Оглавление

[1. Введение 2](#_Toc20206316)

[1.1 Недостатки существующих решений и предлагаемые решения 2](#_Toc20206317)

[1.2 Постановка задачи 3](#_Toc20206318)

[1.3 Научная новизна 4](#_Toc20206319)

[2. Аналитическая часть 5](#_Toc20206320)

[2.1 Обзор исследований 5](#_Toc20206321)

[2.1.1 Бот компании OpenAI игры Dota 2 5](#_Toc20206322)

[2.1.2 Бот компании OpenAI играющий в сумо и футбол 6](#_Toc20206323)

[2.1.3 Игровая среда Deep RTS 7](#_Toc20206324)

[2.1.4 Управление армией в StarCraft 9](#_Toc20206325)

[2.1.5 Эволюционный алгоритм поиска дерева Монте-Карло для игры Hero Academy 18](#_Toc20206326)

[2.1.6 AlphaStar – полноценный ИИ для StarCraft II 25](#_Toc20206327)

[Конструкторская часть 30](#_Toc20206328)

[Используемые методологии разработки и технологии 30](#_Toc20206329)

[Test-driven development 30](#_Toc20206330)

[C4Model 32](#_Toc20206331)

[Нотация ER-диаграмм Crow’s foot 36](#_Toc20206332)

[UML 40](#_Toc20206333)

[Используемые программные продукты и языки программирования 47](#_Toc20206334)

[Язык JavaScript 47](#_Toc20206335)

[Список литературы 49](#_Toc20206336)

[Приложение А. Техническое задание. 57](#_Toc20206337)

[Введение 57](#_Toc20206338)

[Назначение разработки 57](#_Toc20206339)

[Основание для разработки 57](#_Toc20206340)

[Требования к программе или программному изделию 57](#_Toc20206341)

[Требования к программной документации 58](#_Toc20206342)

[Стадии и этапы разработки 58](#_Toc20206343)

[Порядок контроля и приемки 58](#_Toc20206344)

[Приложение Б. Программная спецификация. 59](#_Toc20206345)

[Документация 59](#_Toc20206346)

[Компоненты 59](#_Toc20206347)

[Приложение В. Программная документация. 60](#_Toc20206348)

[Диаграмма контекста 60](#_Toc20206349)

# Введение

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

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

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

## 1.1 Недостатки существующих решений и предлагаемые решения

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

## 1.2 Постановка задачи

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

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

Для достижения поставленной цели необходимо выполнить следующие задачи:

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

## 1.3 Научная новизна

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

# 2. Аналитическая часть

## 2.1 Обзор исследований

В рамках ВКР был проведён обзор последних исследований в применении машинного обучения в играх, в том числе и игр жанра стратегий реального времени. Рассмотренные исследования будут сравниваться по следующим критериям:

1. Работает с дорожным графом(картой дорог)
2. Это игра является стратегией
3. События происходят в реальном времени
4. Юниты имеют большое количество параметров(например, запас сил, боеприпасов, еды, воды, помимо основных вроде скорости атаки и количества здоровья)
5. Объекты местности влияют на боеспособность(например, горы будут мешать артиллерии попадать по пехоте, спрятавшейся за ней)
6. Есть туман войны
7. Нейросеть управляет группой разнотипых юнитов

### 2.1.1 Бот компании OpenAI игры Dota 2

Компания OpenAI в 2019 году создала бота для игры Dota 2. Представленный бот обучался с нуля на самом себе, без имитационного обучения и поиска по дереву. [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 своих противников, благодаря чему смог обыграть профессиональных игроков.

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

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

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

### 2.1.3 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 обозреваются среды 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 самом себе.
* AlphaGO – первый ИИ, который победил профессионального игрока в Go. AlphaGO является фреймворком обучения с подкреплением, который использует дерево поиска Монте-Карло и две глубоких нейронных сети стоимости и стратегическую оценку.[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 проводить исследования на разных уровнях сложности в планировании, реагирования и контроле. Deep RTS имеет несколько конфигурации, при этом они являются детерминированными и недолговечными. Действия в недолговечной конфигурации непосредственно применяются к среде в течение нескольких игровых кадров, что делает корреляцию между действием и вознаграждением более наблюдаемой. Авторы решили не использовать долговременные конфигурации, так как они значительно усложняют пространство состояний и время обучения.

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

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

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

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

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

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

Deep RTS, на момент написания статьи, была одной из самых производительных сред. Deep RTS могла обновлять игру 7 000 000 раз в секунду (на самой маленькой карте 10 на 10 клеток), тогда как ELF только 36000[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.

### 2.1.4 Управление армией в StarCraft

Kun Shao, Yuanheng Zhu, Member, IEEE and 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 оценки на основе части последовательности, не дожидаясь окончательного результата [Id\_2\_1\_4\_\_25]. Наиболее широко известными алгоритмами обучения TD являются Q-learning и Sarsa:
   1. Q-learning. Он оценивает ценность совершения действия в данном состоянии и итеративно обновляет оценку Q-значения в соответствии с наблюдаемым вознаграждением. Ошибка TD δt в Q-обучении вычисляется как δt = r t+1 + γ max Q (st+1, a) - Q (st, at) (2). Q-learning - это метод обучения вне политики, это означает, что он изучает различные политики сравнивая с одним выбранным действием.
   2. Sarsa - это метод, основанный на политике, что означает, что политика используется как для выбора действий, так и для обновления предыдущего значения Q [Id\_2\_1\_4\_\_24].
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) [Id\_2\_1\_4\_\_30] [Id\_2\_1\_4\_\_31]. Как самый известный алгоритм DRL, глубокая Q-сеть (deep Q-network(DQN)) использует технику воспроизведения опыта и целевую сеть для удаления корреляций между выборками и стабилизации процесса обучения [Id\_2\_1\_4\_\_5]. За последние несколько лет появилось множество улучшений DQN, включая двойное DQN [Id\_2\_1\_4\_\_32], приоритетное DQN [Id\_2\_1\_4\_\_33], дуэльное DQN [Id\_2\_1\_4\_\_34], распределенное DQN [Id\_2\_1\_4\_\_35] и асинхронное DQN [Id\_2\_1\_4\_\_36].

Помимо основанных на значениях методов DRL, таких как DQN и его вариантов, методы DRL на основе политики используют глубокие сети для параметризации и прямой оптимизации политики [Id\_2\_1\_4\_\_37]. Глубокий детерминированный градиент политики (Deep deterministic policy gradient, DDPG) - это непрерывный аналог DQN, который использует критику для оценки ценности текущей политики и актёра(actor) для обновления политики [Id\_2\_1\_4\_\_38]. Методы DRL на основе политик играют важную роль в непрерывном контроле, включая асинхронно преимущественную критику актёра (asynchronous advantage actor-critic (A3C)) [Id\_2\_1\_4\_\_36], оптимизацию политики в области доверия (trust region policy optimization , TRPO) [Id\_2\_1\_4\_\_39], оптимизацию проксимальной политики (proximal policy optimization, PPO) [Id\_2\_1\_4\_\_40] и так далее.

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

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

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)

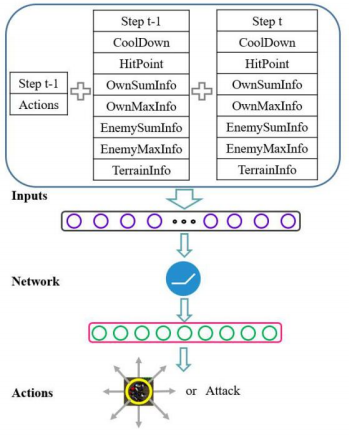


Рисунок 1. Представление модели обучения одного юнита в сценарии микроуправления StarCraft. Представление состояния состоит из трех частей и нейронная сеть используется как аппроксиматор функции. Сеть выводит вероятности движения в 8 направлениях и атаки.

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)

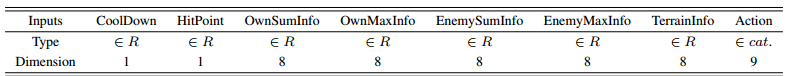


Таблица 1. Типы данных и размерность модели. R – число с плавающей запятой, cat – категория действия, является одним из 8 направлений движения или атакой

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.

#### 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.

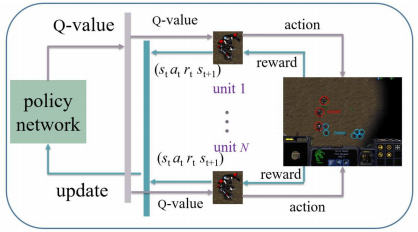


Рисунок 2. Диаграмма обучения с подкреплением PS-MAGDS в сценариях микроуправления StarCraft.

***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 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.

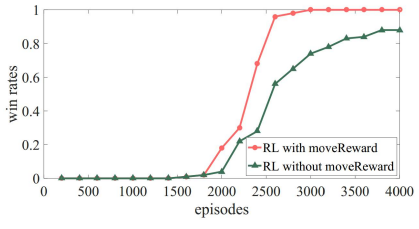


Рисунок 3. Процент побед подразделений в сценарии «3 голиафа против 6 зилотов» через каждые 200 тренировок. moveReward – награда за перемещение.

#### 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).

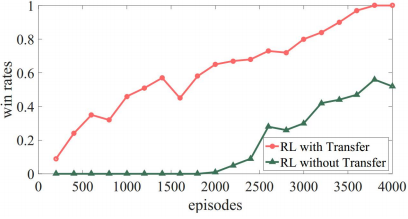


Рисунок 4. Процент побед юнитов в сценарии «3 голиафа против 20 зерглингов» через каждые 200 тренировок.

. 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.

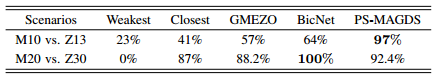


Таблица 2. Сравнение эффективности модели с базовыми методами в двух сценариях «Морпехи против зерглингов»

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 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.

### 2.1.5 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.



Рис. 1: 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»).

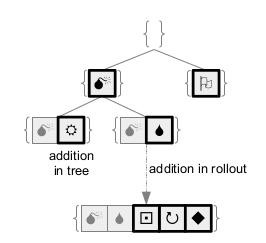


Рисунок 2. Дерево поиска в классическом MCTS и ее вариантов (не исследующий MCTS, BB-MCTS). Узлы представляют собой частичные последовательности действий или состояния, возникающие из них. Ребра представляют собой добавление атомарного действия к последовательности или применение атомарного действия 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.

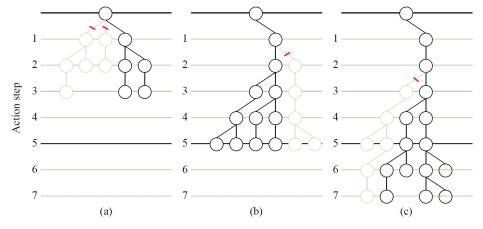


Рис. 3: Стратегия поиска «сжигания моста» (иллюстрация адаптирована из [53]). (а) После фазы 1 все ветви, кроме лучшей, обрезаются в корне. (б, в) После фаз 2, 3,. , , n, обрезка применяется на глубине 2, 3,. , , п. Часть дерева из лучшей ветви сохраняется.

**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.

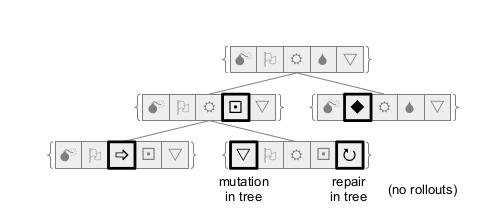


Рисунок 4: Дерево EMCTS. Узлы представляют собой последовательности завершённых действий (геномы) или состояния, возникающие из них. Рёбра представляют собой мутацию простейшего действия в геноме. Ремонт может быть необходимым, если эти мутации могут привести к некорректным геномам. После каждого расширения 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.

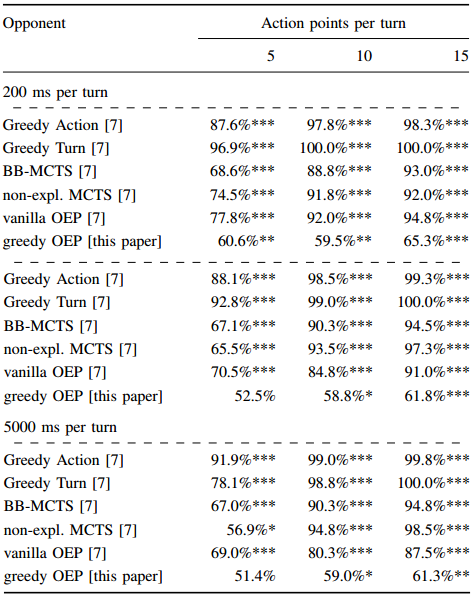


Таблица 1. Процент выигранных EMCTS сражений против других алгоритмов. Звездочки указывают на значительное превосходство, р вероятность победы противника: \* р <0,05, \*\* р <0,01, \*\*\* р <0,001

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.

### 2.1.6 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.

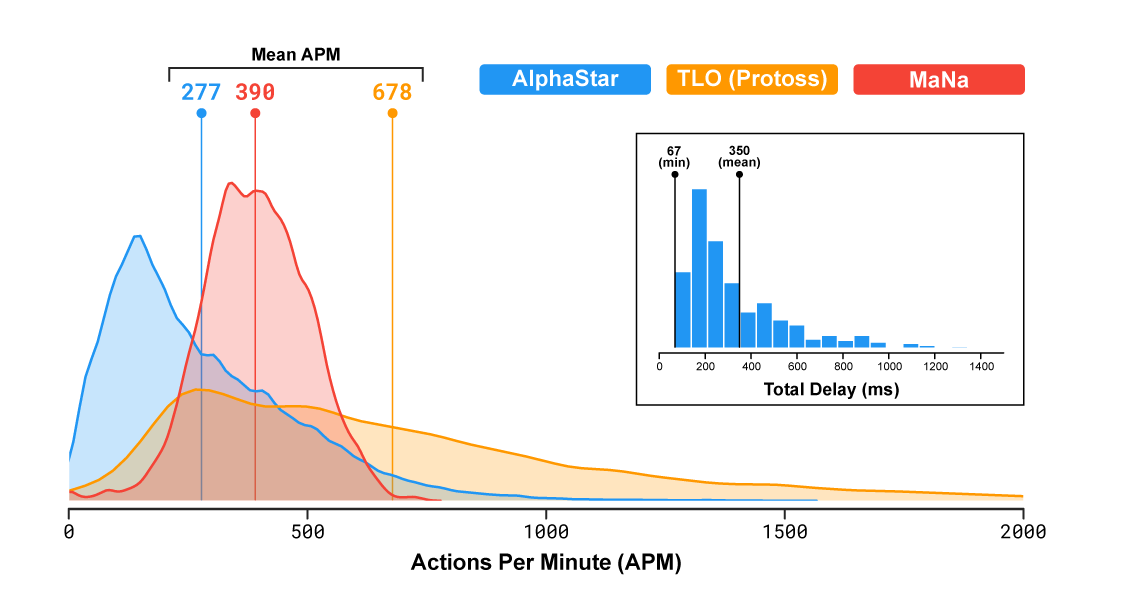
Коэффициенты (веса) нейронной сети каждого агента обновлялись с помощью обучения с подкреплением на основе игр с оппонентами, чтобы оптимизировать их для конкретной цели обучения. Правило обновления весов — это новый эффективный алгоритм обучения вне политичный алгоритм обучения с подкреплением с критикой 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 12. Распределение APM AlphaStar в матчах против MaNa и TLO, и общая задержка между наблюдением и действием.

В первом противостоянии против TLO и MaNa, AlphaStar взаимодействовал с игровым движком StarCraft'а через базовый 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

## Используемые методологии разработки и технологии

### Test-driven development

**Test-driven development, или TDD** (**разработка через тестирование)** — техника разработки программного обеспечения, которая предполагает сначала написание теста для нового кода, а затем самого кода с дальнейшем тестированием. [28]

**Цикл разработки** через тестирование:

1. Добавление теста.

Сначала нужно написать тест, в который входит описание ,где указывается какой функционал тестируется и каким должно быть поведение, и проверка.

1. Запуск теста.

Новый тест не должен проходиться, так как функционал не написан. Запуск проводиться для проверки работоспособности тестов.

1. Написание кода.

Далее пишется функциональность, которая была ранее описана.

1. Повторный запуск. Проверка функционала.

Запустив тест, программист убедиться, что нужный функционал работает и соответствует требованиям.

1. Рефакторинг.

На этом этапе можно спокойно править код и, запустив тесты, можно будет убедиться в его работоспособности или неработоспособности.

Важное замечание: написанные тесты не гарантируют, что тест действительно тестирует то, для чего он был разработан. Возможна такая ситуация, что тест проверяет не все ситуации, которые могут возникнуть.

**Плюсы технологии**:

* Этот метод позволяет избежать множества ошибок, особенно на ранних этапах. Также TDD препятствует возникновению дорогостоящих ошибок, которые могут возникнуть на поздних этапах разработки.
* Добавляя новую функциональность, риск испортить старый функционал очень мал. Чтобы этот риск был низким код должен быть полностью покрыт тестами, то есть для каждой функциональной единицы должен быть как минимум один тест. Это позволит при тестировании сразу увидеть, где ошибка.
* TDD способствует разработке более модульного, гибкого и расширяемого кода. Это связано с тем, что при этой методологии разработчику необходимо думать о программе как о множестве небольших модулей, которые написаны и протестированы независимо и лишь, потом соединены вместе. Это приводит к меньшим, более специализированным классам, уменьшению связанности и более чистым интерфейсам.
* Тесты могут использоваться в качестве документации. Хороший код расскажет о том, как он работает, лучше любой документации. Документация и комментарии в коде могут устаревать. Это может сбивать с толку разработчиков, изучающих код. А так как документация, в отличие от тестов, не может сказать, что она устарела, такие ситуации, когда документация не соответствует действительности — не редкость.

**Минусы технологии**:

* Существуют задачи, которые невозможно (по крайней мере, на текущий момент) решить только при помощи тестов. В частности, TDD не позволяет механически продемонстрировать адекватность разработанного кода в области безопасности данных и взаимодействия между процессами. Безусловно, безопасность основана на коде, в котором не должно быть дефектов, однако она основана также на участии человека в процедурах защиты данных. Тонкие проблемы, возникающие в области взаимодействия между процессами, невозможно с уверенностью воспроизвести, просто запустив некоторый код.
* Разработку через тестирование сложно применять в тех случаях, когда для тестирования необходимо прохождение функциональных тестов. Например, разработка интерфейса пользователя или СУБД, в которой работа будет зависеть от конфигурации сети.
* Требуется больше времени на разработку и поддержку
* Большое количество используемых тестов может создать ложное ощущение надежности, приводящее к меньшему количеству действий по контролю качества.
* Тесты сами по себе являются источником накладных расходов. Плохо написанные тесты, например, содержат жёстко вшитые строки с сообщениями об ошибках или подвержены ошибкам, дороги при поддержке. Чтобы упростить поддержку тестов, следует повторно использовать сообщения об ошибках из тестируемого кода.
* Уровень покрытия тестами, получаемый в результате разработки через тестирование, не может быть легко получен впоследствии. Исходные тесты становятся всё более ценными с течением времени. Если неудачные архитектура, дизайн или стратегия тестирования приводят к большому количеству непройденных тестов, важно их все исправить в индивидуальном порядке. Простое удаление, отключение или поспешное изменение их может привести к необнаруживаемым пробелам в покрытии тестами.

Данная технология разработки была выбрана по **следующим причинам**:

* Позволяет избежать ошибок на ранних и последующих этапах. Данный микро-сервис будет разрабатываться на javascript, из-за того что он динамически типизированный могут возникнуть различные ошибки, которые без тестов будут заметны только при запуске системы. Например, опечатки или передача аргумента не того типа, на такие ошибки динамически типизированные языки не обращают внимания.
* TDD способствует разработке более модульного, гибкого и расширяемого кода. Код, разделённый на модули, легче тестировать и эти тесты дают уверенность в том, что когда модули будут объединены в единую систему, то всё заработает.
* Тесты могут использоваться в качестве документации. Это значительно ускорит ознакомление новых программистов с системой.

### C4Model

C4Model представляет из себя несколько диаграмм: диаграмма контекста (Context diagram), диаграмма контейнеров (Container diagram), диаграмма компонентов (Component diagram), диаграмма классов (Class diagram). [29]

1. **Диаграмма контекста**.

**Рассматриваемая область**: программные системы и их взаимодействие, люди, взаимодействующие с разрабатываемой системой.

**Основные элементы**: программные системы, пользователи, взаимодействия пользователей и программных систем.

**Вспомогательные элементы:** Люди и программные системы напрямую связанные с разрабатываемой системой.

**Предполагаемая аудитория**: Все, включая людей, которые не являются разработчиками. Например, заказчик или инвестор.

Диаграмма не должна содержать информацию об используемых технологиях, это будет указано на следующей диаграмме.

На схеме должны быть показаны: программные системы в виде прямоугольников, пользователи в виде человечков, связи с описанием воздействия. Внутри каждого элемента сначала указывается название, затем ниже в квадратных скобках тип элемента(программная система, пользователь и т.д), и в конце описание. Ниже будет приведён пример диаграммы контекста для интернет-банкинга (см. рис 1).



Рисунок 2. Пример диаграммы контекста для интернет-банкинга

1. **Диаграмма контейнеров**

Под контейнером понимается то, что можно развернуть отдельно от других компонентов системы. Например, веб-приложение, база данных.

**Рассматриваемая область**: комплекс взаимодействующих контейнеров и программных систем.

**Основные элементы**: контейнеры в разрабатываемой системе.

**Вспомогательные элементы:** Люди и программные системы, взаимодействующие с контейнерами.

**Предполагаемая аудитория**: Разработчики, архитекторы ПО, вспомогательные сотрудники.

На этой диаграмме в каждом контейнере указывается используемая технология в строке с типом элемента следующим образом: [Container:<Используемая технология>]. На связях указывается способ, технология или при помощи чего будет происходить взаимодействие в виде: <Как взаимодействует>(<При помощи чего>). Ниже показан пример диаграммы контейнеров для интернет-банкинга(см. рис 2).



Рисунок 3. Диаграмма контейнеров для интернет-бакинга

1. **Диаграмма компонентов**

Для каждого контейнера нужно составлять эту диаграмму. На ней будет показано, из каких компонентов состоит контейнер и как они взаимодействуют. Также нужно указать взаимодействия с другими контейнерами.

**Рассматриваемая область**: один контейнер.

**Основные элементы**: компоненты контейнера.

**Вспомогательные элементы:** Люди, программные системы и контейнеры, взаимодействующие с данным контейнером и его компонентами.

**Предполагаемая аудитория**: Архитекторы и разработчики программного обеспечения.

Ниже показан пример диаграммы компонентов для контейнера API Application(см. рис 3).

****

Рисунок 4. Диаграмма компонентов для контейнера API Application

1. **Диаграмма классов**

Затем для каждого компонента строится диаграмма класса. Здесь показывается, как компонент будет выглядеть в коде, для этого можно использовать UML или другую подобную технологию. Я использовал диаграммы классов UML, об этом будет сказано ниже.

**Рассматриваемая область**: один компонент.

**Основные элементы**: Элементы кода (например, классы, интерфейсы, объекты, функции, таблицы базы данных и т. д.) В рамках компонента в области.

**Предполагаемая аудитория**: Архитекторы и разработчики программного обеспечения.

Данная технология разработки была выбрана по следующим причинам:

* Использование этих диаграмм позволяет структурировать информацию о разрабатываемом продукте в виде понятном разработчику, его команде и его заказчику (относится только к диаграмме контекста).
* Эти диаграммы позволяют программистам разных категорий понять суть работы системы, не углубляясь в детали.
* Нарисованные диаграммы позволяют увидеть систему целиком и отдельно рассмотреть какую-то её часть, что очень сложно сделать без диаграмм, просто заглядывая в код.

### Нотация ER-диаграмм Crow’s foot

Существуют две нотации дляER-диаграмм: нотация Чена и нотация Crow’s foot[11]. Далее я опишу обе нотации, и затем объясню, почему была выбрана нотация Crow’s foot.

**Crow’s foot (англ. Воронья лапа)** — нотация диаграмм отношений сущностей, где отношения показываются с помощью линий, с указанием названия связи над линией и количественных отношений на концах линии. Это одна из наиболее распространенных нотаций. [30]

В нотации Crow’s foot есть следующие элементы:

1. **Сущность**(Entity). Объект, который хранит в себе нужные вам данные (строки, числа, даты и так далее). Чтобы показать поле первичным ключом, нужно перед названием поставить звёздочку(\*).
2. **Отношения**(Relationship). Показывают связи между сущностями. На схеме показывается простой линией с названием связи над линией.
3. **Мощность**(Cardinality) и **модальность**(Modality). Под мощностью понимается, что, в сущности, может быть несколько экземпляров другой сущности. Под модальностью понимается, что, в сущности, может быть один или не одного экземпляра другой сущности. Чтобы показать мощность или модальность рисуется линия с указанием количества экземпляров на конце. Если количество сущностей равно 1, то рисуется вертикальная линия, если может быть равным 0, то рисуется кружок. Чтобы показать что количество экземпляров может быть больше одного нужно нарисовать воронью лапу, состоящую из 3 пальцев (поэтому нотация и называется «Вороньей лапой»)

Чтобы было проще понять, как элементы должны выглядеть, ниже будет картинка с примерами.



Рисунок 5. Примеры элементов нотации Crow’s foot

На рисунке будут показаны примеры сущностей и отношений. Рисунок сделан в программе “Visio 2016” и есть отличие в обозначении поля с первичным ключом. В Visio первичный ключ обозначается как квадрат с надписью «PK»(primal key). На картинке слева в прямоугольнике указан тип отношения, справа сущности и связь между ними.Чтобы рисунок не занимал слишком много места, только вначале будет показан пример сущностей со всеми полями, далее будет только сокращённая версия без свойств и полей.

**Нотация Чена(англ. Chen notaion)** — нотация в которой отношения показываются с помощью ромба, соединённого линиями с сущностями. [31]

В нотации Чена есть те же элементы, что и в нотации Crow’s foot, но есть отличия:

1. Сущность. На схеме изображается в виде прямоугольника с названием в центре. К нему подсоединяются свойства, которые показываются как овалы с названиями в центре. В нотации Crow’s foot свойства находятся внутри сущности и разделены линией.
2. Отношения. Изображаются ромбом, соединённым линиями с сущностями, Внутри ромба находится название связи.
3. Мощность и модальность. Указываются на линиях, которые подсоединены к ромбу. Вместо обозначений вроде не закрашенного кружка или «Вороньей лапы» линии используются числовое обозначение.

Помимо этого в рассматриваемой нотации дополнительные элементы, которых нет в Crow’s foot:

1. Есть ещё два вида сущности
   1. Слабая сущность (Weak entity) — объект, который не может быть однозначно идентифицирован только по его атрибутам. Существование слабого объекта зависит от другого объекта, называемого сущностью владельца. Идентификатор слабого объекта представляет собой комбинацию идентификатора объекта владельца и частичного ключа слабого объекта. Обозначается как прямоугольная рамка с названием сущности в центре.
   2. Ассоциативный объект (Associative entity) - объект, используемый в отношении «многие ко многим» (представляет собой дополнительную таблицу). Обозначается как прямоугольник с ромбом в центре, внутри которого находится название сущности.
2. Существует несколько видов атрибутов:
   1. Ключевой атрибут выглядит как обычное свойство, только название внутри подчёркивается.
   2. Атрибут частичного ключа (дискриминатор) (Partial key attribute (discriminator)) - атрибут, который в сочетании с ключевым атрибутом объекта владельца предоставляет уникальную идентификацию для слабого объекта. Этот элемент изображается как обычное свойство, в котором название подчёркивается пунктирной линией.
   3. Многозначный атрибут (multivalued attribute) - атрибут, который может иметь много значений (для него есть много разных значений в одном столбце таблицы). Многозначный атрибут изображается овальной рамкой с названием в центре.
   4. Производный атрибут (или вычисленный атрибут)(derived attribute (or computed attribute)) - атрибут, значение которого вычисляется (выводится) из других атрибутов. Полученный атрибут может или не может быть физически сохранен в базе данных. Данный элемент обозначается овалом, граница которого изображена пунктирной линией)
3. Есть второй вид отношений:
   1. Слабое (идентифицирующее) отношение(weak (identifying) relationship) — отношение, в котором дочерняя сущность зависит от родителя, и первичный ключ взят из родительской сущности.
4. Есть необязательное отношение. На рисунке изображается пунктирной линией.
5. Существует ограничения участия. Сущности могут участвовать в отношении полностью или частично.
   1. Полное участие. Означает, что каждая сущность участвует в отношении. Например, каждого студента обучает преподаватель.
   2. Не полное (частичное) участие. Означает, что не каждая сущность может участвовать в отношении. Например, преподаватель обучает студента, но могут быть преподаватели, которые не имеют учеников.

На следующей странице будут показаны примеры элементов нотации Чена.



Рисунок 6. Примеры элементов нотации Чена

Теперь сравним рассмотренные нотации. Нотация Чена имеет больше видов элементов, что позволит подробнее описать базу данных, но для небольших баз данных это лишнее. ER-диаграмма в нотации Чена будет больше по размеру и выглядеть она будет более сложной для восприятия.

Я выбрал нотацию Crow’s foot, потому что итоговая схема будет более компактной и простой, чем в случае использовании нотации Чена. Также не весь функционал нотации Чена мне будет нужен, и ещё при использовании нотации Чена нужно было бы позаботиться о более-менее красивом расположении свойств, что занимает определённое время, в случае с Crow’s foot таких проблем нет.

### UML

**UML** (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур. [32]

В этот язык входит множество диаграмм(чтобы не было неоднозначности с названиями в скобках будет указано название на английском языке):

* **Структурные диаграммы(Structure Diagrams):**
  + **Диаграмма классов (Class diagram).** Содержит в себе классы, их методы, поля и взаимосвязи.[33]
  + **Диаграмма компонентов (Component diagram).** Содержит в себе компоненты программной системы. Компонента – всё что можно развернуть (поставить, установить) отдельно от других частей системы (например, модуль, библиотека, веб-приложение и так далее).[34]
  + **Диаграмма композитной/составной структуры (Composite structure diagram).** Данная диаграмма показывает внутреннюю структуру и взаимодействие внутренних элементов класса.[35]
  + **Диаграмма развёртывания (Deployment diagram).** На этой диаграмме показываются компоненты, которые нужно установить, чтобы разрабатываемая система заработала.[36]
  + **Диаграмма объектов (Object diagram).** В этой диаграмме показывают экземпляры классов и отношения между ними в какой-то момент времени.[37]
  + **Диаграмма пакетов (Package diagram).** На этой диаграмме показываются пакеты, связи и зависимости между ними. Пакет – это группа моделей (классов, сущностей, случаев использования и т.д).[38]
* **Диаграммы поведения (Behavior Diagrams):**
  + **Диаграмма деятельности (Activity diagram).** Она отображает действия, которые описаны в диаграмме состояний.[39]
  + **Диаграмма состояний (State Machine diagram).** На данной диаграмме изображается конечный автомат.[40]
  + **Диаграмма вариантов использования (Use case diagram).** На этой диаграмме показываются актёры(люди, исполняющие определённую роль) и их действия на описываемую систему, которые изменяют состояние системы.[41]
  + **Диаграммы взаимодействия (Interaction Diagrams):**
    - **Диаграмма коммуникации/кооперации (Communication diagram)**. Содержит взаимодействия между частями композитной структуры или ролями кооперации. На этой диаграмме указываются отношения между объектами и не указывают время для измерения работы, как на диаграмме последовательности.[42]
    - **Диаграмма обзора взаимодействия (Interaction overview diagram)**. Данная диаграмма представляет из себя комбинацию диаграмм деятельности и диаграмм последовательности. Представляет из себя диаграммы последовательности, разбитые на отдельные диаграммы деятельности. С помощью этой диаграммы можно показать поток управления.[43]
    - **Диаграмма последовательности (Sequence diagram)** . На этой диаграмме показывается прецедент и всё, что нужно для его исполнения (объекты, актёров, жизненный цикл объектов). [44]
    - **Диаграмма синхронизации (Timing diagram)** . В этой диаграмме рассматриваются состояния, и их изменение с течением времени.[45]

Большую часть диаграмм я не использовал, так нет в этом необходимости. Далее будет сказано только про те диаграммы, которые были использованы. Я использовал следующие диаграммы:

* Диаграмма активностей или деятельности(activity diagram) для схемы алгоритма обработки гео-данных и диаграммы классов.
* Диаграмма классов. Она была использована при следовании концепции C4Model.

В UML и C4Model есть диаграмма компонентов, но они сильно отличаются. Диаграмма компонентов из UML показывает интерфейсы компонентов и также отличается содержимое контейнера, в нём нет описания. Подробнее диаграмма компонентов UML будет описана ниже.

**Диаграмма компонентов**

**Диаграмма компонентов** – диаграмма, показывающая компоненты разрабатываемой системы и их взаимодействия. [34]

В диаграмме компонентов изображаются:

* Компоненты. Они рисуются как прямоугольники содержащие:
  + Значок компонентного элемента. Ставится, если данный контейнер является элементом другого контейнера. Значок выглядит как прямоугольник с двумя меньшими прямоугольниками, выступающими с левой стороны.
  + Фраза «Component». Эта фраза указывается, если данный контейнер является стереотипным. Здесь под стереотипом имеется ввиду один из механизмов расширения (в UML есть ещё тэги и ограничения), который является специализированной моделью на основе которой можно создать другие компоненты.
  + Фраза «Subsystem». Указывается вместо прошлой фразы, если данный контейнер является подсистемой. Архитектор сам решает что является подсистемой, в UML это жестко не задано.
  + Интерфейсы. Требуемые интерфейсы показываются линией с полукругом на конце и названием интерфейса над линией. Предоставляемые интерфейсы обозначаются почти также как и требуемые, только вместо полукруга рисуется целый не закрашенный круг. Если один контейнер требует интерфейс «А», а другой его предоставляет, то их линии связи соединяются так, чтобы внутри полукруга требуемого интерфейса был круг предоставляемого интерфейса.
* Отношения
  + Зависимость. Показывается также как и в диаграмме классов. Стрелка зависимости должна выходить из гнезда требуемого интерфейса.

Ниже будут показаны примеры элементов диаграммы коспонентов.



Рисунок 7. Примеры элементов UML диаграммы компонентов

**Диаграмма деятельности**

**Диаграмма деятельности** – диаграмма, показывающая процесс и последовательность действий необходимых для его выполнения. [39]

Данный вид диаграмм очень похож на блок-схемы и имеет ту же функцию. В отличие блок-схем рассматриваемый вид более сложный, но у него больше возможностей:

1. С помощью жирных линий можно показать начало (разветвление) и окончание (схождение) ветвления действий. Это используется для изображения параллельных операций и дальнейшей синхронизации потоков.
2. Можно показать состояния объектов.
3. У каждой сущности участника есть своя дорожка, на которой видно, какое действие выполняет конкретная сущность, и на каком этапе.

Данный вид диаграмм был использован из-за возможности использовать дорожки. В алгоритме обработки гео-данных используются несколько функций, которые нужно было показать в удобном виде. Блок схемы не позволили бы мне этого сделать.

**Диаграмма классов**

**Диаграмма классов** – диаграмма, показывающая классы разрабатываемой системы, их взаимодействия, свойства и методы. [33]

В диаграмме классов изображаются:

1. Сущности (классы) в виде прямоугольников. В прямоугольнике содержится:
   * Название. Находится вверху посередине, выделено жирным шрифтом.
   * Поля и свойства. Находятся под заголовком, отделены чертой. Тип указывается после знака двоеточия, который стоит после названия.
   * Методы. Находятся под полями и свойствами, отделены чертой. После названия следуют скобки, внутри которых указываются аргументы через запятую. Тип возвращаемого значения указывается после двоеточия, которое стоит после скобок.

Поля и свойства могут быть:

* Публичными. Это означает, что к ним могут обратиться другие сущности. Обозначается «+» и ставится перед названием.
* Приватными. Это означает, что они будут недоступны другим сущностям, в том числе и тем, которые наследуют этот метод или свойство. Обозначается «-» и ставится там же, где и прошлый признак.
* Защищёнными. Это означает, что они будут недоступны другим сущностям, но при наследовании они могут быть доступны только наследнику. Обозначается «#» и ставится там же, где и другие признаки.
* Статическими. Это означает что данное поле или метод одинаковый для всех экземпляров данного класса. Название такого элемента подчёркивается.

1. Отношения:
   * Ассоциация – связь между сущностями, которая является общим случаем композиции и агрегации. Показывается обычной стрелкой.
   * Наследование – то же самое что и агрегация, но здесь могут включаться не все методы или поля и уровень доступа может измениться, при включении. Показывается линией с не закрашенным треугольником.
   * Реализация – отношение, при котором один класс реализует метод(ы) другого класса. Показывается пунктирной линией с не закрашенным треугольником.
   * Зависимость – отношение, при котором изменение спецификации класса-поставщика может повлиять на работу зависимого класса, но не наоборот.
   * Агрегация – отношение между классами, при котором один класс включает в себя всё содержимое другого. Показывается линией с пустым ромбом на конце.
   * Композиция – отношение между классами, при котором один класс включает в себя экземпляр другого. Показывается линией с закрашенным ромбом на конце.

При обозначении отношения указывается количественное отношение на концах линий: один к одному, один ко многим, многие ко многим. Ниже будут показаны примеры элементов диаграммы классов.



Рисунок 8. Отношения в диаграммах классов



Рисунок 9. Пример сущностей и связей между ними

## Используемые программные продукты и языки программирования

### Язык JavaScript

**JavaScript** — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией языка ECMAScript.[46]

Данный язык используется в основном для доступа и управления объектами различных приложений. Чаще всего применяется в браузерах в качестве языка сценариев для создания интерактивности на веб-странице.

Возможности языка:

* Динамическая типизация
* Сборщик мусора (автоматическое управление памятью)
* Прототипное программирование
* Функции являются объектами первого класса
* Автоматическое приведение типов
* Встраивание в код веб-страницы

В языке отсутствуют такие полезные вещи, как:

* стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода-вывода, базовых типов для бинарных данных;
* стандартные интерфейсы к веб-серверам и базам данных;
* система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

JavaScript состоит из нескольких частей:

* ядро (ECMAScript)
* объектная модель браузера (Browser Object Model или BOM (англ.))
* объектная модель документа (Document Object Model или DOM)

**ECMAScript** — это встраиваемый, расширяемый, не имеющий средств ввода-вывода язык программирования. ECMAScript является стандартом и основой для других языков: JavaScript, JScript и ActionScript. [47]

ECMAScript предоставляет пять основных примитивных типов:

1. Число
2. Строка
3. Логический (или булевый (англ. boolean))
4. нулевой (англ. Null)
5. неопределённый (англ. Undefined)

Есть составной тип: объектный (англ. Object).

Также есть стандартные для большинства современных языков инструкции (блок, условие, цикл и другие), операторы («.»(доступ к свойству), «++»(инкремент), «\*»( умножение) и другие).

Функции в JavaScript являются объектами и могут быть использованы как обычные переменные. В ECMAScript имеется два типа функций:

1. внутренние функции (например, parseInt)
2. функции, определённые в тексте программы.

В ECMAScript есть перегрузка функций и рекурсия, как и в других языках.

**Объектная модель документа (Document Object Model или DOM)** — интерфейс для программирования приложений, использующие HTML и XML документы. [46] Согласно DOM, документ может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции:

* генерация и добавление узлов,
* получение узлов,
* изменение узлов,
* изменение связей между узлами,
* удаление узлов.

**Объектная модель браузера (Browser Object Model или BOM (англ.))** — часть JavaScript, позволяющая скрипту управлять поведением веб-страницы. Данная часть JavaScript связывает ядро и DOM. [46, 48]

В качестве языка разработки был выбран JavaScript, потому что система трекинга поездок транспортного средства изначально была написана на нём. Разработка на JavaScript позволит переиспользовать код от разных частей системы трекинга, а также упростит администрирование.

# 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. Kun Shao, Yuanheng Zhu, Member, IEEE and Dongbin Zhao, Senior Member «StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning» [Online]. Available: <https://arxiv.org/pdf/1804.00810.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. 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.
17. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction.MIT Press, 1998.
18. 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
19. 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.
20. 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.
21. 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.
22. T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in International Conference on Learning Representations, 2016.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.
29. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint, 2017.
30. S. Levine and V. Koltun, “Guided policy search,” in International Conference on Machine Learning, 2013, pp. 1–9.
31. 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.
32. 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.
33. M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” Machine Learning Proceedings, pp. 157–163, 1994.
34. T. Ming, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” in Proceedings of the tenth International Conference on Machine Learning, 1993, pp. 330–337.
35. 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.
36. S. Sukhbaatar, A. Szlam, and R. Fergus, “Learning multiagent communication with backpropagation,” in Neural Information Processing Systems, 2016, pp. 2244–2252.
37. L. Marc, Z. Vinicius, and G. Audrunas, “A unified game-theoretic approach to multiagent reinforcement learning,” arXiv preprint arXiv:1711.00832.
38. 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.
39. 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.
40. 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.
41. Y. Wu and Y. Tian, “Training agent for first-person shooter game with actor-critic curriculum learning,” in International Conference on Learning Representations, 2017.
42. 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.
43. 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.
44. V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in International Conference on Machine Learning, 2010, pp. 807–814.
45. S. P. Singh and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” Machine Learning, vol. 22, no. 1, pp. 123–158, 1996.
46. 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.
47. 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.
48. 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
49. 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.
50. 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.
51. 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.
52. 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.
53. 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.
54. 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.
55. 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.
56. 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.
57. 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.
58. R. Coulom, “Computing elo ratings of move patterns in the game of Go,” in Computer Games Workshop, 2007.
59. 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.
60. 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.
61. 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.
62. 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.
63. 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.
64. 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.
65. S. M. Lucas and G. Kendall, “Evolutionary Computation and Games,” IEEE Computational Intelligence Magazine, vol. 1, no. 1, pp. 10–18, 2006.
66. 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.
67. 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.
68. 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.
69. 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.
70. 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.
71. 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.
72. 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.
73. 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.
74. N. Justesen, “Artificial Intelligence for Hero Academy,” Master’s thesis, IT University of Copenhagen, 2015.
75. 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.
76. 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.
77. 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.
78. 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.
79. 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.
80. 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.
81. T. Hong, K. Huang, and W. Lin, “Adversarial Search by Evolutionary Computation,” Evolutionary Computation, vol. 9, no. 3, pp. 371–385, 2001.
82. Niels Justesen, “Hero AIcademy.” [Online]. Available: https://github.com/njustesen/hero-aicademy
83. Robot Entertainment, “Hero Academy.” [Online]. Available: <http://www.robotentertainment.com/games/heroacademy/>
84. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II [Online]. Available: <https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii>
85. David Balduzzi, Karl Tuyls, Julien Perolat, Thore Graepel “Re-evaluating Evaluation” [Online]. Available: <https://papers.nips.cc/paper/7588-re-evaluating-evaluation.pdf>
86. Starcraft AI Competition [Online]. Available: http://www.cs.mun.ca/~dchurchill/starcraftaicomp/results.shtml

Приложение А. Техническое задание.

Введение

На

Назначение разработки

Данный

Основание для разработки

Так

Требования к программе или программному изделию

1. Архитектура

Требования к программной документации

1. Должны
2. Также

Стадии и этапы разработки

1. С

Порядок контроля и приемки

1. П

Приложение Б. Программная спецификация.

Документация

Ранее в основной части использовались перечисленные ниже диаграммы, поэтому здесь тут даны только ссылки на них.

В документацию входят:

* Диаграмма контекста. (см. рис )

Компоненты

При разработке микро-сервиса будут использоваться следующие программные продукты:

* PostgreSQL – база данных

Документация по программным комплексам:

* PostgreSQL – https://www.postgresql.org/docs/

Приложение В. Программная документация.

Диаграмма контекста

Рисунок .