2. Удаляется случайное количество вершин в пределах [1/4; 1/2] всех вершин. Следующая удаляемая вершина должна находиться на расстоянии больше чем «расстояние между узлами»\*1.5, чтобы не получился недоступный узел.
3. Случайно сдвигаются вершины по оси абсцисс и ординат на [-«мин. расстояние между узлами» / 3: +«мин. расстояние между узлами» / 3]
4. Рассчитываются расстояния между вершинами, до этого шага они равны 0. Полученные расстояния используются при поиске путей.

Чтобы не добавлять зависимость от последовательности шагов генерации из класса ModelGraphBuider, сгенерированный граф выдаётся классом ModelGraphBuildingDirector, который имеет метод generateGraph c параметром builder по умолчанию равным экземпляру ModelGraphBuider.

### Поиск пути

Для поиска пути используется алгоритм A\*. При поиске пути игрок-человек или ИИ, указывают конечную точку, а начальная точка будет на месте юнита. Так как эти точки могут находиться не прямо над узлами графа, а рядом, то нужно искать ближайший узел. Для этой цели создан алгоритм быстрого поиска ближайшего узла.

Суть его в том, что карта делится на области и для выбранной точки находится соответствующая область (см. пример, ниже, на ). В каждой из областей лежит от 1 до 4 вершин(так как часть удаляется случайным образом), размер области 2 \* «мин. расстояние между узлами». Координаты области лежат в промежутке [2dn – 0,5d; 2d(n + 1) – 0,5d) для оси абсцисс и ординат, где d - мин. расстояние между узлами, n – индекс сектора по оси абсцисс или ординат, 0.5d – сдвиг области влево, так как наименьшее значение координаты входящей в область равно -0,5d. Координаты самых удалённых секторов по оси абсцисс или ординат лежат в промежутке (2dn – 0,5d; 2d(n + 1) – 0,5d] (отсчёт индекса сектора начинается из левого верхнего угла).

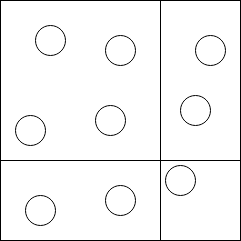


Рисунок . Пример разделения карты на области. Кружками отмечены вершины, квадратами области.

Поиск ближайшего узла происходит следующим образом:

1. Каждая из координат должна лежать в пределах [2dn – 0,5d; 2d(n + 1) – 0,5d), где n – количество узлов по одной оси, если выходит, то бросаем исключение, так как за пределы карты нельзя выходить.
2. Далее определяем сектор для переданной точки по формуле n = (x + 0.5d / 2d) для каждой координаты, где x – координата x или y, d - мин. расстояние между узлами. Каждое из полученных чисел округляется в меньшую сторону и будет номером области по оси абцисс и ординат соответсвенно. Обозначим эти числа как x и y.
3. Далее по формуле x + y \* n, где n – количество узлов вдоль одной из осей, находим индекс вершины

## Скриптовый бот

Скриптовый бот – бот с постоянным поведением, представляющий из себя скрипт генерирующий случайные команды со случайными параметрами. На представленном боте обучется нейросеть. Так как поведение бота постоянное, то прогресс в обучении будет хорошо виден в виде процента побед, количества уничтоженных подразделений и награды за бой в виде числа.

Бот каждый кадр выбирает случайное подразделение(я) и случайно выбирает одну из доступных, для выбранного подразделения, команду:

1. moveOrAttack – двигаться или атаковать. Параметры: unit\_id – id подразделения; position – позиция, куда нужно двигаться. Передвижение и атака в одной команде, так как подразделение останавливается и начинает атаковать, когда находит противника в радиусе действия своего оружия.
2. retreatOrStorm – отступать или штурмовать. Параметры: unit\_id – id подразделения; position – позиция, куда нужно двигаться. Эта команда доступна, если подразделение атаковано.
3. stopOrDefense – остановиться или обороняться. Параметры: unit\_id. Во время движения по маршруту подразделение может остановиться. Когда подразделение стоит у него растёт уровень защиты, так как оно окапывается.

Пояснения для 1 и 2 команды:

1. Для вычисления параметра position случайно выбирается одно из 8 направлений (вверх, вправо вверх, вправо, вправо вниз, вниз, влево вниз, влево, влево вверх) и дистанция. Подразделение может передвигаться на короткие дистанции до 5 км включительно (при этом подразделение перемещается по прямой, игнорируя дороги, но в таком случае двигаться будет медленнее) или на длинные (больше 5 км и подразделение передвигается по дорогам). Далее строится вектор, у которого начало имеет координату позиции подразделения, а координата конца будет вычислена исходя из выбранного направления и дистанции.
2. При выборе короткой дистанции будет выбрано расстояние 5 км, длинной 10 км.

## Нейросеть



### Общая структура

Общее структура нейросети показана на .

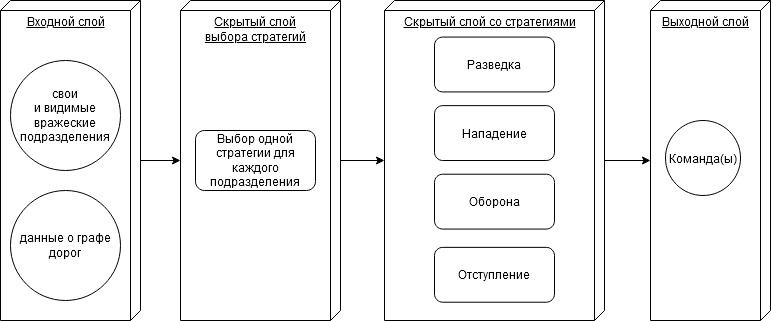


Рисунок . Общая структура нейросети

Процесс управления подразделениями происходит следующим образом:

1. На входной слой приходят данные о своих и видимых подразделениях противника, и данные о графе дорог.
2. Слой выбора стратегий выбирает для каждого подразделения слой-стратегию. У каждого вида подразделения свой набор стратегий, но структура слоёв-стратегий одинаковая, отличия только в весах.
3. Выбранный слой-стратегия генерирует команду для подразделения, полученная команда помещается в массив команд, который будет передан игре после того как для всех подразделений будет сгенерированы команды.

В процессе проектирования и применения постепенного обучения было решено разделить процесс «командования» подразделениями на несколько отдельных задач, которые были перечислены выше при описании слоя стратегий на . Каждый из слоёв обучается отдельно, затем эти слои объединяются в один слой-стратегий и используются в итоговой нейросети. Итоговой нейросетью называется нейросеть, которая может управлять всеми видами подразделений и использовать все слои-стратегии для каждого подразделения.

Для оценки возможностей технологий машинного обучения в задачах на графе дорог достаточно результатов обучения одного слоя-стратегий для одного подразделения. Обучение нейросетей подобных слою-стратегий занимает много времени, чтобы получить 60% шанс победы, предполагается, что необходимо примерно 2500 тренировок как в [], поэтому скорее всего не удастся получить итоговую и хорошо обученную(с шансом выигрыша минимум 60%) нейросеть на момент защиты ВКР.

Из-за выше сказанного было принято решение описать слой выбора стратегий и отложить его реализацию, пока все 4 слоя-стратегий не будут готовы хотя бы для двух видов подразделений.

Слой-стратегия рассчитан на управление подразделением одного вида, поэтому, чтобы итоговая нейросеть могла управлять всеми видами, необходимо создавать для каждого вида свои слои-стратегии. Для ускорения обучения каждого вида подразделений применяется трансферное обучение.



### Представление состояния игры

Представление состояния игры строится на основе данных о своих подразделениях, видимых подразделениях противника, данных о графе. Оно формируется игровым сервером и отправляется ИИ-серверу, ИИ-сервер передаёт его входному слою.

Чтобы нейросеть могла работать с картами любых размеров, необходимо обобщать данные, для этого карта делится на 16 равных областей. Для каждой из областей вычисляются определенные параметры, о которых будет сказано дальше. За основу представления состояния игры было взято представление из [] и адоптировано для игры «WarOnMap». Представление состояние игры содержит следующие поля:

* Общие данные для всех подразделений, являются вещественными числами равных сумме значений всех подразделений:
  + TroopAmount: Численность состава подразделения (аналог «очков здоровья» в стратегиях).
  + Organization: Уровень организации. Боевой дух, если значение будет равно нулю, подразделение будет уничтожено, повышается пока подразделение стоит и не участвует в бою. Сильно снижается во время боя, во время передвижения уменьшается очень медленно.
  + Experience: боевой опыт(чем больше, тем больше урона наносит подразделение)
  + Overlap: подразделения могут пересекаться, если стоят слишком близко друг к другу, из-за этого оба подразделения получают больше урона, чем ближе другу к другу тем больше урона. Также пересечение подразделений снижает скорость передвижения.
  + Speed. Скорость передвижения. Измеряется относительно максимальной скорости конкретного подразделения, то есть вычисляется *текущая скорость / максимальная скорость*.
* Данные о своих подразделениях
  + OwnSumInfo: расстояния между собственными подразделениями, суммируются в каждой области;
  + OwnMaxInfo: расстояния между собственными подразделениями, максимизированы в каждой области;
* Данные о видимых подразделениях противника
  + EnemySumInfo: расстояния между вражескими подразделениями, суммируются в каждой области;
  + EnemyMaxInfo: расстояния между вражескими подразделениями, максимизированы в каждой области.
* Данные о графе дорог
  + GraphDensity: Плотность графа дорог в 16 секторах, измеряется в количестве перекрёстков
* Время игры. Нужно для вычисления награды за потраченное время.

В каждой области определяется плотность графа дорог. Нейросети нужно будет понимать, есть ли дороги в этой области, и имеются ли дороги, пройдя по которым можно дойти до цели. Всё это важно, так как по дорогам передвижение значительно быстрее, чем передвижение по прямой, игнорируя дороги. Так как анализ самих дорог занимает много времени, то нужен более быстрый в вычислении критерий. Известно, что у каждого перекрёстка, есть как минимум одна, соединённая с ним, дорога, это значит, что чем больше перекрёстков будет в области, тем больше дорог, следовательно, вероятность того, что в области есть дорога, ведущая к цели, выше. Поэтому для ориентации на графе была выбрана плотность дорог в области.

Плотность графа определяется следующим образом:

* 1. В области ставятся 9 точек, каждая из которых равноудалена от соседних и границ области. Пример области приведён на .

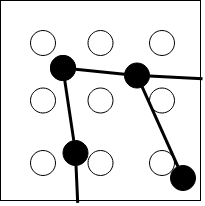


Рисунок . Пример области «карты графа». Белые круги это рассматриваемые точки, черные круги – вершины графа дорог, черные линии - дороги

* 1. Для каждой выбранной точки находится ближайший к нему перекрёсток (на модельном графе перебираются точки лежащие в области, если граф реальный, то гео-сервер ищет ближайшую точку). Если этот перекрёсток уже находился ранее, его не включают в список найденных, иначе найденный перекрёсток включается в список найденных. Длина списка найденных перекрёстков будет равна плотности графа в рассматриваемой области.

Расстояния между подразделениями для параметров OwnSumInfo, OwnMaxInfo, EnemySumInfo и EnemyMaxInfo рассчитываются следующим образом:

* Определяется ближайшее к центру подразделение.
* Вычисляется вектор от других подразделений до центрального, если подразделение в другой области, то вектор будет равно нулевым.
* Полученные вектора линеаризуются, чтобы обобщить данные:
  + Для этого каждую координату вектора делят на длину и ширину сектора соответственно.
  + Далее определяется коэффициент масштабирования для каждой координаты, для этого нужно определить, что больше длина или ширина, для большей стороны коэффициент будет равен 1, для меньшей равен *длине меньшей стороны / длину большей стороны*.
  + Координаты линеаризованного вектора умножаются на полученные в предыдущем шаге коэффициенты.
* Вычисляется длина линеаризованного вектора, который будет расстоянием до подразделения.



### Слой стратегий

Существуют следующие виды слоёв-стратегий: разведка, нападение, оборона, отступление.

Стратегия **разведки –** стратегия, при которой подразделение должна найти хотя бы одно подразделение или базу противника (далее «база» будет упоминаться только, если рассматриваемое касается только базы, в остальных случаях под «подразделениями» будут иметься ввиду подразделения и базы).

Стратегия разведки доступна только в том случае, если остались не найденные подразделения противника. Если подразделение противника уходит из обзора войск, оно станет снова ненайденным.

Стратегия **нападения –** стратегия, при которой подразделение должно подойти достаточно близко к противнику, чтобы атаковать его, и уничтожить подразделение.

Стратегия нападения доступна, если есть видимые подразделения противника.

Стратегия **обороны** **–** стратегия, при которой подразделение должно остановиться, и перейти в режим обороны, при котором увеличивается коэффициент защиты. Чем дольше стоит подразделение в режиме обороны, тем больше коэффициент защиты и тем меньше получает урона. Стратегия обороны доступна всегда.

Стратегия **отступления** **–** стратегия, при которой подразделение должно уйти от противника, после чего задача считается выполненной, если подразделение в состоянии «не атакован». Стратегия отступления доступна в том случае, если подразделение в состоянии «атакован».

### Слой выбора стратегий

Представленный слой выбирает для каждого подразделения слой-стратегию, который сгенерирует команду. Слой-стратегия принимает на вход состояние игры, которое передаёт слой выбора стратегий:

Слой выбора стратегий будет обучаться в последнюю очередь, так как для его работы нужны обученные слои-стратегии. При запуске итоговой нейросети, можно будет обновлять веса только слоя выбора, или слоя выбора вместе со слоями стратегиями.



### Метод обучения

Игровая партия в игре «WarOnMap» представляет из себя сражение двух армий, в которых есть несколько подразделений и одна или несколько баз. Рассматриваемое сражение аппроксимируется как игра Маркова с несколькими агентами, где используется набор состояний S для описания свойств всех агентов, и среды, набор действий A1, ..., AN и наблюдений O1, ..., ON для каждого агента.

В ходе сражения подразделения должны действовать совместно, чтобы победить. Чтобы создать гибкую структуру нейросети, которая позволит управлять произвольным количеством подразделений, необходимо, чтобы каждое подразделение действовало самостоятельно исходя из своего наблюдения Oi, при котором соседние подразделения рассматриваются как часть среды, также.

Много-агентная кооперация разных видов подразделений является сложной задачей, поэтому слои-стратегии обучаются на группе подразделений одного вида, против группы подразделений того же или другого вида. Используемая политика в слое-стратегии применяется ко всем подразделениям определённого вида. Цель каждого подразделения - максимизировать результат действий группы.

Для решения проблемы много-агентного обучения используется обучение с подкреплением, при котором процесс взаимодействия агента и среды сформулирован как процесс принятия решений Маркова. В качестве метода обучения нейросети используется алгоритм Адама, который реализован и находится в библиотеке Tensorflow. Был выбран именно этот алгоритм, потому что он эффективно работает с большим количеством данных, хорошо работает с «шумными» или «редкими» градиентами, потребляет мало памяти и имеет высокую скорость вычислений.[11]

### Функция вознаграждения

Функция вознаграждения обеспечивает полезную обратную связь для RL-агентов, которая оказывает значительное влияние на результаты обучения. Существует проблема отсроченных вознаграждений, суть которой заключается в том, что если вознаграждение рассчитывать, исходя только из текущего состояния, то такая функция не позволит нейросети достигнуть цели, но и расчёт награды для большого количества шагов затрудняет выяснение того, какой набор действий отвечает за соответствующие вознаграждения. Поэтому разработаны функции вознаграждения с промежуточными награждениями. Функций несколько, потому что у каждого слоя-стратегии своя задача.

Слой-стратегия обороны только генерирует команду «остановиться», поэтому у неё не будет функции вознаграждения. Слой разведки имеет подфункцию вознаграждения за потраченное время. У слоя нападения нет этой подфункции, потому что фактор времени при нападении не важен, так как важно нанести урон. Слой отступления не имеет подфункции награждения за время потому, что подразделение может быстрее отступить, пройдя по прямой, с большими потерями, или медленее, передвигаясь по дорогам, но при этом не известно, сколько получит урона(это зависит от разветвлённости сети дорог)

Подфункция вознаграждения за потраченное время имеет вид:

rt = (time\_last – time\_current) \* 1.2

где *time\_last* - это текущее время в прошлом представлении игры, измеряется в тиках, *time\_current* - соответственно время в текущем представлении, «1.2» - коэффициент определяющий важность потраченного времени.

У остальных функций есть общее – это подфункции вознаграждения за нанесение урона, потерю уровня организации, получение боевого опыта, пересечение подразделений и скорость передвижения. Подфункции вознаграждения за нанесение урона и потерю уровня организации имеют следующий вид(за основу взята функция из []):

rt = ρ×(damage\_amountt - (unit\_hitpointt−1 – unit\_hitpointt)) (1)

где *damage\_amount* - это количество урона или количество потерянных очков организации, нанесенного атакой наших подразделений, *unit\_hitpoint* – численность нашего подразделения или количество очков организации. Численность состава базы в функции умножается на 10, так уничтожение баз(ы) приведёт к поражению. ρ является нормализованным фактором для баланса общей численности состава или уровня организации наших подразделений и подразделений противника,



где H - количество вражеских подразделений, а N - количество наших подразделений. Представленный фактор нормализации необходим для управления различным количеством и типами подразделений. Без надлежащей нормализации, политика сети будет хуже сходиться, и нашим подразделениям нужно гораздо больше времени для нахождения полезного поведения.

Подфункции вознаграждения за получение боевого опыта и скорости имеют следующий вид:

rt = (unit\_experiencet – unit\_experiencet−1) / unit\_experiencet \* unit\_amount (2)

Где *unit\_experience* – общее количество опыта подразделений, *unit\_amount* – количество подразделений. Количество полученного опыта нужно делить на количество за прошлый шаг и количество подразделений, чтобы получить относительный прирост и обобщить данные.

Подфункция вознаграждения за скорость подразделений имеет следующий вид:

rt = current\_speed\_award + ((unit\_speedt – unit\_speedt−1) / unit\_speedt \* unit\_amount) (2)

Где *unit\_*speed – сумма значений *текущей скорости / максимальная скорость*, *unit\_amount* – количество подразделений, *current\_speed\_award* – награда за текущую скорость равная unit\_speedt, нужна для награждения за сохранение скорости и предотвращения остановок во время передвижения.

Подфункция вознаграждения за пересечение подразделений выглядит следующим образом:

rt = - m \* (unit\_overlapt – unit\_overlapt−1) / unit\_overlapt \* unit\_amount (3)

Где *unit\_overlap* – коэффициент пересечения подразделений, m – коэффициент, который равен 3 если (unit\_overlapt – unit\_overlapt−1) больше 0, иначе равен 1.

Награда за передвижение на короткие дистанции(5 км) не по дорогам равна 0, на длинные(10 км) по дорогам 0,1 за тик/кадр.

Дополнительные вознаграждения:

1. Стратегия разведки:
   1. Обнаружено подразделение противника: 50
2. Стратегия нападения
   1. Уничтожена база противника: 10000
   2. Уничтожена своя база: -50000
   3. Уничтожено подразделение другого вида: 1000
   4. Уничтожено своё подразделение другого вида: -5000
3. Стратегия отступления
   1. Награды из прошлого пункта сюда тоже относятся
   2. Удалось убежать: 10

# Эксперименты и итоги



## Процесс обучения

Инициировать обучение нейросети будет специальный скрипт, который сначала отправляет запрос на создание специальной игры для обучения, затем создаётся игра, сокет и происходит подключение к игре.

Игровой сервер позволяет создать два вида игр для обучения нейросети:

1. Игра для обучения итоговой нейросети.

Запрос:

/game-for-result-network?nodeAmount=<число>&distanceBetweenNodes=<число>

, где nodeAmount – количество узлов на одной из сторон графа, distanceBetweenNodes – расстояние между узлами графа. Создаётся модельный граф с заданными параметрами и полным набором подразделений всех видов.

1. Игра для обучения конкретного слоя стратегии.

Запрос:

/game-for-layer?layerName=<строка>&troopType=<строка> &nodeAmount=<число>&distanceBetweenNodes=<число>

, где layerName – название слоя стратегии, troopType – тип подразделения. Здесь в зависимости от вида слоя-стратегии создаются разные наборы подразделений:

* для слоя-стратегии разведки набор подразделений состоит из базы и 3 подразделений вида troopType
* для слоя нападения тот же набор, что и для слоя разведки
* для слоя отступления набор подразделении состоит из базы и одного подразделения вида troopType. База будет поставлена в дальний угол карты, а боевое подразделение нейросети и противника в центре карты.

## Результаты

Для эксперимента был выбран слой-стратегия разведки для мотострелковых подразделений, так как у них самая большая скорость передвижения. Цель слоя разведки: найти одно подразделение противника за наименьшее время. Обучающая карта состоит из модельного графа размером 12 на 12 узлов, размер карты 200 км, расстояние между узлами равно 200 км / 14 (12 узлов + 2, потому что узлы имеют отступ от краёв карты равный расстоянию между узлами, подробнее см раздел 3.4.2 и 3.4.3). Одно подразделение (мотострелковое) на таком графе может доехать из левого нижнего в правый верхний угол карты за 20 реальных секунд, поэтому было выбрано время обучения равное 40 секундам или 1000 тиков, из которых 20 секунд – дополнительное время. Обучение прекращается, когда найдено одно подразделение. В начале тренировки войска ставятся в ряд, начало которого устанавливается в случайном месте на половине карты соответствующего игрока, что предотвращает переобучение и поощряет слой-разведку разведывать в разных направлениях, чтобы разведать как можно большую территорию. Расставленные войска расположены близко друг к другу, из-за чего происходит пересечение, что значительно снижает скорость передвижения в начале тренировки.

Награда за нахождение противника была выбрана так, чтобы можно было по графику вознаграждений определить, через какое количество тренировок цель была достигнута. Это означает, что награда за нахождение является достаточно большой, чтобы компенсировать отрицательное вознаграждение, полученное ранее, поэтому на графике наград, для наглядности, не будет учитываться награда за нахождение, кроме того обнаружение противника может быть случайным, так как скриптовый бот действует случайным образом. Всего было проведено 100 тренировок.

Рисунок . Среднее количество тиков в тренировке слоя-разведки

Рисунок . Среднее вознаграждение во время тренировки слоя-разведки без учёта награды за обнаружение (равна 50). Зелёными точками слева направа отмечены тренировки, в которых было обнаружено подразделение противника (13, 22, 27, 34). После 34 тренировки, войска противника всегда обнаруживались, поэтому дальше точки не расставлены.

Обучение слоя-разведки происходило в несколько этапов:

1. Определение направления разведки и расхождение войск в разные стороны. (Первые 30 тренировок) На этом этапе подразделения перемещаются практически в случайном направлении, из-за чего они не отходят далеко от начальных позиций. Кроме того из-за пересечения войска получают большое отрицательное вознаграждение.
2. Определение оптимальной дистанции передвижения (по дорогам(10 км) или по прямой(5 км), что медленнее) и того, по каким участкам лучше передвигаться (Остальные 70 тренировок). В начале длина выбиралась случайно, позже подразделения стали передвигаться на длинные дистанции по дорогам. Также случайно выбирался сектор карты для исследования, постепенно нейросеть стала «понимать», что быстрее передвигаться можно по участкам с большей плотностью дорог. Так как бот-противник действует непредсказуемо, поэтому нейросеть изучала общие закономерности, и обучение не прекратилось после 34 тренировки, когда отрицательное вознаграждение за пересечение стало минимальным.

# Заключение

В результате проведённого исследования было сделано следующее:

1. Предложено представление дорожного графа, которое позволяет нейросети выполнять свои задачи на графе любого размера.
2. Спроектирован и создан модельный граф, точнее его генератор. Представленный граф имитирует реальный граф дорог, но обладает гораздо меньшим размером и генерируется случайно, что позволяет избежать переобучения.
3. Предложена архитектура нейросети, реализован только слой-разведки для мотострелковых подразделений.
4. Обучен слой-разведка на скриптовом боте, который генерировал случайные команды со случайными параметрами.

Результаты обучения слоя-разведки показывают, что обучение с подкреплением позволяет выполнять задачи на дорожном графе. Предложенное представление дорожного графа можно применить для других задач на графах дорог, так как оно является достаточно обобщённым. Но в других задачах, вероятно, потребуется изменить количество секторов по диагонали и горизонтали, для большей или меньшей детализации.

Предложенная архитектура нейросети рассчитана на сражение одного вида подразделений против другого, что означает, что подразделение одного вида будет хорошо сражаться только против определённого вида, и плохо против остальных видов. Эта проблема не решалась, так это выходит за рамки текущего исследования.

# Список литературы

1. OpenAI and Dota 2 [Электронный ресурс] - Режим доступа: https://openai.com/blog/dota-2/ , свободный
2. Википедия. Dota 2[Электронный ресурс] - Режим доступа: <https://ru.wikipedia.org/wiki/Dota_2> , свободный
3. Competitive Self-Play [Электронный ресурс] - Режим доступа: <https://openai.com/blog/competitive-self-play/> , свободный
4. 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
5. 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>
6. 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>
7. 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.
8. 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>
9. 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>
10. 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
11. Diederik P. Kingma, Jimmy Lei, «Adam: A method for stohasti optimization» [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf
12. 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/
13. 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
14. 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>
15. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.
16. M. A. Yasin, W. A. Al-Ashwal, A. M. Shire, S. A. Hamzah, and K. N. Ramli, “Tri-band planar inverted F-antenna (PIFA) for GSM bands and bluetooth applications,” ARPN Journal of Engineering and Applied Sciences, vol. 10, no. 19, pp. 8740–8744, 2015.
17. N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, “Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks,” in International Conference on Learning Representations, 2017.
18. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction.MIT Press, 1998.
19. L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” Journal of Artificial Intelligence Research, vol. 4, no. 1, pp. 237–285, 1996
20. D. Zhao, K. Shao, Y. Zhu, D. Li, Y. Chen, H. Wang, D. Liu, T. Zhou, and C. Wang, “Review of deep reinforcement learning and discussions on the development of computer Go,” Control Theory and Applications, vol. 33, no. 6, pp. 701–717, 2016.
21. Z. Tang, K. Shao, D. Zhao, and Y. Zhu, “Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero,” Control Theory and Applications, vol. 34, no. 12, pp. 1529–1546, 2017.
22. H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in The Thirtieth AAAI Conference on Artificial Intelligence, 2016, pp. 2094–2100.
23. T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in International Conference on Learning Representations, 2016.
24. Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, “Dueling network architectures for deep reinforcement learning,” in International Conference on Machine Learning, 2016, pp. 1995–2003.
25. A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria,V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen et al., “Massively parallel methods for deep reinforcement learning,” in International Conference on Machine Learning, 2015.
26. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in International Conference on Machine Learning, 2016, pp. 1928–1937.
27. D. Li, D. Zhao, Q. Zhang, and C. Luo, “Policy gradient methods with gaussian process modelling acceleration,” in International Joint Conference on Neural Networks, 2017, pp. 1774–1779.
28. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. P. Wierstra, “Continuous control with deep reinforcement learning,” in International Conference on Learning Representations, 2016.
29. J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, “Trust region policy optimization,” in International Conference on Machine Learning, 2015, pp. 1889–1897.
30. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint, 2017.
31. S. Levine and V. Koltun, “Guided policy search,” in International Conference on Machine Learning, 2013, pp. 1–9.
32. S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep Q-learning with model-based acceleration,” in International Conference on Machine Learning, 2016, pp. 2829–2838.
33. M. Watter, J. T. Springenberg, J. Boedecker, and M. A. Riedmiller, “Embed to control: a locally linear latent dynamics model for control from raw images,” in Neural Information Processing Systems, 2015, pp. 2746–2754.
34. M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” Machine Learning Proceedings, pp. 157–163, 1994.
35. T. Ming, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” in Proceedings of the tenth International Conference on Machine Learning, 1993, pp. 330–337.
36. Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai, “FMRQ a multiagent reinforcement learning algorithm for fully cooperative tasks,” IEEE Transactions on Cybernetics, vol. 47, no. 6, pp. 1367–1379, 2017.
37. S. Sukhbaatar, A. Szlam, and R. Fergus, “Learning multiagent communication with backpropagation,” in Neural Information Processing Systems, 2016, pp. 2244–2252.
38. L. Marc, Z. Vinicius, and G. Audrunas, “A unified game-theoretic approach to multiagent reinforcement learning,” arXiv preprint arXiv:1711.00832.
39. K. G. Jayesh, E. Maxim, and K. Mykel, “Cooperative multi-agent control using deep reinforcement learning,” in International Conference on Autonomous Agents and Multiagent Systems, 2017, pp. 66–83.
40. Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 41–48.
41. A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabskabarwiska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, and J. Agapiou, “Hybrid computing using a neural network with dynamic external memory.” Nature, vol. 538, no. 7626, p. 471, 2016.
42. Y. Wu and Y. Tian, “Training agent for first-person shooter game with actor-critic curriculum learning,” in International Conference on Learning Representations, 2017.
43. Q. Dong, S. Gong, and X. Zhu, “Multi-task curriculum transfer deep learning of clothing attributes,” in IEEE Winter Conference on Applications of Computer Vision, 2017, pp. 520–529.
44. X. Glorot, A. Bordes, Y. Bengio, X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.
45. V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in International Conference on Machine Learning, 2010, pp. 807–814.
46. S. P. Singh and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” Machine Learning, vol. 22, no. 1, pp. 123–158, 1996.
47. A. Y. Ng, D. Harada, and S. J. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in International Conference on Machine Learning, 1999, pp. 278–287.
48. T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” in Advances in neural information processing systems, 2016, pp. 3675–3683.
49. Hendrik Baier, Peter I. Cowling “Evolutionary MCTS for Multi-Action Adversarial Games” [Online]. Available: https://se1f330a320707f8e.jimcontent.com/download/version/1532703212/module/13604570727/name/evolutionary%20mcts%20for%20multi-action%20adversarial%20games.pdf
50. L. Kocsis and C. Szepesvari, “Bandit based Monte-Carlo planning,” ´ in 17th European Conference on Machine Learning, ECML 2006, ser. Lecture Notes in Computer Science, vol. 4212, 2006, pp. 282–293.
51. R. Coulom, “Efficient selectivity and backup operators in Monte-Carlo Tree Search,” in 5th International Conference on Computers and Games, CG 2006. Revised Papers, ser. Lecture Notes in Computer Science, vol. 4630, 2007, pp. 72–83.
52. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,” CoRR, vol. abs/1712.01815, 2017.
53. C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez-Liebana, S. Samothrakis, and S. Colton, “A survey of Monte Carlo Tree Search methods,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1–43, 2012.
54. N. Justesen, T. Mahlmann, S. Risi, and J. Togelius, “Playing Multi Action Adversarial Games: Online Evolution versus Tree Search,” IEEE Transactions on Computational Intelligence and AI in Games, 2017, in print.
55. M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk, “Single-Player Monte-Carlo Tree Search for SameGame,” Knowledge-Based Systems, vol. 34, pp. 3–11, 2012.
56. N. Justesen, T. Mahlmann, and J. Togelius, “Online Evolution for Multi action Adversarial Games,” in 19th European Conference on Applications of Evolutionary Computation (EvoApplications 2016), ser. Lecture Notes in Computer Science, G. Squillero and P. Burelli, Eds., vol. 9597. Springer, 2016, pp. 590–603.
57. P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-Time Analysis of the Multiarmed Bandit Problem,” Machine Learning, vol. 47, no. 2-3, pp. 235–256, 2002.
58. S. Gelly and Y. Wang, “Exploration Exploitation in Go: UCT for MonteCarlo Go,” in Neural Information Processing Systems Conference (NIPS), On-line trading of Exploration and Exploitation Workshop, 2006.
59. R. Coulom, “Computing elo ratings of move patterns in the game of Go,” in Computer Games Workshop, 2007.
60. G. M. J. B. Chaslot, M. H. M. Winands, J. v. d. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, “Progressive Strategies for Monte-Carlo Tree Search,” New Mathematics and Natural Computation, vol. 4, no. 03, pp. 343–357, 2008.
61. D. Churchill and M. Buro, “Hierarchical Portfolio Search: Prismata’s Robust AI Architecture for Games with Large Search Spaces,” in 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015, A. Jhala and N. Sturtevant, Eds. AAAI Press, 2015, pp. 16–22.
62. N. Justesen, B. Tillman, J. Togelius, and S. Risi, “Script- and clusterbased UCT for StarCraft,” in 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, 2014, pp. 1–8.
63. D. Churchill and M. Buro, “Portfolio Greedy Search and Simulation for Large-scale Combat in StarCraft,” in IEEE Conference on Computational Intelligence in Games, CIG 2013. IEEE, 2013, pp. 1–8.
64. S. Ontan ˜on, “The Combinatorial Multi-Armed Bandit Problem and Its ´ Application to Real-Time Strategy Games,” in 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13, G. Sukthankar and I. Horswill, Eds. AAAI, 2013.
65. T. Yee, V. Lisy, and M. H. Bowling, “Monte Carlo Tree Search in ´ Continuous Action Spaces with Execution Uncertainty,” in Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 690–697.
66. S. M. Lucas and G. Kendall, “Evolutionary Computation and Games,” IEEE Computational Intelligence Magazine, vol. 1, no. 1, pp. 10–18, 2006.
67. S. Risi and J. Togelius, “Neuroevolution in Games: State of the Art and Open Challenges,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 1, pp. 25–41, 2017.
68. N. Cole, S. J. Louis, and C. Miles, “Using a genetic algorithm to tune first-person shooter bots,” in 2004 Congress on Evolutionary Computation (CEC 2004), 2004, pp. 139–145.
69. G. M. J. B. Chaslot, M. H. M. Winands, I. Szita, and H. J. van den Herik, “Cross-entropy for Monte-Carlo Tree Search,” ICGA Journal, vol. 31, no. 3, pp. 145–156, 2008.
70. A. M. Alhejali and S. M. Lucas, “Using genetic programming to evolve heuristics for a Monte Carlo Tree Search Ms Pac-Man agent,” in 2013 IEEE Conference on Computational Intelligence and Games, CIG 2013, 2013, pp. 1–8.
71. A. Benbassat and M. Sipper, “Evomcts: A scalable approach for general game learning,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, no. 4, pp. 382–394, 2014.
72. C. F. Sironi and M. H. M. Winands, “On-line parameter tuning for Monte-Carlo Tree Search in General Game Playing,” in Computer Games Workshop, 2017, pp. 75–95.
73. D. Perez-Liebana, S. Samothrakis, S. M. Lucas, and P. Rohlfshagen,“Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games,” in 2013 Genetic and Evolutionary Computation Conference, GECCO ’13, C. Blum and E. Alba, Eds. ACM, 2013, pp. 351–358.
74. R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, “Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing,” in 20th European Conference on Applications of Evolutionary Computation, EvoApplications 2017, ser. Lecture Notes in Computer Science, G. Squillero and K. Sim, Eds., vol. 10199, 2017, pp. 418–434.
75. N. Justesen, “Artificial Intelligence for Hero Academy,” Master’s thesis, IT University of Copenhagen, 2015.
76. C. Wang, P. Chen, Y. Li, C. Holmgard, and J. Togelius, “Portfolio Online Evolution in StarCraft,” in Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-16, 2016, pp. 114–120.
77. N. Justesen and S. Risi, “Continual Online Evolutionary Planning for Ingame Build Order Adaptation in StarCraft,” in Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, P. A. N. Bosman, Ed. ACM, 2017, pp. 187–194.
78. R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, “Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing,” in 2017 IEEE Congress on Evolutionary Computation, CEC 2017. IEEE, 2017, pp. 1956–1963.
79. H. Horn, V. Volz, D. P. Liebana, and M. Preuss, “MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation,” in IEEE Conference on Computational Intelligence and Games, CIG 2016. IEEE, 2016, pp. 1–8.
80. S. M. Lucas, S. Samothrakis, and D. Perez-Liebana, “Fast Evolutionary Adaptation for Monte Carlo Tree Search,” in 17th European Conference on Applications of Evolutionary Computation, EvoApplications 2014, ser.Lecture Notes in Computer Science, A. I. Esparcia-Alcazar and A. M. ´Mora, Eds., vol. 8602. Springer, 2014, pp. 349–360.
81. D. Perez-Liebana, S. Samothrakis, and S. M. Lucas, “Knowledge-based Fast Evolutionary MCTS for General Video Game Playing,” in 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, 2014, pp. 1–8.
82. T. Hong, K. Huang, and W. Lin, “Adversarial Search by Evolutionary Computation,” Evolutionary Computation, vol. 9, no. 3, pp. 371–385, 2001.
83. Niels Justesen, “Hero AIcademy.” [Online]. Available: https://github.com/njustesen/hero-aicademy
84. Robot Entertainment, “Hero Academy.” [Online]. Available: <http://www.robotentertainment.com/games/heroacademy/>
85. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II [Online]. Available: <https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii>
86. David Balduzzi, Karl Tuyls, Julien Perolat, Thore Graepel “Re-evaluating Evaluation” [Online]. Available: <https://papers.nips.cc/paper/7588-re-evaluating-evaluation.pdf>
87. Starcraft AI Competition [Online]. Available: http://www.cs.mun.ca/~dchurchill/starcraftaicomp/results.shtml