**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**
**ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

# «Применение методов машинного обучения для анализа матчей многопользовательских стратегий реального времени»

Автор Смыкалов Владимир Павлович

Направление подготовки (специальность) Прикладная математика и информатика

Квалификация Бакалавр прикладной математики и информатики

Руководитель Ульянцев Владимир Игоревич, к.т.н., программист кафедры ИС

**К защите допустить**

Зав. кафедрой Васильев В. Н., проф., д.т.н.

“___”_____ 2016 г.

Санкт-Петербург, 2016 г.

Студент __Смыкалов В. П.__ Группа __М3437__ Кафедра __КТ__ Факультет __ИТиП__

Направленность (профиль), специализация: __Математические модели и алгоритмы в разработке программного обеспечения__

Квалификационная работа выполнена с оценкой _____

Дата защиты "____"_____20 ____г.

Секретарь ГЭК _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

**Факультет** *Информационных Технологий и Программирования*
**Кафедра** *Компьютерных технологий* **Группа** *M3437*
**Направление (специальность)** *Прикладная математика и информатика*
**Квалификация (степень)** *Бакалавр прикладной математики и информатики*

# З А Д А Н И Е
## НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

**Студент** *Смыкалов В. П.*
**Руководитель** *Ульянцев В. И., к.т.н., програтмист кафедры ИС Университета ИТМО*

**1. Наименование темы** *Применение методов машинного обучения для анализа матчей многопользовательских стратегий реального времени*

**2. Срок сдачи студентом законченной работы** *31 мая 2016 г.*

**3. Техническое задание и исходные данные к работе**
*Рассматривается задача предсказания победителя матча игры Dota 2 на основе выбранных игроками героев. Данная задача была рассмотрена во множестве научных статей. Необходимо придумать новый способ подготовки и обработки данных, которые впоследствии будут подаваться на вход алгоритмам машинного обучения. Продемонстрировать преимущества нового способа обработки данных.*

**4. Содержание выпускной работы (перечень подлежащих разработке вопросов)**
*1. Постановка задачи и краткое описание игры Dota 2*
*2. Получение и подготовка данных*
*3. Применение методов машинного обучения*

**5. Перечень графического материала (с указанием обязательного материала)**
*не предусмотрено*

## 6. Исходные материалы и пособия


1. *Conley, Kevin, and Daniel Perry. "How does he saw me? A Recommendation Engine for Picking Heroes in Dota 2." Np, nd Web 7 (2013).*
2. *Schubert, Matthias, Anders Drachen, and Tobias Mahlmann. "Esports Analytics Through Encounter Detection." MIT Sloan Sports Analytics Conference. MIT Sloan, 2016.*
3. *Johansson, Filip, and Jesper Wikström. "Result Prediction by Mining Replays in Dota 2." (2015).*
4. *Agarwala, Atish, and Michael Pearce. "Learning Dota 2 Team Compositions."*


## 7. Консультанты по работе с указанием относящихся к ним разделов работы

| **Раздел** | **Консультант** | **Подпись, дата** | |
|---|---|---|---|
| | | **Задание выдал** | **Задание принял** |
| Экономика и организация производства | | | |
| Технология приборостроения | | | |
| Безопасность жизнедеятельности и экология | | | |

**КАЛЕНДАРНЫЙ ПЛАН**

| **№№ п/п** | **Наименование этапов выпускной квалификационной работы** | **Срок выполнения этапов работы** | **Примечание** |
|---|---|---|---|
| 1 | Ознакомление с предметной областью | 11.2015 | |
| 2 | Изучение существующих методов решения поставленной задачи | 02.2016 | |
| 3 | Разработка парсера реплеев Dota 2 и подготовка данных | 03.2016 | |
| 4 | Применение методов машинного обучения на обработанных данных | 04.2016 | |
| 5 | Написание пояснительной записки | 05.2016 | |

**8. Дата выдачи задания** *1 сентября 2015 г.*

Руководитель _____

Задание принял к исполнению _____

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

# АННОТАЦИЯ
## ПО ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

**Студента** *Смыкалов В. П.*
( Фамилия, И., О. )

**Факультет** Информационных технологий и программирования

**Кафедра** Компьютерных технологий **Группа** *M3437*

**Направление (специальность)** Прикладная математика и информатика

**Квалификация (степень)** Бакалавр прикладной математики и информатики

**Наименование темы:** *Применение методов машинного обучения для анализа матчей многопользовательских стратегий реального времени*

**Руководитель** *Ульянцев В. И., к. т. н., программист кафедры ИС Университета ИТМО*
( Фамилия, И., О., ученое звание, степень )

**Консультант**
( Фамилия, И., О., ученое звание, степень )

**КРАТКОЕ СОДЕРЖАНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ И ОСНОВНЫЕ ВЫВОДЫ**

**объем** _____ стр., **графический материал** 0 стр., **библиография** _____ наим.

- Направление и задача исследований

Задачей исследования является задача предсказания победителя матча игры Dota 2 на основе выбранных игроками героев.

- Проектная или исследовательская часть (с указанием основных методов исследований, расчетов и результатов)

Написание парсера реплеев игры Dota 2, анализ эвристик, применение методов машинного обучения на обработанных данных

- Экономическая часть (какие использованы методики, экономическая эффективность результатов)

Не рассматривалась

- Характеристика вопросов экологии, техники безопасности и др.

Не рассматривалась

- Является ли работа продолжением курсовых проектов (работ), есть ли публикации

Данная работа не является продолжением курсового проекта, публикаций нет.

**Практическая ценность работы. Рекомендации по внедрению**

Написан парсер реплеев Dota 2, который может быть использован для извлечения информации для дальнейших исследований. Результаты данной работы могут быть использованы для построения более качественных алгоритмов предсказывания победителя матча Dota 2.

Выпускник _____
(подпись)

Руководитель _____
(подпись)

“____” _____ 2016 г.

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

С каждым годом киберспорт становится все популярнее среди молодежи и находит все больше признательности со стороны сообщества. Совсем недавно в России вышел приказ, в котором заявляется, что отныне киберспорт официально признан спортом в России. Призовые на киберспортивных соревнованиях едва-ли можно назвать игрушечными — призовой фонд последнего чемпионата мира по Dota 2 составил свыше 18 миллионов долларов. Об игре Dota 2 и пойдет речь.

Dota 2 — одна из самых популярных многопользовательский игр, игра жанра MOBA (букв. перевод. *многопользовательская онлайновая боевая арена*). В одной игре принимают участие 2 команды из 5 человек. Каждый игрок в самом начале игры выбирает себе уникального героя и играет им. Одна игра в среднем занимаем 40-50 минут. Герои в процессе игры получают опыт и золото, что позволяет им улучшать свои способности и покупать предметы для улучшения боевых характеристик. Победившей считается та сторона, которая первой уничтожит «трон» соперника. Набор из 10 героев (5 героев с одной стороны, 5 с другой) называется *пиком*.

Существует ряд научных статей, в которых изучается зависимость между пиком команд и победившей стороной — строятся различные модели, предсказывающие победителя исходя из пика команд. Например, в [1] была использована логистическая регрессия (англ. *logistic regression*) для предсказания победителя и были достигнуты весьма приятные и неожиданные результаты. Есть так же ряд работ, в которых ставиться задача предсказания победителя матча после прошествия определенного промежутка времени после начала игры [2—5].

В данной работе мы остановимся на изучении зависимости пика команд и победителя. Цель работы — реализовать парсер реплеев (англ. *replay*) Dota 2, который выделит признаки, позволяющие изменить и улучшить результат алгоритмов машинного обучения, используемых для предсказанию победителя матча игры Dota 2 исходя из пика команд.

В главе 1 вводятся основные определения и краткое описание механики игры Dota 2.

В главе 2 рассмотрены проблемы, возникающие при получении и обработки данных, а также методы их решения

В главе 3 рассмотрены результаты внедрения использования выделенных из реплеев признаков и особенностей

# ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ И КРАТКОЕ ОПИСАНИЕ ИГРЫ DOTA 2

## 1.1. Описание игры Dota 2

Dota 2 — игра с уникальным жанром, сочетающим в себе как стратегию реального времени, так и элемент экшн игр. Изначально DotA (Defence Of The Ancients) появилась как отдельная карта к игре Warcraft III: The Frozen Throne. Идея игры настолько понравилась игрокам, что спустя несколько лет, в 2009 году компания Valve наняла на работу создателя этой карты с целью создания полномасштабного игрового клиента на основе механики игры той карты. Игровой клиент вышел на свет в 2011 году под названием Dota 2 одновременно с чемпионатом мира по игре Dota 2 с призовым фондом 1 600 000 долларов. Это был первый в истории киберспортивный турнир с призовым фондом такого масштаба.

### 1.1.1. Герои

В одной игре Dota 2 участвуют 10 игроков. Пять из них представляют силы света (англ. *radiant*), другие пять представляют силы тьмы(англ. *dire*). В начале игры каждый из игроков обязан выбрать себе уникального героя из 111 (доступных на момент написания работы) и играть им всю игру. Каждый герой обладает целым рядом характеристик и показателей.

Здоровье — один из наиболее важных показателей. Если у персонажа заканчивается здоровье, он умирает и вновь появляется на свет лишь сперва определенный промежуток времени.

Мана — как и в любой другой игре, вторая составляющая показателей персонажа после здоровья. Практически все заклинания и способности в игре требуют ману.

Далее, как правило, выделяют 3 основных характеристики:

— сила,

— ловкость,

— интеллект.

Сила персонажа влияет на максимальное количество здоровья и скорость регенерации здоровья. Ловкость влияет на скорость атаки и количество брони. Интеллект влияет на максимальное количеством маны и скорость ее восстановления, а также немного усиливает наносимый заклинаниями урон.

У каждого героя один из этих 3 аттрибутов основной, тем самым герои делятся на 3 подгруппы: те, у которых основной аттрибут сила; те у которых основной аттрибут ловкость; те, у которых основной аттрибут интеллект. Количество основного аттрибута также влияет на наносимый персонажем урон. Например, если основной аттрибут у персонажа это ловкость, то получив дополнительную ловкость герой увеличит наносимый с обычных атак урон.

Не менее важны скорость передвижения, скорость атаки, наносимый урон, количество брони, но их название говорит само за себя и не требует излишних комментариев. Есть также более сложные характеристики такие как базовая скорость атаки и скорость поворота, но их влияние на персонажа весьма сложно и запутанно, поэтому я пропущу детальное описание всех характеристик. Подробное описание можно найти в [6].

У героев также есть такое понятие, как «дальность атаки». Это то расстояние, с которого герой может наносить обычный удар, без использования каких-либо заклинаний. Герои бывают ближнего боя, которым нужно находится вполтную к врагу, и герои дальнего боя.

Помимо этого, одним из показателей силы персонажа является количество полученного опыта и вместе с тем, уровень персонажа. Именно уровень персонажа по большей части влияет на количество времени, которое персонажу предстоит провести вне игры после смерти. При приобритении каждого нового уровня персонаж получает право усилить одну из своих способностей.

Основной источник получения опыта — убийство крипов и вражеских героев. В случае смерти крипа или вражеского героя все дружественные герои в определенном небольшом радиусе получают поровну опыта. Опыт героя никогда не теряется, даже при смерти. Максимальный уровень в игре 25-ый. После достижения этого уровня, опыт героя больше не накапливается.

Практически все герои обладают 4 способностями. Способности бывают как активные (которые можно использовать при нажатии на кнопку), так и пассивные (которые действуют всегда). Эти способности (заклинания) могут быть совершенно разными, от оглушения врага, телепорта по карте до аур которые восстанавливают здоровье или усиляют урон наносимый «с руки». Некоторые заклинания имеют настолько сложную механику, что многие опытные игроки в Dota 2 до сих пор путаются в некоторых деталях использования этих способностей.

Большинство активных способностей требуют ману для их использования, а так же имеют время перезарядки; их нельзя использовать слишком часто. Время перезарядки для каждой способности зафиксировано и может быть как 1-2 секунды, так и до 3 минут. Так же, есть способности, которые при активации добавляют какой-либо эффект к обычным атакам героя (например, увеличение урона)

Как правило, у героя есть 3 обычных способности и одна «ультимативная» (англ. *ultimate*). Ультимативную способность можно осваивать только на 6, 11 и 16 уровнях. Обычные способности можно осваивать с первых же уровней, однако есть ограничение, что уровень одного прокачки одного скилла не должен превосходить округленной вверх половины от уровня героя. Тем самым, на 5-ом уровне герой не может прокачать никакую способность больше чем на 3 уровня. Каждую обычную способность можно прокачать всего 4 раза, а ультимативную способность 3 раза.

Внимательный читатель может озадачиться — как так, всего уровней 25, а возможностей осваивать способности всего $4 + 4 + 4 + 3 = 15$. В оставшиеся 10 уровней герой может чуть поднять характеристики героя (вместо освоения способности, герой может получить +2 ко всем характеристикам героя.

Что делает игровой процесс уникальным и неповторимым — так это то, что среди всех способностей 111 героев нет даже 2 одинаковых! Некоторые способности разных героев отлично сочетаются друг с другом, что делает некоторые связки героев крайне сильными.

### 1.1.2. Карта

Все игры происходят на одной и той же карте, которая уже много лет не подвергается глобальным изменениям. Раз в несколько месяцев иногда выходит обновление игры, которое меняет положение пары тропинок на карте и подобные незначительные изменения. Однако глобально карта никогда не менялась.

Карта представляет из себя 3 линии: центральную, верхнюю и нижнюю. На каждой изначально стоит по 3 защитных башни у каждой стороны.

Цель игры — первым уничтожить «трон» соперника, хорошо защищенное здание на его базе. Раз в 30 секунд по каждой линии с каждой стороны выходят крипы (англ. *creeps*) — слабые неуправляемые юниты, которые идут в сторону вражеской базы и помогают сносить башни. Герои могут убивать

вражеских крипов и получать за это золото и опыт. Большое количество опыта влечет за собой большой уровень героя, в то время как большое количество золота позволяет покупать предметы, о которых речь будет чуть позже.

Карта условна поделена на 2 половины рекой, по каждую сторону от которой находится лес, в котором можно обнаружить лагеря нейтральных крипов. Нейтральные крипы стоят в лесу и никак не помогают сносить башни врага, однако их убийство можно получить дополнительные золото и опыт.

Несмотря на то, что карта не симметрична принято считать, что она сбалансирована и у героев одной стороны нет явного преимущества.

### 1.1.3. Предметы

Помимо большого разнообразия героев и способностей, в игре так же есть свыше 100 различных покупаемых предметов. Предметы, как и способности героев, могут быть как активными (которые имеют некоторый эффект при нажатии на кнопку), так и пассивными (которые просто усиливают некоторые характеристики героя). Герой ограничен только 6 слотами под предметы. Купить можно неограниченное число предметов, но одновременно персонаж может носить только 6.

Предметы бывают совершенно разные по стоимости, самый дешевый стоит 50 игровых монет, в то время как самый дорогой стоит 7195 (что более, чем в 100 раз больше!). Как правило, в игре соблюдается концепция, что чем дешевле предмет, тем лучше у него соотношение цена/качество. В связи с этим в ранние этапы игры игроки часто покупают более дешевые предметы, которые потом продают. Продасть предмет можно за половину стоимости.

Основной источник заработка золота в игре это убийство крипов и вражеских героев. Однако, в случае убийства крипов золото получает только тот герой, который нанес последний удар. В связи с этим, у хороших игроков вырабатывается навык не только добивать вражеских крипов на линии, но и своих. Нанесение последнего удара по своим крипам не приносит никакой прибыли, однако это лишает соперников потенциального золота, которое можно было бы получить за убийство этого крипа.

Покупаемые предметы нельзя передавать (кроме малого числа), тем самым один герой не может потратить золото, чтобы купить предмет другому герою.

Также, золото прибавляется пассивно со скоростью 5 игровых монет за 3 секунды, однако эта скорость крайне мала. Считается, что если игрок хорошо добивал крипов на линии, с хорошим счетом отыграл игру, то в среднем он будет получать свыше 600 игровых монет в минуту. Однако, каким-бы хорошим не был игрок, такой показатель невозможно получить с первых же минут игры. Речь идет про среднее количество золота в минуту, посчитанное после конца матча.

При смерти герой теряет золото, а те герои которые участвовали в его убийстве получают золото. Чем сильнее был герой относительно врагов, тем большую награду дадут за его убийство. После смерти герой имеет право досрочно выкупиться в игру — заплатить дополнительный штраф, но зато появиться в игре. Выкупаться можно не чаще, чем раз в 6 минут.

### 1.1.4. Роли в игре

Как правило, хорошие игроки в самом начале игры распределяют роли между друг другом. Эти роли никак жестко не прописаны игровой механикой, однако так получатеся, что если все герои будут поровну получать опыта и золота, то такая стратегия проиграет той, в которой вся ставка сделается на 1-2 героев из 5. Как правило, выделяют 4 роли: мид (одиночная позиция на средней линии), хард (одиночная позиция на сложной линии), керри (англ. *carry*, одиночная позиция на легкой линии) и 2 саппорта (англ. *support*). Сложной линией для сил света считается верхняя линия, а сложной для сил тьмы считается нижняя линия. Профессиональные команды часто распределяют геройв по линиям 1-1-3, где 3 идут на легкую линию (поэтому она так и называется), а на сложной линии наоборот получается что 1 герой стоит против троих.

Основная задача керри-героя получить как можно золота, купить на них как можно больше нужных предметов и за счет этого вносить много урона в сражениях. Как правило, герои которые выбираются на керри позицию имеют такой набор характеристик и способностей, что делает их слабыми на раннем этапе игры и сильными на позднем этапе.

Основная задача мид-героя в целом совпадает с задачей керри-героя, однако на средней линии герои обычно стоят 1 на 1, в следствии чего мид-герой не может получать так много золота как керри-герой (ведь керри-героя обычно поддерживают еще 2 саппорта). Однако, герои на средней позиции

как правило, получают больше всего опыта (так как герою на средней линии не приходится делиться опытом с еще 2 саппортами). На эту позицию часто выбирается герой, который очень силен на средней стадии игры и имеет хороший потенциал в поздней стадии игры.

Основная задача героя в сложной линии — получить как можно больше опыта и доставить как можно больше дискомфорта керри вражеской команды. Как правило, эти герои получают мало золота на ранних этапах игры, однако они могут оказаться полезными для команды за счет раннего получения ультимативной способности. На эту позицию часто выбираются те герои, которым крайне важно быстрое получение опыта (за счет возможности поскорее освоить некоторые способности, которые потом окажутся решающими в сражении), однако те, которые не сильно зависят от получаемого золота. Но также, подходят и те герои которые за счет своих способностей и характеристик могут спокойно чувствовать себя находясь на одной линии с 3 вражескими героями.

Задача саппортов — предоставить максимальное пространство и возможность для «фарма» (получения золота и опыта) керри и мид героев. Однако, на этом задачи саппортов не заканчиваются. По факту, у них гораздо больше задач, чем у героев любой другой позиции. Герои на саппорт позициях, как правило, почти не получают золота, поэтому под эту позицию чаще всего выбираются те герои, боевой потенциал которых не сильно зависит от предметов. Саппорты в игре должны проводить разведку, следить за положением и действиями вражеских героев. Несмотря на то, что с виду может показаться, что герои на саппорт позициях почти не вносят никакого влияния в исход сражений и матча в целом, это заблуждение. Саппорт-герои самые сильные персонажи на ранней стадии игры, их грамотные действия в самой ранней стадии игры могут решить исход всего матча.

Как правила, в команде есть один мид-герой, один керри, один герой в сложной линии и 2 саппорта. Однако, это не всегда так. Например, некоторые герои за счет своих способностей и характеристик могут с первых же минут получать золото и опыт «в лесу», убивая нейтральных крипов. Так же, иногда команды делают распределение по линиям 2-1-2, что чуть разгружает нагрузку на героя в сложной линии, но и усложняет ситуацию для героя в легкой линии.

### 1.1.5. Dota 2 в киберспорте

Одновременно с выходом игрового клиента Dota 2, компания Valve анонсировала турнир The International. Анонс этого турнира поверг всех в шок — ведь призовой фонд турнира был свыше 1 миллиона долларов. Никогда ранее не проводились киберспортивные соревнования с призовым фондом такого размера и масштаба. Первыми победителями турнира The International оказалась команда Natus Vincere (что с латыни переводится как «рожденные побеждать»). Вероятно именно эта победа и уверенное участие на нескольких последующих турнирах сделало команде Na'Vi одну из самых больших фанбаз. Фанатов этой команды можно найти по всему миру, от Сиэтла в Америке до Манилы в Филиппинах.

Турнир всем понравился (действительно, кому он мог не понравиться) и отныне проводится раз в год, летом. Было проведено уже 5 турниров The International. На последних турнирах компания Valve ввела дополнительную возможность для зрителей. За небольшую плату они могли получить некоторые косметические вещи в игре (ни как не влияющие на геймплей), а 25% этой платы переходило напрямую в призовой фонд турнира. Этой, с виду простой схемой, призовой фонд турнира поднялся с $1,6$ миллиона долларов до 18 в прошлом году!! Вероятно, эта схема снимает вопросы, откуда у компании средства и желание проводить турнир с таким большим прозовым фондом — ведь оставшиеся 75%, собранных с косметических предметов направляются прямиком в компанию.

Если раньше участие в соревнованиях по Dota можно было считать исключительно как хобби или увлечение, то на данный момент таких участников почти не осталось. Сейчас все призовые места занимают профессиональные команды — команды, у которых подписан контракт с киберспортивными организациями и которые получают зарплату за то, что играют в Dota 2. Однако, как можно было бы предположить, высокая зарплата игрока не делает его хорошим игроком, выигрывают только те команды, у которых хорошее слаженное командное взаимодействие и личные умения игроков на высоком уровне.

## 1.2. Постановка задачи

В статье [1] авторы задались задачей предсказывать по пику команд победителя матча. То есть, единственная информация, которую они извлекали

— это набор героев, которыми был сыгран матч и исход этого матча. Авторы утверждают, что смогли добиться процента предсказаний свыше 60%. Стоит отметить, что одной лишь статистикой вопрос на эту задачу не решить, так как различных выборок 10 героев из 111 крайне много. Такой процент предсказаний с одной стороны показывает, что грамотный выбор героев с одной стороны очень важен, но и с другой стороны не является решающим фактором.

Цель данной работы — повторить эксперимент и придумать методологию улучшения полученного результата. Авторы статьи использовали алгоритм логистической регрессия для предсказания победителя матча. В данной работе будет использоваться ровно этот же метод машинного обучения. Однако, будет показано как именно можно извлекать больше информации из записей игр и как именно ее использовать для лучшего предсказания результатов матча (все еще исходя только из пиков).

Задача — реализовать парсер записей игр, который извлекает доплнительную информацию, которую можно использовать для улучшения качества входных данных подаваемых алгоритму машинного обучения.

# ГЛАВА 2. ПОЛУЧЕНИЕ И ПОДГОТОВКА ДАННЫХ

## 2.1. Получение записей игр

Прежде чем писать парсер записей игр, нужно подготовить данные.

Первым препятствием на пути подготовки и анализа данных стало отстутствие открытого API для выкачивания из интернета записей прошедших игр. Ранее компания Valve предоставляла API для получения ссылки на реплей (англ. *replay*, запись игры), однако в 2013 году они убрали это из открытого доступа. Теперь используя их API можно получить только общие результаты матча, но не сам матч.

Эта проблема уже может оказаться крайне неприятной для тех, кто хочет программно анализировать большое число реплеев, однако я смог обойти ее. Чтобы обойти эту проблему мною был разработан скрипт на языке autoit[7]. Этот скрипт моделировал человека, нажимающего на клавиатуру и мышку, который последовательно скачивал один реплей за одним.

Скрипт брал список матчей из «very high skill bracket» с сайта dotabuff[8]. Скрипт моделировал ручное копирования с страницы dotabuff информации о матчах, запускал скрипт на языке python, который вытаскивал список идентификаторов матчей. После этого скрипт переключался в игровой клиент Dota 2 и моделировал там последовательность нажатий на кнопки, которые приводят к загрузке реплея по идентификатору.

Скорость скачивания реплеев составила примерно 1 реплей за 20 секунд. После каждого запуска скрипт стабильно скачивал 100-200 реплеев. За несколько запусков этого скрипта удалось скачать более 1000 реплеев. Отмечу, что авторы [2] тестировали свои разработки всего лишь на 412 реплеях.

Реплеи содержат очень много информации, а потому занимают не мало места на жестком диске. Отмечу, что каждый реплеей занимает в среднем 30-40 мегабайт, поэтому хранение даже 1000 реплеев уже занимает огромное место. А именно, 30-40 гигабайт данных, которые еще нужно распарсить и вытащить нужную информацию.

## 2.2. Парсинг реплеев

### 2.2.1. Начальная стадия

К сожалению, как и в ситуации со скачиванием реплеев, Valve не предоставили никаких утилит для парсинга своих же реплеев. Но, к счастью, нашлись люди, которые сделали опенсурсные библиотеки, которые позволяют

выдергивать хоть какую информацию. Из наиболее известных – clarity[9] и manta[10]. Библиотеки по функционалу похоже, я предпочел использовать manta для своих исследований.

Следующая проблема, которая возникает — нигде нет никаких документаций по поводу использования этих библиотек а так же о внутреннем устройстве реплеев Dota 2. Единственный способ продвинуться в этом направлении — найти людей, создали эти библиотеки, или же активно использовали и общаться с ними напрямую. Сообщество тех, кто работает с парсингом реплеев Dota 2 мало, но мне посчастливилось найти опытного программиста в этих кругах.

Сказать, что реплеи содержат удобно структурированную информацию нельзя. Полезную информацию приходиться извлекать из самых разных уголков реплея. Неудобств добавляет тот факт, что часть данных (например скорость передвижения героя) просчитывается в клиенте по ходу игры и эта информация не записана в реплеях.

Стоит отметить, что существующие библиотеки дают крайне сырые выходные данные. К примеру такая простая задача, как «проследить точные координаты героя в зависимости от времени» уже может поставить новичка в тупик. Сам я с ней смог справится не сразу, ее решение не так тривиально и без опыта написание парсеров реплеев Dota 2 его сложно придумать.

Так же стоит отметить, что не малое время при написании парсера заняли исследования «что можно легко извлекать из реплеев? как извлекать, что нельзя, а что вообще невозможно». Эти исследования сильно различаются от исследований вида «изучить устройство нового языка программирования». К примеру, основы языка Go я освоил за полчаса, а вот исследования по возможностям извлечения информации из реплеев Dota 2 длились недели. Но, к счастью, я успел закончить все эти исследования во время, получив ответы почти на все интересующие меня вопросы относительно парсинга реплеев Dota 2.

### 2.2.2. Общая структура парсера

Как только были закончены исследования возможностей извлечения информации из реплеев Dota 2, появилась возможность перейти к самому написанию парсера реплеев.

Одно из первых естественных желаний — запомнить как можно больше информации о героях на карте в различные моменты времени. Несмотря на то, что в игре время выглядит непрерывным (игра в режиме реального времени, а не пошаговая), в реплеях Dota 2 прослеживается минимальная единица времени, тик. Опытным путем выяснилось, что 30 тиков составляют 1 секунду. Поэтому, в первую очередь был написан код, который в каждый момент времени (тик) сохранял в памяти информацию обо всех героях на карте.

Совершенно всю информацию нельзя запомнить (части нет в реплеях и просчитывается на стороне клиента, часть менее актуальна). Было решено запоминать следующую информацию:

— координаты героя на карте
— уровень героя
— количество полученного золота
— количество полученного опыта
— наносимый урон с руки
— текущее здоровье
— максимальное здоровье
— текущая мана
— максимальная мана

Стоит отметить, что для более точного анализа можно также сохранять информацию о кулдаунах способностей героя, предметы героя, а также кулдауны предметов, но в данной работе эта информация не обрабатывалась.

Практически единственное, что остается для анализа – это огромный Combat Log, в котором записана инфомарция о всех использованиях способностей/предметов, ударов и получении урона.

При разработке в тупик может поставить тот факт, что идентификаторы сущностей в Combat Log-е не соответствуют идентификаторам игроков в матче или же идентификаторам сущностей героев на карте. Однако, я решил эту проблему путем рассмотрения первых записей о нанесении урона и простым анализом/перебором возможных вариантов.

Хоть немного разобравшись с устройством CombatLog-а можно ставить более глобальные цели. Одной из таких целей я поставил себе задачу поиска и выделения сражений. Под сражением подразумевается столкновение нескольких героев противоположных команд с нанесением урона и примене-

нием способностей с целью убить или как-либо помешать героям противоположной команды. Важно отметить, что здесь за сражение не считается так называемый «фарм крипов», учитывается лишь столкновения героев противоположных команд.

Само понятие это интуитивно крайне простое. Человек, имеющий опыт в Dota 2 без проблем просматривая реплей сможет указать несколько сражений на нем. Однако выявить программно подобные сражения крайне сложно. Для этого был мною придуман и реализован алгоритм нахождения сражений.

В [2] был описан некоторый алгоритм поиска столкновений, однако мною была придумана и реализована иная схема.

### 2.2.3. Алгоритм нахождения сражений

В первую очередь необходимо было определить какую-то жесткую грань снизу — какое столкновение героев считать сражением, а какое считать незначительной встречей. Было решено за сражения считать только те столкновения, в которых умер хотя бы 1 герой.

Алгоритм просматривает все записи в журнале боя (англ. *Combat Log*) и, найдя событие смерти героя, начинет анализ столкновения. Создается бинарный массив с флажками, какие герои на данный момент участвуют в сражении. В этом массиве помечается что умерший герой, а также те, кто участвовали в убийстве этого героя (получили assist) как те, кто уже участвуют.

После этого алгоритм проходит назад и вперед по записям из журнала боя в поисках других записей, в которых участвуют учавствующие в сражении герои. Все герои из подобных записей считаются учавствующими в сражении и так продолжается до тех пор, пока есть записи о применении способностей о нанесении урона героями противоположных сторон, участвующих в сражении.

Тут возникает вопрос, а как далеко искать записи в журнале боя о нанесении урона или применении способностей. Экспериментальным путем была поставлена и проверена константа в 5 секунд: если за 5 секунд не появилось ни одной записи о нанесении урона, то считалось что сражение окончилось. Стоит отметить, что в игре в подобной ситуации сражение может продолжаться — например происходит затяжная напряженная погоня по всей карте без возможности нанесения урона догоняющей стороной. Однако в таком слу-

чае нет однозначного ответа, считать ли эту погоню продолжением сражения и вообще за сражение.

Алгоритм был вручную протестирован на ряде реплеев, недочетов или неточностей выявлено не было. Однако, были найдены некоторые нюансы, такие как:

— стоит игнорировать записи о восстановлении здоровья героев (речь про активируемые способности и потребляемые предметы)
— стоит игнорировать ситуации, когда герой сам себе наносит урон

Разобравшись с подобными тонкостями, алгоритм заработал точнее и других проблем не было замечено.

Научившись выделять сражения в матче возникает вопрос, как именно их дальше анализировать, но об этом в следующей секции.

### 2.3. Выделение дополнительных признаков из записей игр

Научившись находить сражения в игре возникает естественное желание анализировать их. Одним из первых естественных шагов является определения победителя в сражении (radiant или dire). Однако, к сожалению, ответ на этот вопрос нельзя дать однозначно и человеку смотрящему реплей. Бывают как сражения с очевидным победителем, так и нет. Например ситуация, в которой с одной стороны умер керри-герой, а с другой строны 2 саппорта может быть расценена по разному в зависимости от этапа игры и кучи прочих факторов.

Однако, это не мешает ввести некоторые эвристики, которые помогут определить победителя в сражении. Эвристики могут рассматривать изменения различных характеристик героев, таких как количество золота, количество опыта, здоровье персонажа и многое другое.

В своей эвристике я рассматривал изменение полученного золота героем, изменение полученного опыта, изменение количества здоворья и маны. Золото и опыт были взяты с большим весом чем изменения здоворья и маны, так как на почти всех стадиях игры (кроме стадии глубокой поздней игры, когда все герои достигли максимального уровня), они считаются в приоритете.

Для каждого героя был посчитано число очков $points_i$, как много $i$-ый герой приобрел в сражении. Была использована следующая формула:

$$points_i = (newExperience_i - oldExperience_i) +$$
$$+\frac{3}{2}(newNetworth_i - oldNetworth_i) +$$
$$+\frac{1}{4}(newHealth_i - oldHealth_i) +$$
$$+\frac{1}{5}(newMana_i - oldMana_i)$$

Где *experience* – опыт героя, *networth* – суммарное количество заработанного золота, *health* – здоровье персонажа, *mana* – мана персонажа.

Исход сражения вычислялся как разность суммы количества очков по всем героям сил света и по всем героям сил тьмы:

$$points = \sum_{i \in radiantheroes} points_i - \sum_{i \in direheroes} points_i$$

В итоге от каждого сражения я смог получить число — как много преимущества одна команда получила по сравнению с другой.

Многочисленные эксперименты показали, что в случае успешно выигранной драки в средней стадии игры, командна получает несколько тысяч очков преимущества.

На этом часть парсинга доты можно заканчивать и начинать думать о том, какую именно информацию хочется извлекать, чтобы улучшить результаты предсказаний алгоритмов машинного обучения. Тут есть огромный набор возможностей, что именно делать, но сложно найти именно то, что может улучшить качество предсказаний.

Стоит отметить, что с получившимся кодом парсера можно без проблем проводить практически любые исследования. Например, отслеживать перемещения героев, то как часто они используют заклинания для фарма крипов и многое другое. По началу было желание оценивать качество игры игрока — а именно насколько удачно была использована способность. Мною было проделано много работы в этом направлении, но, к сожалению, они не привели к успехам, поэтому подробно рассказывать в этой работе я о них не буду.

Вернемся к вопросу, как можно улучшить долю правильных предсказаний исходов игр. Мною была придумана следующая схема: давайте оценим, насколько сильно одна команда переиграла другую. В случае, если преиму-

щество было явным, стоит в обучающей выборке взять этот матч с большим весом, а те игры в которых преимущество было менее явным считать с меньшим весом.

Теперь нужно научиться понимать, насколько равной была игра. Здесь снова появляется огромный пулл различных эвристик, которые можно использовать. Например, можно оценивать время игры, счет и многое, многое другое.

Было рассмотрено несколько эвристик, лучше всего себя показала эвристика «просуммируем значения исходов сражений за первые 20 минут игры». Стоит отметить, что эта схема отличается от простой эвристики «посмотреть на разницу количеств полученного опыта или золота по прошествии 20 минут». Бывают игры, в которых у одной команды может лучше получаться с фармом на линии и леса, но которая из-за неудачной комбинации героев проигрывает сражения команде врага.

Если за первые 20 минут одна команда сильно переиграла другую и в последствии выиграла игру, то можно считать, что матч закончился с большим перевесом в одну сторону, а значит один пик оказался сильно сильнее другого.

# ГЛАВА 3. ПРИМЕНИЕ МАШИННОГО ОБУЧЕНИЯ

Разобравшись с парсингом реплеев Dota 2 и достав информацию из более 1000 реплеев можно перейти к анализу полученных данных, используя алгоритмы машинного обучения. В данной работе данные были протестированы логистической регрессией и нейронной сетью, более подробно я расскажу про логистическую регрессию (так как модель более простая для описания, а цель работы не подобрать алгоритм машинного обучения, а показать новый способ обрабатывать входные данные).

В первую очередь нужно определиться, как именно по пику героев будет строится соответствующая точка в многомерном пространстве (на которые потом будет навешан алгоритм логистической регрессии). В статье [1] была использована следующая технология:

— пронумеруем всех героев от 0 до 110
— если герой с номером $id$ играет за radiant (силы света), присвоим $x_{id} = 1$
— если герой с номером $id$ играет за dire (силы тьмы), присвоим $x_{id+111} = 1$

В качестве выходных данных использовалась 1, если выиграли radiant и 0, если dire. Тем самым, в этой схеме мы имеем дело с 222-мерным пространством.

Был придуман и использован другой способ представления:

— пронумеруем всех героев от 0 до 110
— если герой с номером $id$ играет за radiant (силы света), присвоим $x_{id} = 1$
— если герой с номером $id$ играет за dire (силы тьмы), присвоим $x_{id} = -1$

В этом случае получается вдвое меньшая размерность пространства, что ускоряет время работы алгоритмов машинного обучения не менее, чем в 2 раза. Хоть и скорость обработки этих данных достаточно мала по сравнению со скоростью алгоритма для извлечения данных из 1000 реплеев, это ускорение все равно оказалось приятным.

Разобравшись, как именно мы будем подавать входные данные для проведения эксперимента в [1], можно начать. На данных, собранных с более 1000 реплеев был запущен алгоритм логистической регрессии с кросс-валидацией. Для кросс-валидации выборка точек разделялась на обучающую и тестирующую в отношении 4:1. Внутри каждой итерации кросс-валидации был посчитан процент правильных предсказаний исходов матчей. В конце за результат я считал средний процент по всем итерациям.

Так как в данном случае соотношение матчей в который победили силы света к матчам, в которых победили силы тьмы примерно равное, можно оставить подобную оценку с процентом угаданных исходов матчей, нет необходимости прибегать к использованию $F$-меры.

На запуске логистической регрессии без каких-либо изменений (бралась информация исключительно по пику команды и результату) был получен результат $57,46\%$ правильных предсказаний. Стоит отметить, что это уже достаточно впечатляющий результат!

Теперь, давайте внесем веса, полученные из подробного анализа реплеев. Самый простой способ сделать это — дважды добавлять точку в обучающую выборку, если матч в этой точке был односторонним. Подобный трюк крайне удобен тем, что в этом случае нет необходимости переписывать алгоритм логистической регрессии, чтобы он работал с весами точек. Этот трюк можно расширить, и брать не только веса 1 и 2, по уже даже эти веса дали ощутимый вклад в процент предсказаний.

С таким преобразованием обучающей выборки был получен результат $60,74\%$. Как можно заметить, это более чем на 3 процента лучше результата без дополнительных модификаций, что без всяких сомнений не может не радовать!

Однако стоит заметить, что полученный результат хуже того, что был получен в [1], но это можно легко объяснить тем, что использовалась куда меньшая база реплеев. В [1] были проанализированы лишь результаты игр, но ни как не сами реплеи, что снимает жесткие ограничения на количество данных, которые можно обработать.

Но все же, данная технология позволяет улучшить качество предсказаний, что и требовалось получить.

Для получениях более хороших результатов следует протестировать эти методы на больше базе реплеев Dota 2. Так же, чем больше база реплеев, тем более точно можно выявить, какие именно эвристики работают в данной области, а какие нет. Однако про некоторые эвристики (такие как та, что была использовна в этой работе) интуитивно понятно, что они должны работать, что и было продемонстрировано. Но про эвристики вроде выводов на основе длительности матча стоит лишь догадываться и без большой базы реплеев нельзя будет с уверенностью сказать, работают они или нет.

Так же стоит заметить, что важно, чтобы все реплеи из базы были из одного патча Dota 2. Каждые несколько месяцев создатели игры выпускают новое обновление, которое немного меняет баланс героев, чуть изменяет их характеристики и прочее. В одном патче какой-то пик мог быть сильнее другого, а с выходом патча наоборот.

# ЗАКЛЮЧЕНИЕ

Был создан скрипт, моделирующий поведение человека и выкачивающий реплеи Dota 2.

Был написан код, позволяющий вытягивать из реплеев много разной информации. Разработанный код позволяет легко извлекать любую понадобившуюся информацию из реплея, что несомненно облегчит продолжение исследований в этой области.

Был придуман и реализован алгоритм нахождения сражений, который отлично себя показал на многих реплеях.

Был придуман способ, как вытягивать большее количество информации (чем просто пик и победитель), с помощью которого были улучшены результаты предсказаний более, чем на 3 процента.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1 *Conley K.*, *Perry D.* How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2.

2 *Schubert M.*, *Drachen A.*, *Mahlmann T.* Esports Analytics Through Encounter Detection.

3 *Johansson F.*, *Wikström J.* Result Prediction by Mining Replays in Dota 2.

4 *Agarwala A.*, *Pearce M.* Learning Dota 2 Team Compositions.

5 *Yang P.*, *Harrison B.*, *Roberts D. L.* Identifying Patterns in Combat that are Predictive of Success in MOBA Games.

6 Dota 2 Wiki. — URL: `http://dota2.gamepedia.com/Dota_2_Wiki`.

7 AutoIt Home Site. — URL: `https://www.autoitscript.com/site/`.

8 Dotabuff. — URL: `https://github.com/dotabuff/manta`.

9 Clarity, replay parser, written in Java. — URL: `https://github.com/skadistats/clarity`.

10 Manta, fully functional replay parser, written in Go. — URL: `https://github.com/dotabuff/manta`.

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЕ КОДЫ РАЗРАБОТАННЫХ ПРОГРАММ

Листинг А.1 – Исходный код скрипта, скачивающий реплеи Dota 2, часть 1

```
Func ClickPlease($x, $y, $delay = 500)
    MouseMove($x, $y)
    MouseClick("main")
    Sleep($delay)
EndFunc

Func GetNewMatches()
    AutoItSetOption("SendKeyDelay", 100)

    ClickPlease(1886, 92)
    ClickPlease(1732, 99)

    ClickPlease(306, 512, 1000) ; get focus on dotabuff page
    Send("{F5}") ; refresh page
    Sleep(8000)

    Send("{CTRLDOWN}ac{CTRLUP}") ; copy mathes history
    MouseClick("main") ; lose focus

    ClickPlease(1890, 184) ; go back to far


    Send("{SHIFTDOWN}{F4}{SHIFTUP}input{ENTER}")
    Sleep(100)
    Send("{CTRLDOWN}av{CTRLUP}{F2}{ENTER}{ESC}")
    Sleep(100)

    Send("python main.py{ENTER}")
    Sleep(100)
EndFunc


Func DownloadMatches()
    For $i = 1 To 30
        Send("{SHIFTDOWN}{F4}{SHIFTUP}list{ENTER}") ; open list
        Send("{SHIFTDOWN}{END}{SHIFTUP}") ; select current id
        Send("{CTRLDOWN}c{CTRLUP}") ; copy current id
```

```
            Send("{BS}{DEL}{F2}{ESC}")

            ClickPlease(1890, 229, 1000) ; dota

            ClickPlease(819, 115, 1000) ; watch
            ClickPlease(938, 162, 1000) ; replays
            ClickPlease(1449, 160, 1000) ; search

            Send("{CTRLDOWN}v{CTRLUP}{ENTER}")

            Sleep(2500)
            ClickPlease(794, 807)
            Sleep(15000)

            ClickPlease(1890, 184) ; go back to far
        Next
EndFunc


Func FullCycle()
    GetNewMatches()
    DownloadMatches()
EndFunc


Func quit()
    Exit
EndFunc
HotKeySet("q", quit)

While True
    FullCycle()
WEnd
```

Листинг А.2 – Исходный код скрипта, скачивающий реплеи Dota 2, часть 2

```
s = open('input').readlines()

res = []

prefix = "243"
for i in xrange(len(s)):
```

```
    if s[i][:len(prefix)] == prefix:
        matchId = s[i]
        mode = s[i + 2]
        if mode[:8] == 'All Pick':
            res.append(matchId)
        print matchId, mode

res.reverse()

f = open('list', 'a')
for x in res:
    f.write(x)
f.close()


f = open('full', 'a')
for x in res:
    f.write(x)
f.close()
```

Листинг А.3 – Исходный код написанного парсера реплеев Dota 2, часть 1

```
package main

import (
    "log"

    "github.com/dotabuff/manta"
    "github.com/dotabuff/manta/dota"

    "os"
    "math"
    "reflect"
    "fmt"
    "github.com/manucorporat/try"
    "sort"
)

func min(x, y int) int {
    if x <= y { return x }
    return y
}
```

```
func max(x, y int) int {
    if x >= y { return x }
    return y
}
func btoi(x bool) int {
    if x { return 1 }
    return 0
}

type Hero struct {
    x float64
    y float64
    name string
    playerId int
    health int
    mana int
    level int
    team int // 2 -- radiant, 3 -- dire
    xp int
    networth int
    damage int
    maxHealth int
    maxMana int
}

var heroes map[int]Hero

func updateHero(e *manta.PacketEntity) {
    prop := e.Properties.KV
    id := int(e.ClassBaseline.KV["m_iPlayerID"].(int32))

    h := heroes[id]

    h.name = e.ClassName
    h.playerId = id

    h.team = int(e.ClassBaseline.KV["m_iTeamNum"].(uint64))

    cellX := prop["CBodyComponentBaseAnimatingOverlay.m_cellX"]
    vecX := prop["CBodyComponentBaseAnimatingOverlay.m_vecX"]
```

```
if cellX != nil && vecX != nil {
    h.x = float64(cellX.(uint64)) * 128 - 16384 + float64(vecX.(
        float32))
}

cellY := prop["CBodyComponentBaseAnimatingOverlay.m_cellY"]
vecY := prop["CBodyComponentBaseAnimatingOverlay.m_vecY"]

if cellY != nil && vecY != nil {
    h.y = float64(cellY.(uint64)) * 128 - 16384 + float64(vecY.(
        float32))
}

health := prop["m_iHealth"]
if health != nil { h.health = int(health.(int32)) }

mana := prop["m_flMana"]
if mana != nil { h.mana = int(mana.(float32)) }

level := prop["m_iCurrentLevel"]
if level != nil { h.level = int(level.(int32)) }

xp := prop["m_iCurrentXP"]
if xp != nil { h.xp = int(xp.(int32)) }

damageMax := prop["m_iDamageMax"]
damageMin := prop["m_iDamageMin"]
damageBonus := prop["m_iDamageBonus"]

if damageMax != nil && damageMin != nil && damageBonus != nil {
    h.damage = int((damageMin.(int32) + damageMax.(int32)) / 2 +
        damageBonus.(int32))
}

maxHealth := prop["m_iMaxHealth"]
if maxHealth != nil { h.maxHealth = int(maxHealth.(int32)) }

maxMana := prop["m_flMaxMana"]
if maxMana != nil { h.maxMana = int(maxMana.(float32)) }

heroes[id] = h
```

```
}

func (h Hero) out() string {
    return fmt.Sprintf("x=%.3f y=%.3f name=%s playerId=%d health=%d
       mana=%d level=%d team=%d xp=%d networth=%d damage=%d maxHealth
       =%d maxMana=%d",
         h.x, h.y, h.name, h.playerId, h.health, h.mana, h.level, h.
            team, h.xp, h.networth, h.damage, h.maxHealth, h.maxMana)
}

type CombatLogEntry struct {
    tick int
    dota.CMsgDOTACombatLogEntry
}

var tick int

var logToPlayer map[int]int
var playerToLog map[int]int

var gameStartTick int

var memCombatLog []CombatLogEntry
var memHeroes map[int]map[int]Hero

var BAD_COMBATLOG_TYPES map[dota.DOTA_COMBATLOG_TYPES]bool


var PREGAME_TIME = 75

func getTime(tick int) string {
    sec := float64(tick - gameStartTick) / 30 - float64(PREGAME_TIME)
    min := math.Floor(sec / 60)
    sec -= min * 60

    if min < 0 {
        min = math.Abs(min + 1)
        sec = 60 - sec
        return fmt.Sprintf("[time = -%02.0f:%05.02f]", min, sec)
    } else {
        return fmt.Sprintf("[time = %02.0f:%05.02f]", min, sec)
```

```
    }
}


func printTime() {
    log.Println(getTime(tick))
}

type Fight struct {
    startTick int
    endTick int
    heroes map[int]bool
    winner int // 0 -- radiant, 1 -- dire
    RDpoints int
}

func getHero(heroId, tick int) Hero {
    for ;; tick-- {
        if memHeroes[tick] != nil {
            return memHeroes[tick][heroId]
        }
    }
}

func (f *Fight) calculateWinner() {
    radiantPoints := 0
    direPoints := 0
    for hid := range f.heroes {
        h1 := getHero(hid, f.startTick - 30)
        h2 := getHero(hid, f.endTick + 30)

        heroPoints := 0
        heroPoints += h2.xp - h1.xp
        heroPoints += (h2.networth - h1.networth) * 3 / 2
        heroPoints += (h2.health - h1.health) / 4
        heroPoints += (h2.mana - h1.mana) / 5

        if hid <= 4 {
            radiantPoints += heroPoints
        } else {
            direPoints += heroPoints
```

```
        }
    }

    f.RDpoints = radiantPoints - direPoints
    if radiantPoints > direPoints {
        f.winner = 0
    } else {
        f.winner = 1
    }
}

var fightId []int
var fights []Fight

func analyze() {
    if len(logToPlayer) != 10 {
        panic("unusual length logToPlayer")
    }

    fightId = make([]int, len(memCombatLog))
    fights = append(fights, Fight{})

    FI := 0
    for i, e := range memCombatLog {
        if fightId[i] != 0 { continue }
        if *e.Type == dota.DOTA_COMBATLOG_TYPES_DOTA_COMBATLOG_DEATH
           && e.IsTargetHero != nil && *e.IsTargetHero {
            FI += 1

            pid := logToPlayer[int(*e.TargetName)]
                        f := Fight{startTick : e.tick, endTick : e.
                           tick, heroes : make(map[int]bool)}
            f.heroes[pid] = true

            for {
                changed := false

                TIME_GAP := 5  // maximum gap in seconds between
                   combat log entries in one fight
                //walk backward
                for j := i; j >= 0; j-- {
```

```
        ee := memCombatLog[j]
        if BAD_COMBATLOG_TYPES[*ee.Type] { continue }
        if fightId[j] != 0 { continue }
        passedTime := (f.startTick - ee.tick) / 30
        if passedTime > TIME_GAP { break }

        if ee.IsTargetHero != nil && *ee.IsTargetHero {
            target := logToPlayer[int(*ee.TargetName)]
            if !f.heroes[target] { continue }
            fightId[j] = FI
            for _, hid := range ee.AssistPlayers {
                f.startTick = min(f.startTick, ee.tick)
                newPid := int(hid)
                if !f.heroes[newPid] {
                    f.heroes[newPid] = true
                    changed = true
                }
            }
        }
    }

    //walk forward
    for j := i; j < len(memCombatLog); j++ {
        ee := memCombatLog[j]
        if BAD_COMBATLOG_TYPES[*ee.Type] { continue }
        if fightId[j] != 0 { continue }
        passedTime := (ee.tick - f.endTick) / 30
        if passedTime > TIME_GAP { break }

        if ee.IsTargetHero != nil && *ee.IsTargetHero {
            target := logToPlayer[int(*ee.TargetName)]
            if !f.heroes[target] { continue }
            fightId[j] = FI
            for _, hid := range ee.AssistPlayers {
                f.endTick = max(f.endTick, ee.tick)
                newPid := int(hid)
                if !f.heroes[newPid] {
                    f.heroes[newPid] = true
                    changed = true
                }
            }
```

```
                    }
                }



                if !changed { break }
            }
            f.calculateWinner()
            fights = append(fights, f)
        }

    }

}

func main() {
    heroes = make(map[int]Hero)
    logToPlayer = make(map[int]int)
    playerToLog = make(map[int]int)
    gameStartTick = 0
    memHeroes = make(map[int]map[int]Hero)

    replayName := "2384541777.dem"
    if len(os.Args) > 1 {
        replayName = os.Args[1]
    }
    p, err := manta.NewParserFromFile(replayName)
    if err != nil {
        log.Fatalf("unable to create parser: %s", err)
    }

    BAD_COMBATLOG_TYPES = make(map[dota.DOTA_COMBATLOG_TYPES]bool)

    BAD_COMBATLOG_TYPES[dota.DOTA_COMBATLOG_TYPES_DOTA_COMBATLOG_HEAL
        ] = true



    O_winner := -1.0
    O_heroes := []string{}
```

```
p.Callbacks.OnCDemoFileInfo(func(m *dota.CDemoFileInfo) error {
    O_winner = float64(*m.GameInfo.Dota.GameWinner - 2)
    players := m.GameInfo.Dota.PlayerInfo
    cnt := 0
    for _, h := range players {
        if cnt <= 4 && *h.GameTeam != 2 {
            panic("incorrect output")
        }
        if cnt >= 5 && *h.GameTeam != 3 {
            panic("incorrect output")
        }
        cnt += 1

        O_heroes = append(O_heroes, (*h.HeroName)[14:])
    }
    return nil
})


p.Callbacks.OnCMsgDOTACombatLogEntry(func(m *dota.
   CMsgDOTACombatLogEntry) error {
    memCombatLog = append(memCombatLog, CombatLogEntry{tick, *m})
    if gameStartTick == 0 && *m.Type == dota.
       DOTA_COMBATLOG_TYPES_DOTA_COMBATLOG_GAME_STATE {
        if p.Tick > 100 && *m.Value == 4 {
            gameStartTick = int(p.Tick)
        }
    }
    if *m.Type == dota.
       DOTA_COMBATLOG_TYPES_DOTA_COMBATLOG_GAME_STATE && *m.Value
        == 5 {
        gameStartTick = int(p.Tick) - PREGAME_TIME * 30
    }

    if m.AttackerName != nil && m.IsAttackerHero != nil && *m.
       IsAttackerHero && !BAD_COMBATLOG_TYPES[*m.Type] {
        arr := m.AssistPlayers
        attackerName := int(*m.AttackerName)

        ids := []int{}
        for _, id := range arr {
```

```
                if _, ok := playerToLog[int(id)]; !ok {
                    ids = append(ids, int(id))
                }
            }
            if len(ids) == 1 {
                playerToLog[ids[0]] = attackerName
                logToPlayer[attackerName] = ids[0]

            }

        }
        return nil
    })


    p.Callbacks.OnCNETMsg_Tick(func(m *dota.CNETMsg_Tick) error {
        tick = int(p.Tick)
        for _, e := range p.PacketEntities {
            if 457 <= e.ClassId && e.ClassId <= 571 && e.
               ClassBaseline.KV["m_iPlayerID"] != nil {// && e.
               Properties.KV["m_iPlayerID"] != nil {
                updateHero(e)
            }
            if e.ClassName == "CDOTA_DataSpectator" {
                for i := 0; i < 10; i++ {
                    s := fmt.Sprintf("m_iNetWorth.000%d", i)

                    h := heroes[i]
                    networth := e.Properties.KV[s]
                    if networth != nil { h.networth = int(networth.(
                       int32)) }
                    heroes[i] = h
                }
            }

        }
        memHeroes[tick] = make(map[int]Hero)
        for key, val := range heroes {
            memHeroes[tick][key] = val
        }
        return nil
```

```
})

log.Printf("Start! [parsing %s]\n", replayName)


p.Start()

log.Printf("Parse complete. Lets analyze\n")

analyze()

output1, err := os.OpenFile("fights.data", os.O_APPEND | os.
   O_CREATE | os.O_WRONLY, 0666)
if err != nil {
    log.Println(err)
    panic("failed to open a file")
}


C_MINUTES := 20

diff := 0.0
for _, f := range fights {
    if f.startTick >= gameStartTick + C_MINUTES * 60 * 30 { break
        }
    diff += float64(f.RDpoints)
}

log.Println(diff)

output1.WriteString(fmt.Sprintf("%.10f %.10f\n", O_winner, diff))
for i := 0; i < 10; i++ {
    output1.WriteString(O_heroes[i] + "\n")
}


output1.Close()


log.Printf("Finish!\n")
```

```
    //unused imports:
    _ = try.This
    _ = fmt.Println
    _ = sort.Ints
    _ = math.Sin
    _ = reflect.TypeOf
    _ = os.Exit
}
```

Листинг А.4 – Исходный код написанного парсера реплеев Dota 2, часть 2

```
import glob, os
ss = glob.glob("E:\\dotaML\\replays1\\*.dem")
ss += glob.glob("E:\\dotaML\\replays2\\*.dem")

from time import time

average = 20.
cnt = 1
S = 0.

for i in range(len(ss)):
    start = time()
    timeToEnd = average * (len(ss) - i)
    print "%d from %d, time to finish [%.2f seconds] [%.2f minutes]"
      % (i + 1, len(ss), timeToEnd, timeToEnd / 60)
    os.system("1.exe " + ss[i] + " 2>> err")
    end = time()
    dt = end - start
    if i == 0:
        S = 0
        cnt = 0
    S += dt
    cnt += 1
    average = S / cnt

print "the end!"
```

Листинг А.5 – Исходный код программы, запускающей алгоритм логической регрессии на полученных данных, на языке Go

```
package main
```

```
import (
    "github.com/cdipaolo/goml/linear"
    "github.com/cdipaolo/goml/base"
    "fmt"
    "bufio"
    "os"
    "io"
)

var input *bufio.Reader
var END bool

type Fight struct {
    winner float64
    points float64
    heroes []string
}

var lastHeroId int
var ID map[string]int

func getId(s string) int {
    if _, ok := ID[s]; !ok {
        ID[s] = lastHeroId
        lastHeroId += 1
    }
    return ID[s]
}

func readFight() Fight {
    f := Fight{}
    line, _, err := input.ReadLine()
    if err == io.EOF {
        END = true
        return Fight{}
    }
    _, err = fmt.Sscanf(string(line), "%f %f", &f.winner, &f.points)
    if err != nil {
        panic("=((")
    }
```

```
    for i := 0; i < 10; i++ {
        line, _, err := input.ReadLine()
        if err == io.EOF {
            END = true
            return Fight{}
        }
        f.heroes = append(f.heroes, string(line))
    }
    return f
}


func getData(fights []Fight, flag int) (dataX [][]float64, dataY []
   float64) {
    NUMBER_OF_HEROES := 111
    dataX = [][]float64{}
    dataY = []float64{}

    for _, f := range fights {
        row := []float64{}
        for i := 0; i < NUMBER_OF_HEROES; i++ { row = append(row, 0)
           }
        for i, s := range f.heroes {
            id := getId(s)
            if i <= 4 {
                row[id] = 1
            } else {
                row[id] = -1
            }
        }
        dataX = append(dataX, row)
        dataY = append(dataY, f.winner)

        THRESHOLD := 10000.0
        twice := false
        if f.points < -THRESHOLD && f.winner == 1 { twice = true }
        if f.points > THRESHOLD && f.winner == 0 { twice = true }

        if twice && flag == 1 {
            dataX = append(dataX, row)
```

```
            dataY = append(dataY, f.winner)
        }


    }
    return
}


func getSuccess(trainingFights, testFights []Fight) float64 {
    trainingDataX, trainingDataY := getData(trainingFights, 1)
    testDataX, testDataY := getData(testFights, 0)

    bb := linear.NewLogistic(base.BatchGA, 1, 0, 800, trainingDataX,
       trainingDataY)
    err := bb.Learn()
    if err != nil {
        panic("failed to learn")
    }

    CNT := [2][2]int{}

    for i, row := range trainingDataX {
        val1 := int(trainingDataY[i] + 0.5)
        xx, err := bb.Predict(row)
        if err != nil {
            panic("failed to predict")
        }
        val2 := int(xx[0] + 0.5)
        CNT[val1][val2] += 1

    }
    fmt.Println("on training data:", CNT)


    CNT = [2][2]int{}

    for i, row := range testDataX {
        val1 := int(testDataY[i] + 0.5)
        xx, err := bb.Predict(row)
        if err != nil {
```

```
            panic("failed to predict")
        }
        val2 := int(xx[0] + 0.5)
        CNT[val1][val2] += 1

    }
    fmt.Println(CNT)
    success := float64(CNT[0][0] + CNT[1][1]) / float64(CNT[0][0] +
       CNT[0][1] + CNT[1][0] + CNT[1][1])
    fmt.Println("success", success)
    return success
}




func main() {
    lastHeroId = 0
    ID = make(map[string]int)

    file, _ := os.Open("fights.data")
    input = bufio.NewReader(file)
    END = false

    fights := []Fight{}
    for {
        f := readFight()
        if END {
            break
        }
        fights = append(fights, f)
    }


    L := len(fights)
    parts := 5

    success := 0.0

    for i := 0; i < parts; i++ {
        trainingFights := []Fight{}
```

```
        testFights := []Fight{}

        for j := 0; j < L; j++ {
            if j % parts == i {
                testFights = append(testFights, fights[j])
            } else {
                trainingFights = append(trainingFights, fights[j])
            }
        }

        value := getSuccess(trainingFights, testFights)
        success += value
    }

    success /= float64(parts)

    fmt.Println("total success:", success)
}
```