CodeWizards 2016

Правила

Версия 1.0.0

Ноябрь — декабрь, 2016

Оглавление

1 Объявление о проведении Конкурса **2**
- 1.1 Наименование Конкурса . . . 2
- 1.2 Информация об организаторе конкурса . . . 2
- 1.3 Сроки проведения Конкурса . . . 3
- 1.4 Условие получения статуса Участника конкурса . . . 3
- 1.5 Срок регистрации Участников конкурса в Системе Организатора . . . 3
- 1.6 Территория проведения Конкурса . . . 3
- 1.7 Условия проведения Конкурса (существо заданий, критерии и порядок оценки) . . . 3
- 1.8 Порядок определения Победителей и вручения Призов. Призовой фонд Конкурса . . . 4
- 1.9 Порядок и способ информирования участников Конкурса . . . 5

2 О мире CodeWizards 2016 **6**
- 2.1 Общие положения игры и правила проведения турнира . . . 6
- 2.2 Описание игрового мира . . . 8
- 2.3 Классы юнитов . . . 10
- 2.4 Характеристики волшебников . . . 12
- 2.5 Управление волшебником . . . 13
- 2.6 Другие игровые объекты . . . 15
- 2.7 Столкновения юнитов . . . 15
- 2.8 Начисление баллов . . . 15

3 Создание стратегии **17**
- 3.1 Техническая часть . . . 17
- 3.2 Управление волшебником . . . 18
- 3.3 Примеры реализации . . . 20
  - 3.3.1 Пример для Java . . . 20
  - 3.3.2 Пример для C# . . . 20
  - 3.3.3 Пример для C++ . . . 21
  - 3.3.4 Пример для Python 2 . . . 22
  - 3.3.5 Пример для Python 3 . . . 22
  - 3.3.6 Пример для Pascal . . . 23
  - 3.3.7 Пример для Ruby . . . 24

4 Package model **25**
- 4.1 Classes . . . 27
  - 4.1.1 Class **ActionType** . . . 27
  - 4.1.2 Class **Bonus** . . . 28
  - 4.1.3 Class **BonusType** . . . 29
  - 4.1.4 Class **Building** . . . 29
  - 4.1.5 Class **BuildingType** . . . 30
  - 4.1.6 Class **CircularUnit** . . . 31
  - 4.1.7 Class **Faction** . . . 31
  - 4.1.8 Class **Game** . . . 32
  - 4.1.9 Class **LaneType** . . . 44
  - 4.1.10 Class **LivingUnit** . . . 45

4.1.11 CLASS **Message** 46
4.1.12 CLASS **Minion** 47
4.1.13 CLASS **MinionType** 47
4.1.14 CLASS **Move** 48
4.1.15 CLASS **Player** 52
4.1.16 CLASS **Projectile** 53
4.1.17 CLASS **ProjectileType** 54
4.1.18 CLASS **SkillType** 54
4.1.19 CLASS **Status** 57
4.1.20 CLASS **StatusType** 58
4.1.21 CLASS **Tree** 59
4.1.22 CLASS **Unit** 59
4.1.23 CLASS **Wizard** 61
4.1.24 CLASS **World** 63

5 Package < none > 65
5.1 Interfaces 66
5.1.1 INTERFACE **Strategy** 66

Глава 1

Объявление о проведении Конкурса

Общество с ограниченной ответственностью «Мэйл.Ру», созданное и действующее в соответствии с законодательством Российской Федерации, с местом нахождения по адресу: 125167, г. Москва, Ленинградский проспект, д. 39, строение 79, далее по тексту «Организатор конкурса», приглашает физических лиц, достигших к моменту опубликования настоящего Объявления о конкурсе 18 лет, далее по тексту «Участник конкурса», к участию в конкурсе на нижеследующих условиях:

1.1 Наименование Конкурса

«Российский кубок по программированию искусственного интеллекта (Russian AI Cup)».

Целями проведения Конкурса являются:

- повышение общественного интереса к сфере создания программных продуктов;
- предоставление Участникам конкурса возможности раскрыть творческие способности;
- развитие профессиональных навыков Участников конкурса.

Конкурс состоит из 3 (трёх) этапов, каждый из которых завершается определением Победителей. Последний этап Конкурса является решающим.

1.2 Информация об организаторе конкурса

Наименование: ООО «Мэйл.Ру»

Адрес места нахождения: 125167, г. Москва, Ленинградский проспект, д. 39, строение 79

Почтовый адрес: 125167, г. Москва, Ленинградский проспект, д. 39, строение 79, БЦ «SkyLight»

Телефон: (495) 725-63-57

Сайт: http://www.russianaicup.ru

E-мейл: russianaicup@corp.mail.ru

1.3 Сроки проведения Конкурса

Срок проведения Конкурса: с 00.00 часов 7 ноября 2016 года до 24.00 часов 25 декабря 2016 года по Московскому времени.

Первая неделя (с 00.00 часов 7 ноября 2016 года до 24.00 часов 13 ноября 2016 года) и четвёртая неделя (с 00.00 часов 28 ноября 2016 года до 24.00 часов 4 декабря 2016 года) Конкурса являются тестовыми. В течение этого периода функциональность сайта и тестирующей системы Конкурса может быть неполной, а в правила могут вноситься существенные изменения.

Сроки начала и окончания этапов Конкурса:

- первый этап — с 00 часов 00 минут 26 ноября 2016 года до 24 часов 00 минут 27 ноября 2016 года;
- второй этап — с 00 часов 00 минут 10 декабря 2016 года до 24 часов 00 минут 11 декабря 2016 года;
- третий этап (заключительный) — с 00 часов 00 минут 17 декабря 2016 года до 24 часов 00 минут 18 декабря 2016 года.

1.4 Условие получения статуса Участника конкурса

Для участия в Конкурсе необходимо пройти процедуру регистрации в Системе Организатора конкурса, размещённой на сайте Организатора конкурса в сети Интернет по адресу: http://www.russianaicup.ru.

1.5 Срок регистрации Участников конкурса в Системе Организатора

Регистрация Участников конкурса проводится с 00.00 часов 7 ноября 2016 года до 24.00 часов 25 декабря 2016 года включительно.

1.6 Территория проведения Конкурса

Конкурс проводится на территории Российской Федерации. Проведение всех этапов Конкурса осуществляется путем удалённого доступа к Системе Организатора конкурса через сеть Интернет.

1.7 Условия проведения Конкурса (существо заданий, критерии и порядок оценки)

Порядок проведения Конкурса, существо задания, критерии и порядок оценки указаны в конкурсной документации в главе 2.

Конкурсная документация включает в себя:

- Объявление о проведении Конкурса;
- Соглашение об организации и порядке проведения Конкурса;

- Правила проведения Конкурса;
- информационные данные, содержащиеся в Системе Организатора конкурса.

Участник конкурса может ознакомиться с конкурсной документацией на сайте Организатора конкурса в сети Интернет по адресу: http://www.russianaicup.ru, а также при прохождении процедуры регистрации в Системе Организатора конкурса.

Организатор конкурса оставляет за собой право на изменение конкурсной документации, условий проведения Конкурса и отказ от его проведения в соответствии с условиями конкурсной документации и нормами законодательства РФ. При этом, Организатор Конкурса обязуется уведомить Участников конкурса обо всех произошедших изменениях путём отправки уведомления, в порядке и на условиях, предусмотренных в конкурсной документации.

1.8 Порядок определения Победителей и вручения Призов. Призовой фонд Конкурса

Критерии оценки результатов Конкурса, количество и порядок определения Победителей содержатся в главе 2 данного документа.

Призовой фонд Конкурса формируется за счет средств Организатора конкурса.

Призовой фонд:

- 1 место — Apple Macbook Pro 13";
- 2 место — Apple Macbook Air 13";
- 3 место — Apple iPad;
- 4-6 места — ценные призы;
- 1-6 места в Песочнице — ценные призы.

Все участники Конкурса, принявшие участие во втором или третьем этапах, будут награждены футболкой. Все участники Конкурса, принявшие участие в третьем этапе, также получат толстовку с символикой соревнования.

Все участники, занявшие призовые места, будут оповещены посредством отправки сообщения на адрес электронной почты, указанный участником при регистрации в Системе Организатора.

Призы будут высланы участникам в виде посылок, используя Почту России или другую почтовую службу, в течение двух месяцев после окончания финального этапа. Срок доставки приза по почтовому адресу, указанному участником, зависит от сроков доставки используемой почтовой службы. Почтовые адреса призёров для отправки призов Организатор получает из учётных данных участника в Системе Организатора. Адрес должен быть указан участником-призёром в течение трёх дней после получения уведомления о получении приза.

При отсутствии ответа в обозначенные сроки или отказе предоставить точные данные, необходимые для вручения призов Конкурса, Организатор оставляет за собой право отказать такому участнику в выдаче приза Конкурса. Денежный эквивалент приза не выдаётся.

Победители Конкурса обязуются предоставить Организатору конкурса копии всех документов, необходимых для бухгалтерской и налоговой отчётности Организатора конкурса. Перечень документов, которые Победитель обязан предоставить Организатору конкурса, может включать в себя:

- копию паспорта Победителя;
- копию свидетельства о постановке на налоговый учет Победителя;
- копию пенсионного удостоверения Победителя;
- данные об открытии банковского лицевого счета Победителя;
- иные документы, которые Организатор конкурса потребует от Участника конкурса в целях формирования отчётности о проведённом Конкурсе.

Наряду с копиями Организатор конкурса вправе запросить оригиналы вышеуказанных документов.

В соответствии с подпунктом 4 пункта 1 статьи 228 НК РФ Победитель Конкурса, ставший обладателем Приза, самостоятельно несёт все расходы по уплате всех применимых налогов, установленных действующим законодательством Российской Федерации.

1.9 Порядок и способ информирования участников Конкурса

Информирование Участников Конкурса осуществляется путём размещения информации в сети Интернет на Сайте Организатора конкурса по адресу: http://www.russianaicup.ru, а также через Систему Организатора конкурса, в течение всего срока проведения Конкурса.

Глава 2

О мире CodeWizards 2016

2.1 Общие положения игры и правила проведения турнира

Данное соревнование предоставляет вам возможность проверить свои навыки программирования, создав искусственный интеллект (стратегию), управляющий волшебником в специальном игровом мире (подробнее об особенностях мира CodeWizards 2016 можно узнать в следующих разделах). Правила соревнования базируются на популярном в мире компьютерных игр жанре MOBA. В каждой игре вам будет противостоять пять стратегий других игроков. В то же время, у вас будет четыре союзника. Пять стратегий, находящиеся на одной стороне, составляют фракцию: Академию или Отступников. Основной командной целью этих пяти игроков является уничтожение базы противоположной фракции. Основной персональной целью каждого волшебника является сбор максимально возможного количества баллов. Звание победителя игры, а также все остальные места распределяются в соответствии с количеством набранных баллов. Два или более игроков могут делить одно место, если их баллы равны. Игроку начисляются баллы, если его волшебник наносит урон, уничтожает или просто находится рядом во время смерти юнита другой фракции, а также за некоторые другие действия. Всем игрокам фракции начисляется значительное количество баллов в случае достижения основной командной цели.

Правила игры практически полностью соответствуют классическим канонам жанра. Фракционные базы соединены тремя дорожками (верхней, центральной и нижней), в промежутках между которыми находятся лесные массивы. На самих дорожках находятся охранные башни: по 2 на дорожку от каждой фракции. Таким образом, в начале игры на карте присутствует 14 строений. С определённым периодом база каждой фракции генерирует 3 одинаковых отряда приспешников («миньонов») волшебников: по одному на каждую дорожку. Они сразу же устремляются по своей дорожке в направлении базы противоположной фракции, атакуя всех противников на пути.

Турнир проводится в несколько этапов, которым предшествует квалификация в Песочнице. Песочница — соревнование, которое проходит на протяжении всего чемпионата. В рамках каждого этапа игроку соответствует некоторое значение рейтинга — показателя того, насколько успешно его стратегия участвует в играх.

Начальное значение рейтинга в Песочнице равно 1200. По итогам игры это значение может как увеличиться, так и уменьшиться. При этом победа над слабым (с низким рейтингом) противником даёт небольшой прирост, также и поражение от сильного соперника незначительно уменьшает ваш рейтинг. Со временем рейтинг в Песочнице становится всё более инертным, что позволяет уменьшить влияние случайных длинных серий побед или поражений на место участника, однако вместе с тем и затрудняет изменение его положения при существенном улучшении стратегии. Для отмены данного эффекта участник может сбросить изменчивость рейтинга до начального состояния при отправке новой стратегии, включив соответствующую опцию. В случае принятия новой стратегии системой рейтинг участника сильно упадёт после следующей игры в Песочнице, однако по мере дальнейшего участия в играх быстро восстановится и даже станет выше, если ваша стратегия действительно стала эффективнее. Не рекомендуется использовать данную опцию при

незначительных, инкрементальных улучшениях вашей стратегии, а также в случаях, когда новая стратегия недостаточно протестирована и эффект от изменений в ней достоверно не известен.

Начальное значение рейтинга на каждом основном этапе турнира равно 0. За каждую игру участник получает определённое количество единиц рейтинга в зависимости от занятого места (система, аналогичная используемой в чемпионате «Формула-1»). Если два или более участников делят какое-то место, то суммарное количество единиц рейтинга за это место и за следующие `количество_таких_участников` − 1 мест делится поровну между этими участниками. Например, если два участника делят третье место, то каждый из них получит половину от суммы единиц рейтинга за третье и четвёртое места. При делении округление всегда совершается в меньшую сторону. Более подробная информация об этапах турнира будет предоставлена в анонсах на сайте проекта.

Сначала все участники могут участвовать только в играх, проходящих в Песочнице. Игроки могут отправлять в Песочницу свои стратегии, и последняя принятая из них берётся системой для участия в квалификационных играх. Каждый игрок участвует примерно в одной квалификационной игре за час. Жюри оставляет за собой право изменить этот интервал, исходя из пропускной способности тестирующей системы, однако для большинства участников он остаётся постоянной величиной. Существует ряд критериев, по которым интервал участия в квалификационных играх может быть увеличен для конкретного игрока. Если с момента отправки игроком последней стратегии прошло более недели, то интервал участия для этого игрока увеличивается вдвое. Учитываются только принятые системой стратегии. За каждое «падение» стратегии в 10 последних играх в Песочнице начисляется дополнительный штраф, равный 20% от базового интервала тестирования. Подробнее о причинах «падения» стратегии можно узнать в следующих разделах.

Игры в Песочнице проходят по набору правил, соответствующему правилам случайного прошедшего этапа турнира или же правилам следующего (текущего) этапа. При этом, чем ближе значение рейтинга двух игроков в рамках Песочницы, тем больше вероятность того, что они окажутся в одной игре. Песочница стартует до начала первого этапа турнира и завершается через некоторое время после финального (смотрите расписание этапов для уточнения подробностей). Помимо этого Песочница замораживается на время проведения этапов турнира. По итогам игр в Песочнице происходит отбор для участия в Раунде 1, в который пройдут 1080 участников с наибольшим рейтингом (при его равенстве приоритет отдаётся игроку, раньше отправившему последнюю версию своей стратегии), а также добор в следующие этапы турнира, включая Финал.

Этапы турнира:

- В **Раунде** 1 вам предстоит изучить правила игры и освоить базовое управление волшебником. В каждой игре данного этапа примет участие 10 игроков, которые будут распределены по двум фракциям таким образом, что разница между суммами текущих рейтингов[1] участников двух фракций будет минимальной. Волшебник с наибольшим рейтингом в каждой фракции назначается верховным. Его стратегия может отправлять сообщения другим волшебникам из той же фракции, а также ей предоставляется больше вычислительных ресурсов (процессорного времени). Функциональность сообщений ограничена и позволяет только направлять[2] других волшебников на определённую линию. В данном режиме волшебникам доступны только удар посохом и заклинание «Магическая ракета», а количество жизненной энергии всех строений составляет половину от нормального. Коэффициент повреждения при случайном попадании по дружественному волшебнику равен 25%. Независимо от этапа чемпионата, волшебник не может нанести урон дружественным миньонам и строениям. Раунд 1, как и все последующие этапы, состоит из двух частей, между которыми будет небольшой перерыв (с возобновлением работы Песочницы), который позволит улучшить свою стратегию. Для игр в каждой части выбирается последняя стратегия, отправленная игроком до начала этой части. Игры проводятся волнами. В каждой волне каждый игрок участвует ровно в одной игре. Количество волн в каждой части определяется возможностями тестирующей системы, но гарантируется, что оно не будет меньше десяти. 300 участников с наиболее высоким рейтингом пройдут в Раунд 2. Также в Раунд 2 будет

[1] При проведении игры в рамках Раунда 1 учитывается рейтинг в нём. При проведении игры в Песочнице по правилам Раунда 1 учитывается рейтинг в Песочнице.

[2] Независимо от этапа турнира, сообщения верховного волшебника могут быть как частично, так и полностью проигнорированы стратегиями других волшебников.

проведён добор 60 участников с наибольшим рейтингом в Песочнице (на момент начала Раунда 2) из числа тех, кто не прошёл по итогам Раунда 1.

- В **Раунде** 2 вам предстоит улучшить свои навыки управления волшебником, а также изучить механику получения волшебником новых уровней и изучения им умений. Правильный подход к выбору и использованию умений является ключом к победе в данном этапе. Компоновка игр осуществляется аналогично Раунду 1, однако возможности верховного волшебника будут расширены. Он сможет указывать другим волшебникам фракции, какие умения им лучше всего изучить. Строения в данном этапе имеют нормальное количество жизненной энергии, а коэффициент повреждения при случайном попадании по дружественному волшебнику равен 50%. Дополнительно усложняет задачу то, что после подведения итогов Раунда 1 часть слабых стратегий будет отсеяна и вам придётся противостоять более сильным соперникам. По итогам Раунда 2 лучшие 50 стратегий попадут в Финал. Также в Финал будет проведен добор 10 участников с наибольшим рейтингом в Песочнице (на момент начала Финала) из числа тех, кто не прошёл в рамках основного турнира.
- **Финал** является самым серьёзным этапом. После отбора, проведённого по итогам двух первых этапов, останутся сильнейшие. И в каждой игре вам придётся сойтись лицом к лицу с одним из них. Именно так. Для управления пятью волшебниками одной фракции будет запущено 5 экземпляров вашей стратегии. Для управления пятью волшебниками противоположной фракции будет запущено 5 экземпляров стратегии оппонента. В каждой фракции случайный волшебник будет назначен верховным. Он сможет отправлять другим волшебникам фракции сообщения в бинарном формате. Ограничение на размер бинарных данных относительно велико, однако получение такого сообщения произойдёт после задержки, пропорциональной его длине. Коэффициент повреждения при случайном попадании по дружественному волшебнику равен 100%. Для определения победителя баллы, набранные всеми волшебниками каждой из фракций, будут просуммированы. В остальном правила игры не будут отличаться от правил Раунда 2. Система проведения Финала имеет свои особенности. Этап по-прежнему делится на две части, однако они уже не будут состоять из волн. В каждой части этапа будут проведены игры между всеми парами участников Финала. Если позволит время и возможности тестирующей системы, операция будет повторена.

После окончания Финала все финалисты упорядочиваются по невозрастанию рейтинга. При равенстве рейтингов более высокое место занимает тот финалист, чья участвовавшая в Финале стратегия была отослана раньше. Призы за Финал распределяются на основании занятого места после этого упорядочивания. Лучшие шесть финалистов награждаются призами:

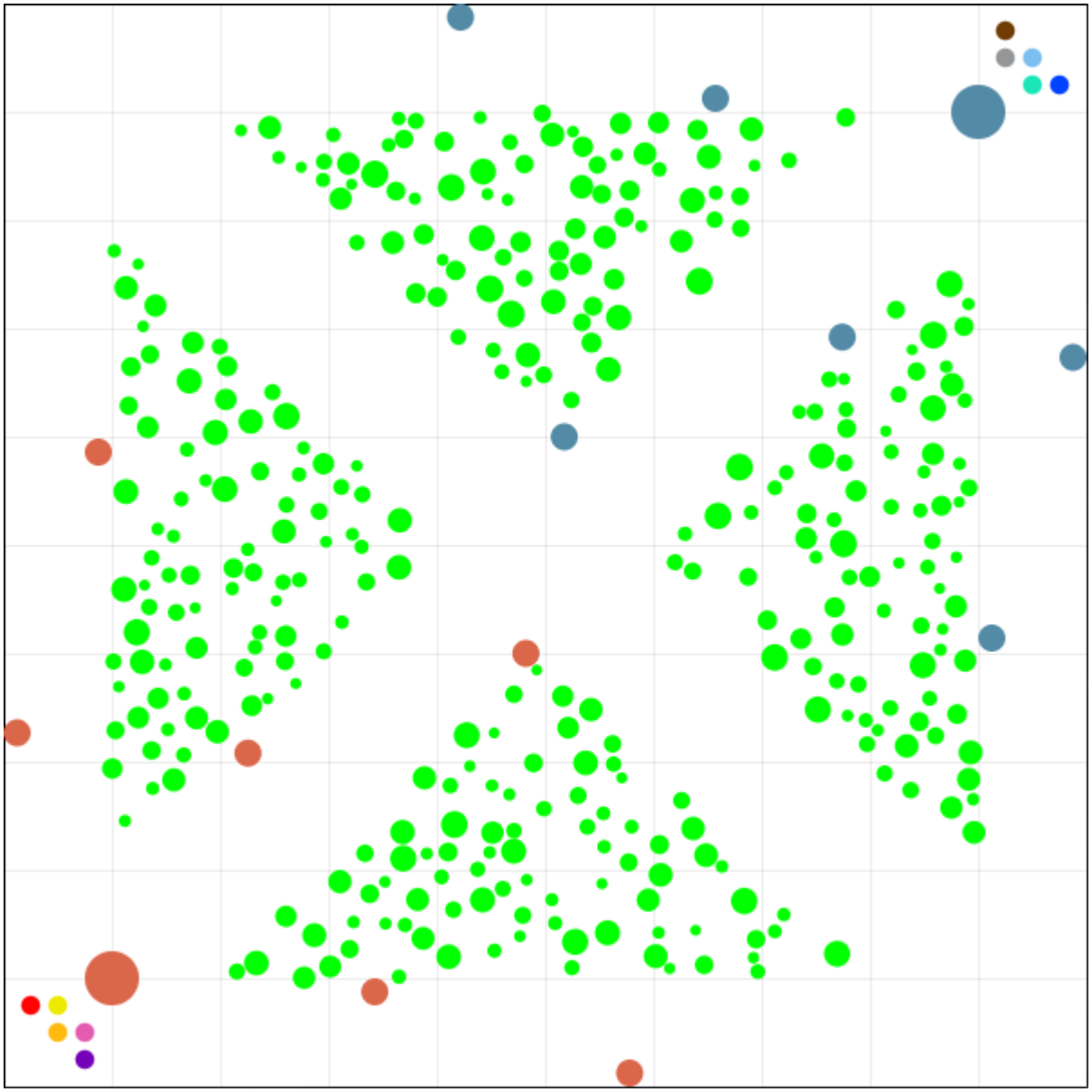
- 1 место — Apple Macbook Pro 13";
- 2 место — Apple Macbook Air 13";
- 3 место — Apple iPad;
- 4-6 места — ценные призы.

После окончания Песочницы все её участники, кроме призёров Финала, упорядочиваются по невозрастанию рейтинга. При равенстве рейтингов более высокое место занимает тот участник, который раньше отослал последнюю версию своей стратегии. Призы за Песочницу распределяются на основании занятого места после этого упорядочивания. Лучшие шесть участников Песочницы награждаются ценными подарками.

2.2 Описание игрового мира

Игровой мир является двумерным, а все юниты в нём имеют форму круга. Ось абсцисс в этом мире направлена слева направо, ось ординат — сверху вниз, угол 0.0 совпадает с направлением оси абсцисс, а положительный угол вращения означает вращение по часовой стрелке. Игровая область ограничена квадратом, левый верхний угол которого имеет координаты (0.0, 0.0), а длина стороны равна 4000.0. Ни один живой юнит не может покинуть пределы игровой области.

Далее приведена схема карты, где медным цветом обозначены строения Академии (кругом большего радиуса — база фракции, кругами меньшего радиуса — охранные башни), стальным синим цветом — строения Отступников, зелёным — деревья. 10 цветных отметок (5 в левом нижнем углу карты для Академии и 5 в правом верхнем углу карты для Отступников) являются начальными позициями волшебников. Стоит отметить, что координаты центров и радиусы деревьев в лесных массивах могут отличаться от игры к игре. Также в процессе игры могут появляться новые деревья.



Время в игре дискретное и измеряется в «тиках». В начале каждого тика игра получает от стратегий желаемые действия волшебников в этот тик и обновляет состояние волшебников в соответствии с этими желаниями и ограничениями мира. Затем происходит расчёт изменения мира и объектов в нём за этот тик, и процесс повторяется снова с обновлёнными данными. Максимальная длительность любой игры равна 20000 тиков, однако игра может быть прекращена досрочно, если достигнута командная цель одной из фракций либо стратегии всех участников «упали». Крайне маловероятно, но всё-таки возможно, что командные цели обеих фракций будут достигнуты в один и тот же тик. Тогда дополнительные баллы получат все участники игры.

«Упавшая» стратегия больше не может управлять волшебником. Стратегия считается «упавшей» в следующих случаях:

- Процесс, в котором запущена стратегия, непредвиденно завершился, либо произошла ошибка в протоколе взаимодействия между стратегией и игровым сервером.
- Стратегия превысила одно (любое) из отведённых ей ограничений по времени. Стратегии на один тик выделяется не более 5 секунд реального времени. Но в сумме на всю игру процессу стратегии обычного волшебника выделяется

$$20 \times <длительность\_игры\_в\_тиках> + 5000 \tag{2.1}$$

миллисекунд реального времени и

$$10 \times <длительность\_игры\_в\_тиках> + 5000 \tag{2.2}$$

миллисекунд процессорного времени, а стратегии верховного волшебника

$$30 \times <длительность\_игры\_в\_тиках> + 5000 \tag{2.3}$$

миллисекунд реального времени и

$$20 \times <длительность\_игры\_в\_тиках> + 5000 \tag{2.4}$$

миллисекунд процессорного времени.[3] В формуле учитывается максимальная длительность игры. Ограничение по времени остаётся прежним, даже если реальная длительность игры отличается от этого значения. Все ограничения по времени распространяются не только на код участника, но и на взаимодействие клиента-оболочки стратегии с игровым симулятором.

- Стратегия превысила ограничение по памяти. В любой момент времени процесс стратегии не должен потреблять более 256 Мб оперативной памяти.

Обнаружение юнитов на карте ограничено туманом войны. Стратегия участника будет получать данные только о тех юнитах, которые находятся в пределах дальности[4] зрения самого волшебника либо любого другого юнита из его фракции.

2.3 Классы юнитов

В мире CodeWizards 2016 существует 6 классов юнитов, некоторые из которых, в свою очередь, делятся на типы:

- волшебники;
- снаряды: магическая ракета (`MAGIC_MISSILE`), ледяная стрела (`FROST_BOLT`), огненный шар (`FIREBALL`) и дротик (`DART`);
- бонусы: усиление (`EMPOWER`), ускорение (`HASTE`) и щит (`SHIELD`);
- строения: база фракции (`FACTION_BASE`) и охранная башня (`GUARDIAN_TOWER`);
- миньоны: орк-дровосек (`ORC_WOODCUTTER`) и фетиш с дротиками (`FETISH_BLOWDART`);
- деревья.

---

[3] Несмотря на то, что ограничение реального времени заметно выше ограничения процессорного времени, запрещено искусственно «замедлять» тестирование стратегии командами типа «`sleep`» (равно как и пытаться замедлить/дестабилизировать тестирующую систему другими способами). В случае выявления подобных злоупотреблений, жюри оставляет за собой право применить к данному пользователю меры на своё усмотрение, вплоть до дисквалификации из соревнования и блокировки аккаунта.

[4] Здесь и далее под расстоянием между юнитами подразумевается расстояние между их центрами, если явно не указано другое.

Волшебники, строения, миньоны и деревья являются живыми юнитами. Основными характеристиками каждого живого юнита являются текущее и максимальное количество жизненной энергии. В общем случае, при падении количества жизненной энергии до нуля юнит считается погибшим и убирается из игрового мира. Волшебники — единственные живые юниты, обладающие регенерацией здоровья. Каждый тик они автоматически восстанавливают некоторое количество жизненной энергии. Скорость регенерации является вещественным числом, как правило, меньшим единицы. На протяжении нескольких тиков может казаться, что здоровье волшебника не восстанавливается, однако это не так. Суммарная регенерация за прошедшие тики накапливается в специальном пуле. Волшебник считается погибшим, если целочисленная часть его жизненной энергии падает до нуля, независимо от того, сколько энергии сейчас накоплено в пуле.

Сравнительные характеристики живых юнитов приведены в следующей таблице:

Характеристика живого юнита	Волшебник	Орк	Фетиш	Охранная башня	База фракции	Дерево
Радиус	35	25	25	50	100	20 – 50
Макс. жизн. энергия	100[5]	100	100	1000[6]	2000[6]	6 – 36
Дальность зрения	600	400	400	600	800	—
Дистанция дальнего боя[7]	500	—	300	600	800	—
Дистанция ближнего боя[8]	70	50	—	—	—	—
Интервал между атаками	30[9]	60	30	240	240	—
Урон одной атаки	12[10]	12	6	24	36	—

Радиус бонуса, независимо от типа, равен 20.

Сравнительные характеристики снарядов приведены в следующей таблице:

Характеристика снаряда	Магическая ракета	Ледяная стрела	Огненный шар	Дротик
Радиус	10	15	20	5
Скорость	40	35	30	50
Урон при прямом попадании	12	24	24[11]	6

Помимо нанесения урона, ледяная стрела дополнительно замораживает (`FROZEN`) цель на 60 тиков. Замороженный юнит не может перемещаться и совершать какие-либо действия, однако количество тиков, оставшееся до применения следующего действия, продолжает уменьшаться. Здания и деревья не могут быть заморожены. Стратегия волшебника не получает управление до окончания действия заморозки.

При попадании в цель либо при достижении максимальной дальности полёта огненный шар взрывается, нанося урон всем живым юнитам, расстояние от центра огненного шара до ближайшей точки которых не превышает 100. При этом на всех юнитов, получивших урон от взрыва, дополнительно накладывается статус горения (`BURNING`). Он действует 240 тиков и за это время наносит 24 единицы урона. Урон от нескольких статусов того же типа суммируется.

Атака строения не вызывает создание снаряда. Нанесение урона происходит мгновенно, в результате разряда магической энергии.

---

[5] Максимальная жизненная энергия волшебника нулевого уровня. При получении каждого нового уровня увеличивается на 10.

[6] Реальное количество жизненной энергии строения в конкретном этапе соревнования может отличаться от табличного.

[7] Максимальная дальность полёта снаряда либо максимальная дальность между центрами юнитов.

[8] Максимально возможная дистанция до ближайшей точки цели.

[9] Минимально возможный интервал между двумя последовательными атаками или заклинаниями волшебника. Каждый тип действия дополнительно ограничен собственным интервалом.

[10] Базовый урон атаки ближнего боя, а также простейшего заклинания «Магическая ракета». Может возрасти вследствие изучения волшебником некоторых умений.

[11] Формально, огненный шар не наносит урона при прямом попадании, однако от столкновения он взрывается, нанося урон всем живым юнитам, расстояние от центра огненного шара до ближайшей точки которых не превышает 100, в том числе и живому юниту, с которым произошло столкновение. Если указанное расстояние не превышает радиус огненного шара, то взрыв наносит максимальный урон. По мере возрастания дальности урон равномерно уменьшается и на периферии области взрыва составляет половину от табличного.

2.4 Характеристики волшебников

Основными характеристиками каждого живого юнита являются текущее и максимальное количество жизненной энергии. Однако у волшебника есть ряд других, не менее важных характеристик. Например, воможность волшебника произносить заклинания ограничена текущим и максимальным количеством магической энергии, а также скоростью её регенерации.

В отличие от других живых юнитов, волшебника невозможно уничтожить окончательно, можно лишь разрушить его телесную оболочку. Через некоторое время он возродится в новом теле. Для возрождения должно пройти не менее 1200 тиков с момента смерти волшебника и не менее 2400 тиков с момента последнего его возрождения. Волшебник появляется на своей начальной позиции либо недалеко от неё, если она занята.

В некоторых режимах игры волшебник может накапливать опыт и повышать свой уровень. Начальный уровень каждого волшебника — нулевой, а максимальный — 25. Тем не менее, правила игры сбалансированы так, что достижение 15-го уровня является почти невозможной задачей. Волшебник получает опыт в том же количестве и в тех же случаях, когда игрок, управляющий им, получает баллы, за исключением случаев, когда волшебник в тик получения баллов мёртв и ожидает возрождения. Для получения первого уровня необходимо набрать 50 единиц опыта, второго — дополнительные 100 единиц, третьего — 150, четвёртого — 200 и т.д.

Количество жизненной энергии на нулевом уровне равно количеству магической и составляет 100 единиц. Прирост количества обоих типов энергии за каждый уровень равен 10. Регенерация жизненной энергии на нулевом уровне составляет 0.05 тиков$^{-1}$ и каждый уровень увеличивается на 0.005 тиков$^{-1}$. Соответствующие характеристики магической энергии в 4 раза выше.

За каждый достигнутый уровень волшебник может изучить ровно одно умение. Умения условно разделены на 5 групп. Умения внутри каждой группы имеют строгий порядок, и изучать их можно только последовательно. Количество умений в каждой группе равно 5. Первое и третье из них, как правило, пассивно увеличивают одну из характеристик волшебника. Второе и четвёртое являются аурами. Действие ауры аналогично действию пассивного умения, однако применяется не только к самому волшебнику, но и ко всем дружественным волшебникам, расстояние до которых не превышает 500. Если на какого-либо волшебника одновременно действует несколько аур, повышающих одну и ту же характеристику, учитывается только та из них, которая оказывает наибольший эффект. Последнее, пятое умение в каждой группе является наиболее важным («ultimate»). Оно позволяет волшебнику произносить новое заклинание либо значительно улучшает уже имеющееся.

- Пассивные умения и ауры первой группы увеличивают дальность заклинаний волшебника на 25 за уровень. Таким образом, дальность заклинаний волшебника, изучившего все эти умения, составит 600. Дополнительно все дружественные волшебники поблизости получат прибавку к дальности своих заклинаний, равную 50. Последнее умение группы позволит использовать заклинание «Магическая ракета» без задержки. По умолчанию, задержка применения данного заклинания составляет 60 тиков. Тем не менее, общая задержка на действия волшебника всё ещё применяется.
- Пассивные умения и ауры второй группы увеличивают урон, наносимый магическими снарядами, на 1 за уровень. Последнее умение группы позволит волшебнику использовать заклинание «Ледяная стрела», которое наносит значительный урон одному противнику, а также замораживает его на 60 тиков.
- Пассивные умения и ауры третьей группы увеличивают урон, наносимый при ударе посохом в ближнем бою, на 3 за уровень. Последнее умение группы позволит волшебнику использовать заклинание «Огненный шар», которое наносит урон группе противников, а также поджигает их.
- Пассивные умения и ауры четвёртой группы увеличивают максимально возможное перемещение волшебника за тик на 5% за уровень. Суммарный бонус четырёх умений составит 20%, а аура добавит 10% к возможности перемещения дружественных волшебников поблизости. Последнее умение группы позволит волшебнику использовать заклинание «Ускорение», которое применяет к дружественному

волшебнику статус `HASTENED` на 600 тиков. Данный статус добавляет дополнительные 30% к скорости перемещения, а также увеличивает скорость поворота на 50%. Действие нескольких статусов того же типа не суммируется.

- Пассивные умения и ауры пятой группы уменьшают урон, получаемый от магических снарядов и атак строений, на 1 за уровень. Последнее умение группы позволит волшебнику использовать заклинание «Щит», которое применяет к дружественному волшебнику статус `SHIELDED` на 600 тиков. Данный статус уменьшает любой прямой урон на 25%. Исключение составляет только урон от статуса `BURNING`. Действие нескольких статусов того же типа не суммируется.

Полный список доступных умений вы можете найти в документации к перечислению `SkillType` (глава 4).

2.5 Управление волшебником

В начале каждого тика симулятор игры отправляет стратегии каждого живого волшебника сведения о текущем состоянии видимой этому волшебнику части мира. В ответ стратегия отправляет набор инструкций для управления волшебником, инкапсулированных в объекте класса `Move`.

Эти инструкции обрабатываются в следующем порядке:

- Каждый волшебник изучает умение `move.skillToLearn`, если оно установлено. Игровой симулятор проигнорирует инструкцию, если количество изученных волшебником умений уже равно его уровню либо волшебник ещё не изучил все предшествующие умения в группе.
- Затем волшебники в случайном порядке совершают действия, установленные в `move.action`:
  - `NONE`. *Ничего не делать.*
  - `STAFF`. *Ударить посохом.* Атака поражает все живые объекты в секторе от $-\pi/12$ до $\pi/12$. Расстояние от центра волшебника до ближайшей точки цели не должно быть больше 70. Если волшебник погиб в результате удара посохом другого волшебника, совершённого ранее в этот же тик, то удар посохом всё равно будет произведён. Погибший волшебник не приобретёт опыта за нанесённый урон, однако игрок, чья стратегия управляет этим волшебником, получит соответствующее количество баллов.
  - `MAGIC_MISSILE`, `FROST_BOLT` или `FIREBALL`. *Создать соответствующий магический снаряд.* При создании снаряда его центр совпадает с центром волшебника, а направление определяется как `wizard.angle + move.castAngle`. Значение `move.castAngle` устанавливается стратегией и ограничено интервалом от $-\pi/12$ до $\pi/12$. Дополнительные параметры `move.minCastDistance` и `move.maxCastDistance` определяют минимальную и максимальную дальность полёта снаряда. Если расстояние от центра снаряда до точки его появления меньше, чем `move.minCastDistance`, то снаряд беспрепятственно проходит сквозь все другие игровые объекты, за исключением деревьев. Если расстояние от центра снаряда до точки его появления больше, чем `move.maxCastDistance`, то снаряд убирается из игрового мира. При этом, снаряд типа `FIREBALL` детонирует. Столкновения магического снаряда и создавшего его волшебника игнорируются. Если волшебник погиб в результате удара посохом другого волшебника, совершённого ранее в этот же тик, то снаряд создан не будет.
  - `HASTE` или `SHIELD`. *Применить к цели соответствующий магический статус.* Статус применяется к дружественному волшебнику с идентификатором `move.statusTargetId` или к самому себе, если такой волшебник не найден. Цель должна находиться в секторе от $-\pi/12$ до $\pi/12$, а расстояние до неё не должно превышать `wizard.castRange`. Значение `wizard.castRange` для волшебника нулевого уровня равно 500, однако может возрасти до 600 после изучения некоторых умений. Если волшебник погиб в результате удара посохом другого волшебника, совершённого ранее в этот же тик, статус не будет применён.

Между любыми двумя последовательными действиями волшебника, отличными от `NONE`, должно пройти не менее 30 тиков. Каждому действию также соответствует своя собственная задержка, которая ограничивает последовательное использование двух действий одного типа. Создание снаряда или применение статуса расходует магическую энергию волшебника. Желаемое действие будет проигнорировано, если её недостаточно.

Характеристика действия	Удар посохом	Ускорение	Щит
Задержка	60	120	120
Стоимость	—	48	48

Характеристика действия	Магическая ракета	Ледяная стрела	Огненный шар
Задержка	60[12]	90	120
Стоимость	12	36	48

- Далее волшебники снова упорядочиваются случайным образом, и происходит их перемещение.

  В мире CodeWizards 2016 нет инерции, и изменение скорости происходит мгновенно. Пожелание стратегии `move.speed` определяет перемещение волшебника вперёд/назад, а `move.strafeSpeed` — перемещение боком. Значение `move.speed` ограничено интервалом от -3.0 до 4.0, где положительные значения соответствуют движению вперёд, а отрицательные — назад. Значение `move.strafeSpeed` ограничено интервалом от -3.0 до 3.0, где положительные значения соответствуют движению правым боком, а отрицательные — левым. Границы обоих интервалов могут быть расширены в зависимости от изученных волшебником умений, от действия некоторых аур, а также в случае действия статуса `HASTENED`. Оба типа перемещения могут осуществляться одновременно в один тик, однако в таком случае к ним применяется дополнительное ограничение: если значение $\sqrt{(\frac{move.speed}{maxSpeed})^2 + (\frac{move.strafeSpeed}{maxStrafeSpeed})^2}$ больше 1.0, то обе установки скорости перемещения (`move.speed` и `move.strafeSpeed`) будут поделены игровым симулятором на это значение. `maxSpeed` и `maxStrafeSpeed` — соответствующие ограничения, действующие на перемещение волшебника в данный тик (с учётом направления перемещения, изученных умений и действующих аур и статусов).

  Перемещение волшебников осуществляется последовательно, согласно выбранному порядку. При этом, частичное перемещение волшебника не применяется. Если позицию волшебника невозможно изменить на всю величину указанного стратегией перемещения[13], то его перемещение откладывается. После окончания перебора игровой симулятор снова итерируется по всем волшебникам и пытается переместить тех, чья позиция в данный тик ещё не изменялась. Так происходит до тех пор, пока не будут перемещены все волшебники. Если на очередной итерации не было перемещено ни одного волшебника, то операция также прерывается.

- Все волшебники поворачиваются на указанный стратегией угол.

Помимо выше перечисленных действий, верховный волшебник может отправлять сообщения другим волшебникам из своей фракции. Поле `message.line` содержит рекомендацию отправляться на соответствующую дорожку. Поле `message.skillToLearn` становится доступным в Раунде 2 и содержит рекомендацию изучить указанное умение. Умение может требовать предварительного изучения других умений или быть недоступно для изучения в данный момент в связи с низким уровнем волшебника. Волшебнику рекомендуется запомнить указание и двигаться в направлении его реализации. При этом, более позднее указание должно считаться более приоритетным. В Финале у верховного волшебника появляется возможность отправлять текстовые сообщения. Поле `message.rawMessage` является массивом байт, а его максимальная длина равна 1024. Обычно волшебники получают сообщения на следующий тик после отправки, однако при наличии текстовой части момент получения сообщения будет отложен на `ceil(message.rawMessage.length / 2.0)` игровых тиков. Если в системе уже зарегистрировано сообщение для какого-либо волшебника, но ещё им не получено, то новые сообщения этому волшебнику будут проигнорированы.

---

[12] Задержка применения заклинания «Магическая ракета» может быть ниже в результате изучения некоторых умений.

[13] Волшебник после перемещения частично или полностью находится за пределами карты либо пересекается с каким-либо другим живым юнитом.

2.6 Другие игровые объекты

Каждые 750 тиков база каждой фракции генерирует 3 отряда миньонов: по одному на каждую дорожку. Каждый такой отряд состоит из трёх орков и одного фетиша. Отряд сразу же устремляется по своей дорожке в направлении базы противоположной фракции, атакуя всех противников на пути. Волшебники используют миньонов как пушечное мясо. При этом сами они стараются держаться в безопасной зоне и атакуют противника на расстоянии.

В лесных зонах с некоторой вероятностью могут появляться нейтральные миньоны. Обычно они не агрессивны, однако в случае нанесения урона одному из них все нейтральные миньоны поблизости устремляются к обидчику, атакуя при этом любого, кто встанет у них на пути.

Каждые 2500 тиков на карте может появиться бонус. Если на карте уже присутствует хотя бы один бонус, то новый не будет создан. Бонус создаётся в случайно выбранной точке из двух возможных: $(1200, 1200)$ или $(2800, 2800)$. Если любая часть области появления бонуса уже занята волшебником, то симулятор попытается создать бонус в другой точке из списка. В случае неудачи создание бонуса будет отложено до окончания очередного интервала.

2.7 Столкновения юнитов

- Коллизия живых юнитов между собой, а также с границами карты не допускается игровым симулятором.
- Если расстояние от центра живого юнита до центра снаряда меньше или равно сумме их радиусов, то живой юнит получает урон, а снаряд убирается из игрового мира. При этом, огненный шар взрывается и наносит урон всем живым юнитам, находящимся поблизости.
- Если расстояние от центра волшебника до центра бонуса меньше или равно сумме их радиусов, то волшебник на 2400 тиков приобретает магический статус, в зависимости от типа бонуса:
  - `EMPOWER` в 2 раза увеличивает урон, наносимый волшебником при ударах посохом и попаданиях магических снарядов в цель.
  - `HASTE` ускоряет перемещение волшебника, аналогично одноимённому заклинанию.
  - `SHIELD` уменьшает урон, получаемый волшебником, аналогично одноимённому заклинанию.

Все типы столкновений, не описанные в данном документе, игнорируются игровым симулятором.

2.8 Начисление баллов

Волшебник получает опыт в том же количестве и в тех же случаях, когда игрок, управляющий им, получает баллы, за исключением случаев, когда волшебник в тик получения баллов мёртв и ожидает возрождения. Баллы начисляются за следующие дествия:

- Нанесение повреждений зданиям или волшебникам противоположной фракции. Повреждения учитываются с коэффициентом 0.25. Округление производится вниз до ближайшего целого числа. Баллы за повреждения начисляются только тому игроку, волшебник которого нанёс эти повреждения.
- Уничтожение зданий или волшебников противоположной фракции, а также миньонов любой фракции, отличной от фракции волшебника. Суммарное вознаграждение за здания и миньонов составляет 25% от максимального количества их жизненной энергии, за волшебников — 100%. Уничтожение может осуществляться как самим волшебником, так и любым другим юнитом из его фракции. Опыт равномерно распределяется между всеми дружественными волшебниками, расстояние от которых до

цели не превышает 600. При этом, если количество таких волшебников больше одного, то суммарное вознаграждение увеличивается на 67%. Округление производится вниз до ближайшего целого числа.

- Подбор любого бонуса приносит 100 баллов/единиц опыта.
- При уничтожении базы противоположной фракции все дружественные игроки получают по 1000 баллов. Игра при этом завершается.

Глава 3

Создание стратегии

3.1 Техническая часть

Сперва для создания стратегии вам необходимо выбрать один из ряда поддерживаемых языков программирования[14]: Java (Oracle JDK 8), C# (Roslyn 1.3+), C++14 (GNU MinGW C++ 6.2+), Python 2 (Python 2.7+), Python 3 (Python 3.5+), Pascal (Free Pascal 3.0+), Ruby (JRuby 9.1+, Oracle JDK 8). Возможно, этот набор будет расширен. На сайте проекта вы можете скачать пользовательский пакет для каждого из языков. Модифицировать в пакете разрешено лишь один файл, который и предназначен для содержания вашей стратегии, например, MyStrategy.java (для Java) или MyStrategy.py (для Python)[15]. Все остальные файлы пакета при сборке стратегии будут замещены стандартными версиями. Однако вы можете добавлять в стратегию свои файлы с кодом. Эти файлы должны находиться в том же каталоге, что и основной файл стратегии. При отправке решения все они должны быть помещены в один ZIP-архив (файлы должны находиться в корне архива). Если вы не добавляете новых файлов в пакет, достаточно отправить сам файл стратегии (с помощью диалога выбора файла) или же вставить его код в текстовое поле.

После того, как вы отправили свою стратегию, она попадает в очередь тестирования. Система сперва попытается скомпилировать пакет с вашими файлами, а затем, если операция прошла успешно, создаст несколько коротких (по 200 тиков) игр разных форматов[16]: 10×1, 10×1 с разблокированными умениями волшебников и 2×5. Для управления волшебником каждого из участников этих игр будет запущен отдельный клиентский процесс с вашей стратегией, и для того, чтобы стратегия считалась принятой (корректной), ни один из экземпляров стратегии не должен «упасть». Игрокам в этих тестовых играх будут даны имена в формате «`<имя_игрока>`», «`<имя_игрока> (2)`», «`<имя_игрока> (3)`» и т.д.

После успешного прохождения описанного процесса ваша посылка получает статус «Принята». Первая успешная посылка одновременно означает и вашу регистрацию в Песочнице. Вам начисляется стартовый рейтинг (1200), и ваша стратегия начинает участвовать в периодических квалификационных боях (смотрите описание Песочницы для получения более подробной информации). Также вам становится доступна функция создания собственных игр, в которых в качестве соперника можно выбирать любую стратегию

---

[14] Для всех языков программирования используются 32-битные версии компиляторов/интерпретаторов.

[15] Исключение составляет C++, для которого можно модифицировать два файла: MyStrategy.cpp и MyStrategy.h. Причём, наличие в архиве файла MyStrategy.cpp является обязательным (иначе стратегия не скомпилируется), а наличие файла MyStrategy.h — опциональным. В случае его отсутствия будет использован стандартный файл из пакета.

[16] Основными параметрами формата игры являются количество игроков, участвующих в нём, и количество юнитов, находящихся под управлением каждого игрока. Кратко формат записывается в виде `<количество_игроков>` × `<количество_юнитов>`, например запись 4×3 означает формат игры, в котором участвует 4 игрока, управляющих тремя юнитами каждый. К формату игры может быть добавлено пояснение в случае, если краткая запись формата для разных этапов чемпионата совпадает. Обратите внимание, что в играх Финала данного чемпионата каждым волшебником одной фракции управляет экземпляр стратегии участника, запущенный в отдельном процессе. С точки зрения игрового API, эти волшебники виртуально будут распределены по пяти различным игрокам, однако определение победителя будет зависеть лишь от суммы их баллов, но не от распределения по конкретным волшебникам.

любого игрока (в том числе и вашу собственную), созданную до момента вашей последней успешной посылки. Созданные вами игры не влияют на рейтинг.

В системе присутствуют ограничения на количество посылок и пользовательских игр, а именно:

- Нельзя отправлять стратегию чаще, чем три раза за пять минут.
- За пять минут нельзя создать более двух пользовательских игр.

Для упрощения отладки небольших изменений стратегии в системе присутствует возможность сделать тестовую посылку (флажок «Тестовая посылка» на форме отправки стратегии). Тестовая посылка не отображается другим пользователям, не участвует в квалификационных играх в Песочнице и играх в этапах турнира, также невозможно собственноручно создавать игры с её участием. Однако, после принятия данной посылки, система автоматически добавляет тестовую игру с десятью участниками (формат 10×1): непосредственно тестовой посылкой и девятью стратегиями из раздела «Быстрый старт». Тестовая игра видна только участнику, сделавшему данную тестовую посылку. Базовая длительность такой тестовой игры составляет 2000 тиков. На частоту тестовых посылок действует то же ограничение, что и на частоту обычных посылок. Тестовые игры на частоту создания игр пользователем не влияют.

У игроков есть возможность в специальном визуализаторе просматривать прошедшие игры. Для этого нужно нажать кнопку «Смотреть» в списке игр либо нажать кнопку «Посмотреть игру» на странице игры.

Если вы смотрите игру с участием вашей стратегии и заметили некоторую странность в её поведении, или ваша стратегия делает не то, что вы от неё ожидали, то вы можете воспользоваться специальной утилитой Repeater для воспроизведения локального повтора данной игры. Локальный повтор игры — это возможность запустить стратегию на вашем компьютере так, чтобы она видела игровой мир вокруг себя таким, каким он был при тестировании на сервере. Это поможет вам выполнять отладку, добавлять логирование и наблюдать за реакцией вашей стратегии в каждый момент игры. Для этого скачайте Repeater с сайта CodeWizards 2016 (раздел «Документация» → «Утилита Repeater») и разархивируйте. Для запуска Repeater вам необходимо установленное ПО Java 8+ Runtime Environment. Обратите внимание, что любое взаимодействие вашей стратегии с игровым миром при локальном повторе полностью игнорируется. Это означает, что в каждый момент времени окружающий мир для стратегии в точности совпадает с миром, каким он был в игре при тестировании на сервере и не зависит от того, какие действия ваша стратегия предпринимает. Утилита Repeater располагает только теми данными, которые отправлялись вашей стратегии, но не полной записью игры. Поэтому визуализация игры не осуществляется. Подробнее об утилите Repeater читайте в соответствующем разделе на сайте.

Помимо всего выше перечисленного у игроков есть возможность запускать простые тестовые игры локально на своём компьютере. Для этого необходимо загрузить архив с утилитой Local runner из раздела сайта «Документация» → «Local runner». Использование данной утилиты позволит вам тестировать свою стратегию в условиях, аналогичных условиям тестовой игры на сайте, но без каких либо ограничений по количеству создаваемых игр. Рендерер для локальных игр заметно отличается от рендерера на сайте. Все игровые объекты в нём отображаются схематично (без использования красочных моделей). Создать локальную тестовую игру очень просто: запустите Local runner с помощью соответствующего скрипта запуска (*.bat для Windows или *.sh для *n*x систем), затем запустите свою стратегию из среды разработки (или любым другим удобным вам способом) и смотрите игру. Во время локальных игр вы можете выполнять отладку своей стратегии, ставить точки останова. Однако следует помнить, что Local runner ожидает отклика от стратегии не более 30 минут. По прошествии этого времени он посчитает стратегию «упавшей» и продолжит работу без неё.

3.2 Управление волшебником

Для вашего волшебника в начале игры создаётся объект класса `MyStrategy`, в полях которого стратегия может сохранять информацию о ходе игры. Управление волшебником осуществляется с помощью метода `move` стратегии, который вызывается один раз за тик. Методу передаются следующие параметры:

- волшебник `self`, для которого вызывается метод;

- текущее состояние мира `world`;
- набор игровых констант `game`;
- объект `move`, устанавливая свойства которого, стратегия и определяет поведение волшебника.

Реализация клиента-оболочки стратегии на разных языках может отличаться, однако в общем случае **не** гарантируется, что при разных вызовах метода `move` в качестве параметров ему будут переданы ссылки на одни и те же объекты. Таким образом, нельзя, например, сохранить ссылки на объекты `world` или `player` и получать в следующие тики обновлённую информацию об этих объектах, считывая их поля.

3.3 Примеры реализации

Далее для всех языков программирования приведены простейшие примеры стратегий, которые перемещают волшебника вперёд и вправо, одновременно также поворачивая его направо. Дополнительно стратегия выражает пожелание произнести заклинание «Магическая ракета». Полную документацию по классам и методам для языка Java можно найти в следующих главах.

3.3.1 Пример для Java

```
import model.*;

public final class MyStrategy implements Strategy {
    @Override
    public void move(Wizard self, World world, Game game, Move move) {
        move.setSpeed(game.getWizardForwardSpeed());
        move.setStrafeSpeed(game.getWizardStrafeSpeed());
        move.setTurn(game.getWizardMaxTurnAngle());
        move.setAction(ActionType.MAGIC_MISSILE);
    }
}
```

3.3.2 Пример для C#

```
using Com.CodeGame.CodeWizards2016.DevKit.CSharpCgdk.Model;

namespace Com.CodeGame.CodeWizards2016.DevKit.CSharpCgdk {
    public sealed class MyStrategy : IStrategy {
        public void Move(Wizard self, World world, Game game, Move move) {
            move.Speed = game.WizardForwardSpeed;
            move.StrafeSpeed = game.WizardStrafeSpeed;
            move.Turn = game.WizardMaxTurnAngle;
            move.Action = ActionType.MagicMissile;
        }
    }
}
```

3.3.3 Пример для C++

```
#include "MyStrategy.h"

#define PI 3.14159265358979323846
#define _USE_MATH_DEFINES

#include <cmath>
#include <cstdlib>

using namespace model;
using namespace std;

void MyStrategy::move(const Wizard& self, const World& world, const Game& game, Move& move) {
    move.setSpeed(game.getWizardForwardSpeed());
    move.setStrafeSpeed(game.getWizardStrafeSpeed());
    move.setTurn(game.getWizardMaxTurnAngle());
    move.setAction(MAGIC_MISSILE);
}

MyStrategy::MyStrategy() { }
```

3.3.4 Пример для Python 2

В языке Python 2 имя переменной текущего волшебника изменено с «`self`» на «`me`».

```
from model.ActionType import ActionType
from model.Wizard import Wizard
from model.Game import Game
from model.Move import Move
from model.World import World


class MyStrategy:
    def move(self, me, world, game, move):
        """
        @type me: Wizard
        @type world: World
        @type game: Game
        @type move: Move
        """
        move.speed = game.wizard_forward_speed
        move.strafe_speed = game.wizard_strafe_speed
        move.turn = game.wizard_max_turn_angle
        move.action = ActionType.MAGIC_MISSILE
```

3.3.5 Пример для Python 3

В языке Python 3 имя переменной текущего волшебника изменено с «`self`» на «`me`».

```
from model.ActionType import ActionType
from model.Wizard import Wizard
from model.Game import Game
from model.Move import Move
from model.World import World


class MyStrategy:
    def move(self, me: Wizard, world: World, game: Game, move: Move):
        move.speed = game.wizard_forward_speed
        move.strafe_speed = game.wizard_strafe_speed
        move.turn = game.wizard_max_turn_angle
        move.action = ActionType.MAGIC_MISSILE
```

3.3.6 Пример для Pascal

В языке Pascal имя переменной текущего волшебника изменено с «`self`» на «`me`».

```
unit MyStrategy;

interface

uses
  StrategyControl, TypeControl, ActionTypeControl, BonusControl, BonusTypeControl, BuildingControl,
  CircularUnitControl, FactionControl, GameControl, LaneTypeControl, LivingUnitControl, MessageContr
  MinionTypeControl, MoveControl, PlayerContextControl, PlayerControl, ProjectileControl, Projectile
  SkillTypeControl, StatusControl, StatusTypeControl, TreeControl, UnitControl, WizardControl, World

type
  TMyStrategy = class (TStrategy)
  public
    procedure Move(me: TWizard; world: TWorld; game: TGame; move: TMove); override;

  end;

implementation

uses
  Math;

procedure TMyStrategy.Move(me: TWizard; world: TWorld; game: TGame; move: TMove);
begin
  move.Speed := game.WizardForwardSpeed;
  move.StrafeSpeed := game.WizardStrafeSpeed;
  move.Turn := game.WizardMaxTurnAngle;
  move.Action := ACTION_MAGIC_MISSILE;
end;

end.
```

3.3.7 Пример для Ruby

В языке Ruby имя переменной текущего волшебника изменено с «`self`» на «`me`».

```
require './model/wizard'
require './model/game'
require './model/move'
require './model/world'

class MyStrategy
  # @param [Wizard] me
  # @param [World] world
  # @param [Game] game
  # @param [Move] move
  def move(me, world, game, move)
    move.speed = game.wizard_forward_speed
    move.speed = game.wizard_strafe_speed
    move.turn = game.wizard_max_turn_angle
    move.action = ActionType::MAGIC_MISSILE
  end
end
```

Глава 4

Package model

Package Contents *Page*

Classes

ActionType 27

Возможные действия волшебника.

Bonus 28

Класс, определяющий бонус — неподвижный полезный объект.

BonusType 29

Тип бонуса.

Building 29

Класс, определяющий строение.

BuildingType 30

Тип строения.

CircularUnit 31

Базовый класс для определения круглых объектов.

Faction 31

Фракция юнита.

Game 32

Предоставляет доступ к различным игровым константам.

LaneType 44

Тип дорожки.

LivingUnit 45

Класс, определяющий живого юнита круглой формы.

Message 46

Класс определяет сообщение, которое верховный волшебник (`wizard.master`) может отправлять другим членам фракции, используя телепатическую связь.

Minion 47

Класс, определяющий приспешника волшебника одной из фракций.

MinionType 47

Тип приспешника.

Move 48

Стратегия игрока может управлять волшебником посредством установки свойств объекта данного класса.

Player 52

Содержит данные о текущем состоянии игрока.

Projectile 53

Класс, определяющий снаряд.
ProjectileType 54
Тип снаряда.
SkillType 54
Тип умения.
Status 57
Магический статус, влияющий на живого юнита.
StatusType 58
Тип магического статуса, влияющего на живого юнита.
Tree 59
Класс, определяющий дерево.
Unit 59
Базовый класс для определения объектов («юнитов») на игровом поле.
Wizard 61
Класс, определяющий волшебника.
World 63
Этот класс описывает игровой мир.

4.1 Classes

4.1.1 CLASS **ActionType**

Возможные действия волшебника.

Волшебник не может совершать новые действия, если он ещё не восстановился после своего предыдущего действия (значение `wizard.remainingActionCooldownTicks` больше `0`).

Волшебник не может использовать действие, если оно ещё не восстановилось после его предыдущего применения (значение `remainingCooldownTicksByAction[actionType.ordinal()]` больше `0`).

DECLARATION

public final class ActionType

extends Enum

FIELDS

- public static final ActionType NONE
  - Ничего не делать.
- public static final ActionType STAFF
  - Ударить посохом.

    Атака поражает все живые объекты в секторе от `-game.staffSector / 2.0` до `game.staffSector / 2.0`. Расстояние от центра волшебника до центра цели не должно превышать значение `game.staffRange + livingUnit.radius`.
- public static final ActionType MAGIC_MISSILE
  - Создать магическую ракету.

    Магическая ракета является базовым заклинанием любого волшебника. Наносит урон при прямом попадании.

    При создании магической ракеты её центр совпадает с центром волшебника, направление определяется как `wizard.angle + move.castAngle`, а абсолютное значение скорости равно `game.magicMissileSpeed`. Столкновения магического снаряда и создавшего его волшебника игнорируются.

    Требует `game.magicMissileManacost` единиц магической энергии.
- public static final ActionType FROST_BOLT
  - Создать ледяную стрелу.

Ледяная стрела наносит урон при прямом попадании, а также замораживает цель.

При создании ледяной стрелы её центр совпадает с центром волшебника, направление определяется как `wizard.angle + move.castAngle`, а абсолютное значение скорости равно `game.frostBoltSpeed`. Столкновения магического снаряда и создавшего его волшебника игнорируются.

Требует `game.frostBoltManacost` единиц магической энергии и изучения умения `FROST_BOLT`.

- public static final ActionType FIREBALL
  - Создать огненный шар.

    Огненный шар взрывается при достижении максимальной дальности полёта или при столкновении с живым объектом. Наносит урон всем близлежащим живым объектам, а также поджигает их.

    При создании огненного шара его центр совпадает с центром волшебника, направление определяется как `wizard.angle + move.castAngle`, а абсолютное значение скорости равно `game.fireballSpeed`. Столкновения магического снаряда и создавшего его волшебника игнорируются.

    Требует `game.fireballManacost` единиц магической энергии и изучения умения `FIREBALL`.
- public static final ActionType HASTE
  - Временно ускорить волшебника с идентификатором `move.statusTargetId` или самого себя, если такой волшебник не найден.

    Требует `game.hasteManacost` единиц магической энергии и изучения умения `HASTE`.
- public static final ActionType SHIELD
  - На время создать магический щит вокруг волшебника с идентификатором `move.statusTargetId` или самого себя, если такой волшебник не найден.

    Требует `game.shieldManacost` единиц магической энергии и изучения умения `SHIELD`.

4.1.2 CLASS **Bonus**

Класс, определяющий бонус — неподвижный полезный объект. Содержит также все свойства круглого юнита.

DECLARATION

public class Bonus **extends** CircularUnit

METHODS

- *getType*
  `public BonusType` **`getType`**`( )`
  - **Returns** - Возвращает тип бонуса.

4.1.3 CLASS **BonusType**

Тип бонуса.

В дополнение к основному эффекту каждый подобранный бонус даёт игроку `game.bonusScoreAmount` баллов, а волшебник получает такое же количество опыта.

DECLARATION

public final class BonusType **extends** Enum

FIELDS

- public static final BonusType EMPOWER
  - На некоторое время значительно увеличивает урон, наносимый волшебником при ударах посохом и попаданиях магических снарядов в цель.
- public static final BonusType HASTE
  - Значительно ускоряет перемещение волшебника.

    Аналогично действию одноимённого заклинания, но длительность статуса выше.
- public static final BonusType SHIELD
  - Уменьшает урон, получаемый волшебником от прямых попаданий магических снарядов.

    Аналогично действию одноимённого заклинания, но длительность статуса выше.

4.1.4 CLASS **Building**

Класс, определяющий строение. Фракционные строения самостоятельно атакуют противников в определённом радиусе.

Строения не могут быть заморожены (`FROZEN`).

DECLARATION

public class Building

extends LivingUnit

METHODS

- *getAttackRange*
  `public double` **getAttackRange**`( )`
  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором данное строение может атаковать другие объекты.

- *getCooldownTicks*
  `public int` **getCooldownTicks**`( )`
  - **Returns** - Возвращает интервал между атаками.

- *getDamage*
  `public int` **getDamage**`( )`
  - **Returns** - Возвращает урон одной атаки.

- *getRemainingActionCooldownTicks*
  `public int` **getRemainingActionCooldownTicks**`( )`
  - **Returns** - Возвращает количество тиков, оставшееся до следующей атаки.

- *getType*
  `public BuildingType` **getType**`( )`
  - **Returns** - Возвращает тип строения.

- *getVisionRange*
  `public double` **getVisionRange**`( )`
  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором данное строение обнаруживает другие объекты.

4.1.5 CLASS **BuildingType**

Тип строения.

DECLARATION

public final class BuildingType

extends Enum

FIELDS

- public static final BuildingType GUARDIAN_TOWER
  - Охранная башня.
- public static final BuildingType FACTION_BASE
  - База фракции.

4.1.6 CLASS **CircularUnit**

Базовый класс для определения круглых объектов. Содержит также все свойства юнита.

DECLARATION

public abstract class CircularUnit **extends** Unit

METHODS

- *getRadius*
  `public double` **getRadius**`( )`
  - **Returns** - Возвращает радиус объекта.

4.1.7 CLASS **Faction**

Фракция юнита.

DECLARATION

public final class Faction **extends** Enum

FIELDS

- public static final Faction ACADEMY
  - Волшебники, последователи и охранные сооружения Академии.
- public static final Faction RENEGADES
  - Волшебники, последователи и охранные сооружения Отступников.
- public static final Faction NEUTRAL
  - Нейтральные юниты. Не нападют первыми, но при получении урона будут обороняться.
- public static final Faction OTHER
  - Все остальные юниты в игре.

4.1.8 CLASS **Game**

Предоставляет доступ к различным игровым константам.

DECLARATION

public class Game **extends** Object

METHODS

- *getAuraSkillRange*
  `public double` **getAuraSkillRange**`( )`
  - **Returns** - Возвращает дальность действия аур.

- *getBonusAppearanceIntervalTicks*
  `public int` **getBonusAppearanceIntervalTicks**`( )`
  - **Returns** - Возвращает интервал появления бонусов.

    Каждый раз по прошествии указанного интервала симулятор игры проверяет количество бонусов на карте и создаёт новый бонус, если оно равно нулю. Бонус создаётся в случайно выбранной точке из двух возможных: (`mapSize * 0.3`, `mapSize * 0.3`) и (`mapSize * 0.7`, `mapSize * 0.7`). Если любая часть области появления бонуса уже занята волшебником, то симулятор попытается создать бонус в другой точке из списка. В случае неудачи создание бонуса не произойдёт вообще и будет отложено до окончания очередного интервала.

- *getBonusRadius*
  `public double` **getBonusRadius**`( )`

- **Returns** - Возвращает радиус бонуса.

- *getBonusScoreAmount*
`public int` **getBonusScoreAmount**`( )`

  - **Returns** - Возвращает количество баллов, начисляемых игроку, волшебник которого подобрал бонус.

    Сам волшебник получает такое же количество опыта.

- *getBuildingDamageScoreFactor*
`public double` **getBuildingDamageScoreFactor**`( )`

  - **Returns** - Возвращает коэффициент опыта, получаемого волшебником при нанесении урона строениям противоположной фракции.

- *getBuildingEliminationScoreFactor*
`public double` **getBuildingEliminationScoreFactor**`( )`

  - **Returns** - Возвращает коэффициент опыта, получаемого волшебником за разрушение строения противоположной фракции.

    Применяется к максимальному количеству жизненной энергии строения.

- *getBurningDurationTicks*
`public int` **getBurningDurationTicks**`( )`

  - **Returns** - Возвращает длительность действия статуса `BURNING`.

- *getBurningSummaryDamage*
`public int` **getBurningSummaryDamage**`( )`

  - **Returns** - Возвращает суммарный урон, получаемый живым юнитом за время действия статуса `BURNING`.

- *getDartDirectDamage*
`public int` **getDartDirectDamage**`( )`

  - **Returns** - Возвращает урон дротика.

- *getDartRadius*
`public double` **getDartRadius**`( )`

  - **Returns** - Возвращает радиус дротика.

- *getDartSpeed*
`public double` **getDartSpeed**`( )`

  - **Returns** - Возвращает скорость полёта дротика.

- *getEmpoweredDamageFactor*
`public double` **getEmpoweredDamageFactor**`( )`

  - **Returns** - Возвращает мультипликатор урона, наносимого живым юнитом под действием статуса `EMPOWERED`.

    Мультипликатор применяется к ударам в ближнем бою, прямым попаданиям снарядов, а также взрыву «Огненного шара», но не применяется к урону, получаемому от статусов.

- *getEmpoweredDurationTicks*
`public int` **getEmpoweredDurationTicks**`( )`

  - **Returns** - Возвращает длительность действия статуса `EMPOWERED`.

- *getFactionBaseAttackRange*
`public double` **getFactionBaseAttackRange**`( )`

  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором база фракции может атаковать другие объекты.

- *getFactionBaseCooldownTicks*
`public int` **getFactionBaseCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными атаками базы фракции.

- *getFactionBaseDamage*
`public int` **getFactionBaseDamage**`( )`

  - **Returns** - Возвращает урон одной атаки базы фракции.

- *getFactionBaseLife*
`public double` **getFactionBaseLife**`( )`

  - **Returns** - Возвращает начальное значение жизненной энергии базы фракции.

- *getFactionBaseRadius*
`public double` **getFactionBaseRadius**`( )`

  - **Returns** - Возвращает радиус базы фракции.

- *getFactionBaseVisionRange*
`public double` **getFactionBaseVisionRange**`( )`

  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором база фракции обнаруживает другие объекты.

- *getFactionMinionAppearanceIntervalTicks*
`public int` **getFactionMinionAppearanceIntervalTicks**`( )`

  - **Returns** - Возвращает интервал, с которым появляются миньоны двух противостоящих фракций (`ACADEMY` и `RENEGADES`).

    Миньоны каждой из этих фракций появляются тремя группами (по одной на дорожку) недалеко от своей базы. Группа состоит и трёх орков и одного фетиша. Сразу после появления миньоны начинают продвижение по своей дорожке в сторону базы противоположной фракции, при этом атакуя всех противников на своём пути.

- *getFetishBlowdartActionCooldownTicks*
`public int` **getFetishBlowdartActionCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными атаками фетиша.

- *getFetishBlowdartAttackRange*
`public double` **getFetishBlowdartAttackRange**`( )`

  - **Returns** - Возвращает дальность полёта дротика, выпущенного фетишем.

- *getFetishBlowdartAttackSector*
`public double` **getFetishBlowdartAttackSector**`( )`

  - **Returns** - Возвращает сектор метания дротика фетишем.

    Угол полёта дротика относительно направления фетиша ограничен интервалом от `-fetishBlowdartAttackSector / 2.0` до `fetishBlowdartAttackSector / 2.0`.

- *getFireballCooldownTicks*
`public int` **getFireballCooldownTicks**`( )`
  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными заклинаниями «Огненный шар».

- *getFireballExplosionMaxDamage*
`public int` **getFireballExplosionMaxDamage**`( )`
  - **Returns** - Возвращает урон «Огненного шара» в эпицентре взрыва.

    Живой юнит получает `fireballExplosionMaxDamage` единиц урона, если расстояние от центра взрыва до ближайшей точки этого юнита не превышает `fireballExplosionMaxDamageRange`. По мере увеличения расстояния до `fireballExplosionMinDamageRange`, урон «Огненного шара» равномерно снижается и достигает `fireballExplosionMinDamage`. Если расстояние от центра взрыва до ближайшей точки живого юнита превышает `fireballExplosionMinDamageRange`, то урон ему не наносится.

    Если живой юнит получил какой-либо урон от взрыва «Огненного шара», то он загорается (`BURNING`).

- *getFireballExplosionMaxDamageRange*
`public double` **getFireballExplosionMaxDamageRange**`( )`
  - **Returns** - Возвращает радиус области, в которой живые юниты получают максимальный урон от взрыва «Огненного шара».
  - **See Also**
    - `Game.getFireballExplosionMaxDamage()`

- *getFireballExplosionMinDamage*
`public int` **getFireballExplosionMinDamage**`( )`
  - **Returns** - Возвращает урон «Огненного шара» на периферии взрыва.
  - **See Also**
    - `Game.getFireballExplosionMaxDamage()`

- *getFireballExplosionMinDamageRange*
`public double` **getFireballExplosionMinDamageRange**`( )`
  - **Returns** - Возвращает радиус области, в которой живые юниты получают какой-либо урон от взрыва «Огненного шара».
  - **See Also**
    - `Game.getFireballExplosionMaxDamage()`

- *getFireballManacost*
`public int` **getFireballManacost**`( )`
  - **Returns** - Возвращает количество магической энергии, требуемой для заклинания «Огненный шар».

- *getFireballRadius*
`public double` **getFireballRadius**`( )`
  - **Returns** - Возвращает радиус «Огненного шара».

- *getFireballSpeed*
`public double` **getFireballSpeed**`( )`
  - **Returns** - Возвращает скорость полёта «Огненного шара».

- *getFriendlyFireDamageFactor*
`public double` **getFriendlyFireDamageFactor**`( )`

  - **Returns** - Возвращает коэффициент урона, наносимого волшебниками одной фракции друг другу в результате дружественного огня.

    Значение зависит от режима игры, но не может выходить за границы интервала от `0.0` до `1.0`.

    Вне зависимости от режима игры, волшебники не могут наносить урон союзным миньонам и структурам.

- *getFrostBoltCooldownTicks*
`public int` **getFrostBoltCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными заклинаниями «Ледяная стрела».

- *getFrostBoltDirectDamage*
`public int` **getFrostBoltDirectDamage**`( )`

  - **Returns** - Возвращает урон «Ледяной стрелы».

- *getFrostBoltManacost*
`public int` **getFrostBoltManacost**`( )`

  - **Returns** - Возвращает количество магической энергии, требуемой для заклинания «Ледяная стрела».

- *getFrostBoltRadius*
`public double` **getFrostBoltRadius**`( )`

  - **Returns** - Возвращает радиус «Ледяной стрелы».

- *getFrostBoltSpeed*
`public double` **getFrostBoltSpeed**`( )`

  - **Returns** - Возвращает скорость полёта «Ледяной стрелы».

- *getFrozenDurationTicks*
`public int` **getFrozenDurationTicks**`( )`

  - **Returns** - Возвращает длительность действия статуса `FROZEN`.

- *getGuardianTowerAttackRange*
`public double` **getGuardianTowerAttackRange**`( )`

  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором охранная башня может атаковать другие объекты.

- *getGuardianTowerCooldownTicks*
`public int` **getGuardianTowerCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными атаками охранной башни.

- *getGuardianTowerDamage*
`public int` **getGuardianTowerDamage**`( )`

  - **Returns** - Возвращает урон одной атаки охранной башни.

- *getGuardianTowerLife*
`public double` **getGuardianTowerLife**`( )`

- **Returns** - Возвращает начальное значение жизненной энергии охранной башни.

- *getGuardianTowerRadius*
`public double` **getGuardianTowerRadius**`( )`

  - **Returns** - Возвращает радиус охранной башни.

- *getGuardianTowerVisionRange*
`public double` **getGuardianTowerVisionRange**`( )`

  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором охранная башня обнаруживает другие объекты.

- *getHasteCooldownTicks*
`public int` **getHasteCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными заклинаниями «Ускорение».

- *getHasteManacost*
`public int` **getHasteManacost**`( )`

  - **Returns** - Возвращает количество магической энергии, требуемой для заклинания «Ускорение».

- *getHastenedBonusDurationFactor*
`public double` **getHastenedBonusDurationFactor**`( )`

  - **Returns** - Возвращает мультилпикатор длительности действия статуса `HASTENED` в случае подбора бонуса.

- *getHastenedDurationTicks*
`public int` **getHastenedDurationTicks**`( )`

  - **Returns** - Возвращает длительность действия статуса `HASTENED`.

- *getHastenedMovementBonusFactor*
`public double` **getHastenedMovementBonusFactor**`( )`

  - **Returns** - Возвращает относительное увеличение скорости перемещения в результате дествия статуса `HASTENED`.

    Увеличение скорости от действия статуса `HASTENED` и увеличение скорости в результате изучения умений, являющихся пререквизитами умения `HASTE`, являются аддитивными. Таким образом, максимальное значение скорости волшебника составляет `1.0 + 4.0 * movementBonusFactorPerSkillLevel + hastenedMovementBonusFactor` от базовой.

- *getHastenedRotationBonusFactor*
`public double` **getHastenedRotationBonusFactor**`( )`

  - **Returns** - Возвращает относительное увеличение скорости поворота в результате дествия статуса `HASTENED`.

- *getLevelUpXpValues*
`public int[]` **getLevelUpXpValues**`( )`

  - **Returns** - Возвращает последовательность неотрицательных целых чисел.

    Количество чисел равно количеству уровней, которые волшебник может получить в данном режиме игры. Значение с индексом `N` определяет количество опыта, которое необходимо набрать волшебнику уровня `N` для получения следующего уровня. Таким образом, количество опыта, необходимое волшебнику начального уровня для получения уровня `N`, равно сумме первых `N` элементов.

- *getMagicalDamageAbsorptionPerSkillLevel*
`public int` **getMagicalDamageAbsorptionPerSkillLevel**`( )`

  - **Returns** - Возвращает абсолютное уменьшение урона, получаемого волшебником в результате прямых попаданий магических снарядов, взрыва «Огненного шара» и атак строений, за каждое последовательное изучение умения, являющегося одним из пререквизитов умения `SHIELD`.

- *getMagicalDamageBonusPerSkillLevel*
`public int` **getMagicalDamageBonusPerSkillLevel**`( )`

  - **Returns** - Возвращает абсолютное увеличение урона, наносимого волшебником в результате прямых попаданий магических снарядов и взрыва «Огненного шара», за каждое последовательное изучение умения, являющегося одним из пререквизитов умения `FROST_BOLT`.

- *getMagicMissileCooldownTicks*
`public int` **getMagicMissileCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными заклинаниями «Магическая ракета».

- *getMagicMissileDirectDamage*
`public int` **getMagicMissileDirectDamage**`( )`

  - **Returns** - Возвращает урон «Магической ракеты».

- *getMagicMissileManacost*
`public int` **getMagicMissileManacost**`( )`

  - **Returns** - Возвращает количество магической энергии, требуемой для заклинания «Магическая ракета».

- *getMagicMissileRadius*
`public double` **getMagicMissileRadius**`( )`

  - **Returns** - Возвращает радиус «Магической ракеты».

- *getMagicMissileSpeed*
`public double` **getMagicMissileSpeed**`( )`

  - **Returns** - Возвращает скорость полёта «Магической ракеты».

- *getMapSize*
`public double` **getMapSize**`( )`

  - **Returns** - Возвращает размер (ширину и высоту) карты.

- *getMinionDamageScoreFactor*
`public double` **getMinionDamageScoreFactor**`( )`

  - **Returns** - Возвращает коэффициент опыта, получаемого волшебником при нанесении урона миньонам другой фракции.

- *getMinionEliminationScoreFactor*
`public double` **getMinionEliminationScoreFactor**`( )`

  - **Returns** - Возвращает коэффициент опыта, получаемого волшебником за уничтожение миньона другой фракции.

    Применяется к максимальному количеству жизненной энергии миньона.

- *getMinionLife*
`public int` **getMinionLife**`( )`

- **Returns** - Возвращает максимальное значение жизненной энергии миньона.

- *getMinionMaxTurnAngle*
`public double` **getMinionMaxTurnAngle**`( )`

  - **Returns** - Возвращает ограничение на изменение угла поворота миньона за один тик.

- *getMinionRadius*
`public double` **getMinionRadius**`( )`

  - **Returns** - Возвращает радиус миньона.

- *getMinionSpeed*
`public double` **getMinionSpeed**`( )`

  - **Returns** - Возвращает скорость миньона при движении вперёд.

    Миньонам недоступно использование других видов движения, а также перемещение со скоростью, отличной от указанной.

- *getMinionVisionRange*
`public double` **getMinionVisionRange**`( )`

  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором миньон обнаруживает другие объекты.

- *getMovementBonusFactorPerSkillLevel*
`public double` **getMovementBonusFactorPerSkillLevel**`( )`

  - **Returns** - Возвращает относительное увеличение скорости перемещения за каждое последовательное изучение умения, являющегося одним из пререквизитов умения `HASTE`.

    Увеличение скорости от действия статуса `HASTENED` и увеличение скорости в результате изучения умений, являющихся пререквизитами умения `HASTE`, являются аддитивными. Таким образом, максимальное значение скорости волшебника составляет `1.0 + 4.0 * movementBonusFactorPerSkillLevel + hastenedMovementBonusFactor` от базовой.

- *getOrcWoodcutterActionCooldownTicks*
`public int` **getOrcWoodcutterActionCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными атаками орка-дровосека.

- *getOrcWoodcutterAttackRange*
`public double` **getOrcWoodcutterAttackRange**`( )`

  - **Returns** - Возвращает дальность действия топора орка.

    Атака топором поражает все живые объекты, для каждого из которых верно, что расстояние от его центра до центра орка-дровосека не превышает значение `orcWoodcutterAttackRange + livingUnit.radius`.

- *getOrcWoodcutterAttackSector*
`public double` **getOrcWoodcutterAttackSector**`( )`

  - **Returns** - Возвращает сектор действия топора орка.

    Атака топором поражает все живые объекты в секторе от `-orcWoodcutterAttackSector / 2.0` до `orcWoodcutterAttackSector / 2.0`.

- *getOrcWoodcutterDamage*
`public int` **getOrcWoodcutterDamage**`( )`

- **Returns** - Возвращает урон одной атаки орка-дровосека.

- *getRandomSeed*
`public long` **getRandomSeed**`( )`

  - **Returns** - Возвращает некоторое число, которое ваша стратегия может использовать для инициализации генератора случайных чисел. Данное значение имеет рекомендательный характер, однако позволит более точно воспроизводить прошедшие игры.

- *getRangeBonusPerSkillLevel*
`public double` **getRangeBonusPerSkillLevel**`( )`

  - **Returns** - Возвращает абсолютное увеличение дальности заклинаний волшебника за каждое последовательное изучение умения, являющегося одним из пререквизитов умения `ADVANCED_MAGIC_MISSILE`.

- *getRawMessageMaxLength*
`public int` **getRawMessageMaxLength**`( )`

  - **Returns** - Возвращает максимально возможную длину низкоуровневого сообщения.

    Сообщения, длина которых превышает указанное значение, будут проигнорированы.

- *getRawMessageTransmissionSpeed*
`public double` **getRawMessageTransmissionSpeed**`( )`

  - **Returns** - Возвращает скорость отправки сообщения.

    Если текстовая часть сообщения пуста, то адресат получит его уже в следующий игровой тик. В противном случае, момент получения сообщения будет отложен на `ceil(message.rawMessage.length / rawMessageTransmissionSpeed)` игровых тиков.

- *getScoreGainRange*
`public double` **getScoreGainRange**`( )`

  - **Returns** - Возвращает максимальное расстояние, на котором волшебник получает опыт при уничтожении союзником юнита другой фракции.

    При уничтожении противника опыт равномерно распределяется между всеми волшебниками, находящимися на расстоянии от цели, на превышающем `scoreGainRange`, а также юнитом, нанёсшим урон, если это тоже волшебник.

    При нанесении противнику урона, не приводящему к уничтожению юнита, данный параметр не применяется, а опыт полностью достаётся атакующему волшебнику. В случае атаки миньона или строения опыт не достаётся никому.

    Учитывается расстояние между центрами юнитов.

- *getShieldCooldownTicks*
`public int` **getShieldCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными заклинаниями «Щит».

- *getShieldedBonusDurationFactor*
`public double` **getShieldedBonusDurationFactor**`( )`

  - **Returns** - Возвращает мультилпикатор длительности действия статуса `SHIELDED` в случае подбора бонуса.

- *getShieldedDirectDamageAbsorptionFactor*
`public double` **getShieldedDirectDamageAbsorptionFactor**`( )`

  - **Returns** - Возвращает часть урона, поглощаемую щитом.

    Снижение урона применяется к ударам в ближнем бою, прямым попаданиям снарядов, а также взрыву «Огненного шара», но не применяется к урону, получаемому от статусов.

- *getShieldedDurationTicks*
`public int` **getShieldedDurationTicks**`( )`

  - **Returns** - Возвращает длительность действия статуса `SHIELDED`.

- *getShieldManacost*
`public int` **getShieldManacost**`( )`

  - **Returns** - Возвращает количество магической энергии, требуемой для заклинания «Щит».

- *getStaffCooldownTicks*
`public int` **getStaffCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между двумя последовательными ударами посохом.

- *getStaffDamage*
`public int` **getStaffDamage**`( )`

  - **Returns** - Возвращает базовый урон удара посохом.

    Эффективный урон может быть выше в результате действия некоторых аур и/или изучения волшебником некоторых умений.

- *getStaffDamageBonusPerSkillLevel*
`public int` **getStaffDamageBonusPerSkillLevel**`( )`

  - **Returns** - Возвращает абсолютное увеличение урона, наносимого волшебником в ближнем бою, за каждое последовательное изучение умения, являющегося одним из пререквизитов умения `FIREBALL`.

- *getStaffRange*
`public double` **getStaffRange**`( )`

  - **Returns** - Возвращает дальность действия посоха волшебника.

    Атака посохом поражает все живые объекты, для каждого из которых верно, что расстояние от его центра до центра волшебника не превышает значение `staffRange + livingUnit.radius`.

- *getStaffSector*
`public double` **getStaffSector**`( )`

  - **Returns** - Возвращает сектор действия посоха волшебника.

    Атака посохом поражает все живые объекты в секторе от `-staffSector / 2.0` до `staffSector / 2.0`. Этим же интервалом ограничены относительный угол снаряда, а также зона применения магического статуса.

- *getTeamWorkingScoreFactor*
`public double` **getTeamWorkingScoreFactor**`( )`

- **Returns** - Возвращает мультипликатор опыта, применяемый в случае уничтожения юнита противника при участии двух или более волшебников.

  После применения мультипликатора количество опыта округляется вниз до ближайшего целого значения.

- *getTickCount*
`public int` **getTickCount**`( )`

  - **Returns** - Возвращает базовую длительность игры в тиках. Реальная длительность может отличаться от этого значения в меньшую сторону. Эквивалентно `world.tickCount`.

- *getVictoryScore*
`public int` **getVictoryScore**`( )`

  - **Returns** - Возвращает количество баллов, получаемых всеми игроками фракции в случае победы — разрушения базы противоположной фракции.

- *getWizardActionCooldownTicks*
`public int` **getWizardActionCooldownTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку между любыми двумя последовательными действиями волшебника.

- *getWizardBackwardSpeed*
`public double` **getWizardBackwardSpeed**`( )`

  - **Returns** - Возвращает базовое ограничение скорости волшебника при движении назад.

    Эффективное ограничение может быть выше в результате действия некоторых аур и/или изучения волшебником некоторых умений, а также в результате действия статуса `HASTENED`.

- *getWizardBaseLife*
`public int` **getWizardBaseLife**`( )`

  - **Returns** - Возвращает максимальное значение жизненной энергии волшебника на уровне `0`.

- *getWizardBaseLifeRegeneration*
`public double` **getWizardBaseLifeRegeneration**`( )`

  - **Returns** - Возвращает количество жизненной энергии, которое волшебник уровня `0` восстанавливает за один тик.

- *getWizardBaseMana*
`public int` **getWizardBaseMana**`( )`

  - **Returns** - Возвращает максимальное значение магической энергии волшебника на уровне `0`.

- *getWizardBaseManaRegeneration*
`public double` **getWizardBaseManaRegeneration**`( )`

  - **Returns** - Возвращает количество магической энергии, которое волшебник уровня `0` восстанавливает за один тик.

- *getWizardCastRange*
`public double` **getWizardCastRange**`( )`

  - **Returns** - Возвращает базовую дальность заклинаний волшебника.

    Эффективная дальность (`wizard.castRange`) может быть выше в результате действия некоторых аур и/или изучения волшебником некоторых умений.

- *getWizardDamageScoreFactor*
`public double` **getWizardDamageScoreFactor**`( )`

  - **Returns** - Возвращает коэффициент опыта, получаемого волшебником при нанесении урона волшебникам противоположной фракции.

- *getWizardEliminationScoreFactor*
`public double` **getWizardEliminationScoreFactor**`( )`

  - **Returns** - Возвращает коэффициент опыта, получаемого волшебником за разрушение телесной оболочки волшебника противоположной фракции.

    Применяется к максимальному количеству жизненной энергии волшебника.

- *getWizardForwardSpeed*
`public double` **getWizardForwardSpeed**`( )`

  - **Returns** - Возвращает базовое ограничение скорости волшебника при движении вперёд.

    Эффективное ограничение может быть выше в результате действия некоторых аур и/или изучения волшебником некоторых умений, а также в результате действия статуса `HASTENED`.

- *getWizardLifeGrowthPerLevel*
`public int` **getWizardLifeGrowthPerLevel**`( )`

  - **Returns** - Возвращает прирост жизненной энергии волшебника за уровень.

- *getWizardLifeRegenerationGrowthPerLevel*
`public double` **getWizardLifeRegenerationGrowthPerLevel**`( )`

  - **Returns** - Возвращает прирост скорости регенерации жизненной энергии волшебника за один уровень.

- *getWizardManaGrowthPerLevel*
`public int` **getWizardManaGrowthPerLevel**`( )`

  - **Returns** - Возвращает прирост магической энергии волшебника за уровень.

- *getWizardManaRegenerationGrowthPerLevel*
`public double` **getWizardManaRegenerationGrowthPerLevel**`( )`

  - **Returns** - Возвращает прирост скорости регенерации магической энергии волшебника за один уровень.

- *getWizardMaxResurrectionDelayTicks*
`public int` **getWizardMaxResurrectionDelayTicks**`( )`

  - **Returns** - Возвращает максимально возможную задержку возрождения волшебника после смерти его телесной оболочки.

    Если волшебник погибает сразу после своего возрождения, то он будет автоматически воскрешён на своей начальной позиции (или недалеко от неё, если это невозможно) через `wizardMaxResurrectionDelayTicks` тиков. Каждый игровой тик жизни волшебника уменьшшает эту задержку на единицу. Задержка возрождения не может стать меньше, чем `wizardMinResurrectionDelayTicks`.

- *getWizardMaxTurnAngle*
`public double` **getWizardMaxTurnAngle**`( )`

  - **Returns** - Возвращает базовое ограничение на изменение угла поворота волшебника за один тик.

    Эффективное ограничение может быть выше в `1.0 + hastenedRotationBonusFactor` раз в результате действия статуса `HASTENED`.

- *getWizardMinResurrectionDelayTicks*
`public int` **getWizardMinResurrectionDelayTicks**`( )`

  - **Returns** - Возвращает минимально возможную задержку возрождения волшебника после смерти его телесной оболочки.

    Если волшебник погибает сразу после своего возрождения, то он будет автоматически воскрешён на своей начальной позиции (или недалеко от неё, если это невозможно) через `wizardMaxResurrectionDelayTicks` тиков. Каждый игровой тик жизни волшебника уменьшшает эту задержку на единицу. Задержка возрождения не может стать меньше, чем `wizardMinResurrectionDelayTicks`.

- *getWizardRadius*
`public double` **getWizardRadius**`( )`

  - **Returns** - Возвращает радиус волшебника.

- *getWizardStrafeSpeed*
`public double` **getWizardStrafeSpeed**`( )`

  - **Returns** - Возвращает базовое ограничение скорости волшебника при движении боком.

    Эффективное ограничение может быть выше в результате действия некоторых аур и/или изучения волшебником некоторых умений, а также в результате действия статуса `HASTENED`.

- *getWizardVisionRange*
`public double` **getWizardVisionRange**`( )`

  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором волшебник обнаруживает другие объекты.

- *isRawMessagesEnabled*
`public boolean` **isRawMessagesEnabled**`( )`

  - **Returns** - Возвращает `true`, если и только если верховные волшебники в данной игре могут передавать низкоуровневые сообщения другим волшебникам своей фракции.

- *isSkillsEnabled*
`public boolean` **isSkillsEnabled**`( )`

  - **Returns** - Возвращает `true`, если и только если в данной игре волшебники могут повышать свой уровень (накапливая опыт) и изучать новые умения.

4.1.9 CLASS **LaneType**

Тип дорожки.

DECLARATION

public final class LaneType **extends** Enum

FIELDS

- public static final LaneType TOP
  - Верхняя. Проходит через левый нижний, левый верхний и правый верхний углы карты.
- public static final LaneType MIDDLE
  - Центральная. Напрямую соединяет левый нижний и правый верхний углы карты.
- public static final LaneType BOTTOM
  - Нижняя. Проходит через левый нижний, правый нижний и правый верхний углы карты.

4.1.10 CLASS **LivingUnit**

Класс, определяющий живого юнита круглой формы.

DECLARATION

public abstract class LivingUnit **extends** CircularUnit

METHODS

- *getLife*
  `public int` **getLife**`( )`
  - **Returns** - Возвращает текущее количество жизненной энергии.
- *getMaxLife*
  `public int` **getMaxLife**`( )`
  - **Returns** - Возвращает максимальное количество жизненной энергии.
- *getStatuses*
  `public Status[]` **getStatuses**`( )`
  - **Returns** - Возвращает магические статусы, влияющие на живого юнита.

4.1.11 CLASS **Message**

Класс определяет сообщение, которое верховный волшебник (`wizard.master`) может отправлять другим членам фракции, используя телепатическую связь.

Сообщение отправляется персонально каждому волшебнику. Другие волшебники не могут его перехватить.

Адресат получает сообщение в следующий игровой тик или позднее, в зависимости от размера сообщения.

Волшебник волен проигнорировать как любую отдельную часть сообщения, так и всё сообщение целиком, однако это может привести к поражению дружественной фракции.

DECLARATION

public class Message **extends** Object

METHODS

- *getLane*
  `public LaneType` **getLane**`( )`
  - **Returns** - Возвращает указание контролировать определённую дорожку.

- *getRawMessage*
  `public byte[]` **getRawMessage**`( )`
  - **Returns** - Возвращает текстовое сообщение на забытом древнем языке.

    Максимальная длина сообщения составляет `game.rawMessageMaxLength`. При этом, скорость отправки сообщения зависит от его длины. Если текстовая часть сообщения пуста, то адресат получит его уже в следующий игровой тик. В противном случае, момент получения сообщения будет отложен на `ceil(rawMessage.length / game.rawMessageTransmissionSpeed)` игровых тиков.

    Значение поля может быть доступно не во всех режимах игры.

- *getSkillToLearn*
  `public SkillType` **getSkillToLearn**`( )`
  - **Returns** - Возвращает указание изучить какое-либо умение.

    Умение может требовать предварительного изучения других умений или быть недоступно для изучения в данный момент в связи с низким уровнем волшебника. Волшебнику рекомендуется запомнить указание и двигаться в направлении его реализации. При этом, более позднее указание должно считаться более приоритетным.

    Значение поля может быть доступно не во всех режимах игры.

4.1.12 CLASS **Minion**

Класс, определяющий приспешника волшебника одной из фракций. Содержит также все свойства живого юнита.

Миньоны, оставшиеся по той или иной причине без хозяина, часто объединяются в небольшие группы и селятся в лесах. Они крайне настороженно относятся ко всем другим волшебникам и их миньонам.

DECLARATION

public class Minion **extends** LivingUnit

METHODS

- *getCooldownTicks*
  `public int` **getCooldownTicks**`( )`
  - **Returns** - Возвращает интервал между атаками.
- *getDamage*
  `public int` **getDamage**`( )`
  - **Returns** - Возвращает урон одной атаки.
- *getRemainingActionCooldownTicks*
  `public int` **getRemainingActionCooldownTicks**`( )`
  - **Returns** - Возвращает количество тиков, оставшееся до следующей атаки.
- *getType*
  `public MinionType` **getType**`( )`
  - **Returns** - Возвращает тип миньона.
- *getVisionRange*
  `public double` **getVisionRange**`( )`
  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором данный миньон обнаруживает другие объекты.

4.1.13 CLASS **MinionType**

Тип приспешника.

DECLARATION

public final class MinionType

extends Enum

FIELDS

- public static final MinionType ORC_WOODCUTTER
  - Боец ближнего боя и, по совместительству, мастер на все руки. Помогает волшебнику в хозяйстве.

    Не так силён, как воин орков, но для потерявшего бдительность противника может быть весьма опасен.
- public static final MinionType FETISH_BLOWDART
  - Магическое создание, поражающее противников хозяина острыми дротиками. В мирное время занимается охотой.

4.1.14 CLASS **Move**

Стратегия игрока может управлять волшебником посредством установки свойств объекта данного класса.

DECLARATION

public class Move

extends Object

METHODS

- *getAction*
  `public ActionType` **`getAction`**`( )`
  - **Returns** - Возвращает текущее действие волшебника.
- *getCastAngle*
  `public double` **`getCastAngle`**`( )`
  - **Returns** - Возвращает текущий угол полёта магического снаряда.

- *getMaxCastDistance*
`public double` **getMaxCastDistance**`( )`
  - **Returns** - Возвращает текущую установку для дальней границы боевого применения магического снаряда.

- *getMessages*
`public Message[]` **getMessages**`( )`
  - **Returns** - Возвращает текущие сообщения для волшебников дружественной фракции.

- *getMinCastDistance*
`public double` **getMinCastDistance**`( )`
  - **Returns** - Возвращает текущую установку для ближней границы боевого применения магического снаряда.

- *getSkillToLearn*
`public SkillType` **getSkillToLearn**`( )`
  - **Returns** - Возвращает выбранное для изучения умение.

- *getSpeed*
`public double` **getSpeed**`( )`
  - **Returns** - Возвращает текущую установку скорости перемещения.

- *getStatusTargetId*
`public long` **getStatusTargetId**`( )`
  - **Returns** - Возвращает идентификатор текущей цели для применения магического статуса.

- *getStrafeSpeed*
`public double` **getStrafeSpeed**`( )`
  - **Returns** - Возвращает текущую установку скорости перемещения боком.

- *getTurn*
`public double` **getTurn**`( )`
  - **Returns** - Возвращает текущий угол поворота волшебника.

- *setAction*
`public void` **setAction**`( ActionType` **action** `)`
  - **Usage**
    * Устанавливает действие волшебника.

      Действие может быть проигнорировано игровым симулятором, если у волшебника недостаточно магической энергии для его совершения и/или волшебник ещё не успел восстановиться после предыдущего действия.

- *setCastAngle*
`public void` **setCastAngle**`( double` **castAngle** `)`
  - **Usage**

* Устанавливает угол полёта магического снаряда.

  Угол полёта задаётся в радианах относительно текущего направления волшебника и ограничен интервалом от `-game.staffSector / 2.0` до `game.staffSector / 2.0`.

  Значения, выходящие за интервал, будут приведены к ближайшей его границе. Положительные значения соответствуют повороту по часовой стрелке.

  Параметр будет проигнорирован игровым симулятором, если действие волшебника не связано с созданием магического снаряда.

- *setMaxCastDistance*
  `public void` **setMaxCastDistance**`( double` **maxCastDistance** `)`

  - **Usage**
    * Устанавливает дальнюю границу боевого применения магического снаряда.

      Если расстояние от центра снаряда до точки его появления больше, чем значение данного параметра, то снаряд убирается из игрового мира. При этом, снаряд типа `FIREBALL` детонирует.

      Значение параметра по умолчанию заведомо выше максимальной дальности полёта любого типа снарядов в игре.

      Параметр будет проигнорирован игровым симулятором, если действие волшебника не связано с созданием магического снаряда.

- *setMessages*
  `public void` **setMessages**`( Message[]` **messages** `)`

  - **Usage**
    * Устанавливает сообщения для волшебников дружественной фракции.

      Доступно для использования только верховному волшебнику (`wizard.master`). Если используется, количество сообщений должно быть строго равно количеству волшебников дружественной фракции (живых или ожидающих возрождения) за исключением самого верховного волшебника. Нарушение данных условий может привести к игнорированию параметра игровым симулятором или даже к обрыву соединения со стратегией участника.

      Сообщения адресуются в порядке возрастания идентификаторов волшебников. Отдельные сообщения могут быть пустыми (равны `null`), если это поддерживается языком программирования, который использует стратегия. В противном случае все элементы должны быть корректными сообщениями.

      Игровой симулятор вправе проигнорировать сообщение конкретному волшебнику, если для него в системе уже зарегистрировано и ещё им не получено другое сообщение. Если в тик получения сообщения волшебник мёртв, то данное сообщение будет удалено из игрового мира и волшебник никогда его не получит.

      Отправка сообщений доступна не во всех режимах игры.

- *setMinCastDistance*
  `public void` **setMinCastDistance**`( double` **minCastDistance** `)`

  - **Usage**
    * Устанавливает ближнюю границу боевого применения магического снаряда.

      Если расстояние от центра снаряда до точки его появления меньше, чем значение данного

параметра, то боевые свойства снаряда игнорируются. Снаряд беспрепятственно проходит сквозь все другие игровые объекты, за исключением деревьев.

Значение параметра по умолчанию равно `0.0`. Столкновения снаряда и юнита, который его создал, игнорируются.

Параметр будет проигнорирован игровым симулятором, если действие волшебника не связано с созданием магического снаряда.

- *setSkillToLearn*
`public void` **setSkillToLearn**`( SkillType` **skillToLearn** `)`
  - **Usage**
    - Задаёт установку изучить указанное умение до начала следующего игрового тика.

    Установка будет проигнорирована игровым симулятором, если текущий уровень волшебника меньше либо равен количеству уже изученных умений. Некоторые умения также могут требовать предварительного изучения других умений.

    Изучение умений доступно не во всех режимах игры.

- *setSpeed*
`public void` **setSpeed**`( double` **speed** `)`
  - **Usage**
    - Задаёт установку скорости перемещения на один тик.

    Установка скорости перемещения по умолчанию лежит в интервале от `-game.wizardBackwardSpeed` до `game.wizardForwardSpeed`, однако границы интервала могут быть расширены в зависимости от изученных волшебником умений, от действия некоторых аур, а также в случае действия статуса `HASTENED`.

    Значения, выходящие за интервал, будут приведены к ближайшей его границе. Положительные значения соответствуют движению вперёд.

    Если `hypot(speed / maxSpeed, strafeSpeed / maxStrafeSpeed)` больше `1.0`, то обе установки скорости перемещения (`speed` и `strafeSpeed`) будут поделены игровым симулятором на это значение.

- *setStatusTargetId*
`public void` **setStatusTargetId**`( long` **statusTargetId** `)`
  - **Usage**
    - Устанавливает идентификатор цели для применения магического статуса.

    Допустимыми целями являются только волшебники дружественной фракции. Если волшебник с указанным идентификатором не найден, то статус применяется непосредственно к волшебнику, совершающему действие. Относительный угол до цели должен лежать в интервале от `-game.staffSector / 2.0` до `game.staffSector / 2.0`, а максимальная дистанция ограничена дальностью полёта магического снаряда этого волшебника. Её базовое значение равно `game.wizardCastRange`, однако оно может быть увеличено после изучения некоторых умений.

    Значение параметра по умолчанию равно `-1`.

    Параметр будет проигнорирован игровым симулятором, если действие волшебника не связано с применением магического статуса.

- *setStrafeSpeed*
`public void` **setStrafeSpeed**`( double` **strafeSpeed** `)`

  - **Usage**
    * Задаёт установку скорости перемещения боком на один тик.

      Установка скорости перемещения по умолчанию лежит в интервале от `-game.wizardStrafeSpeed` до `game.wizardStrafeSpeed`, однако границы интервала могут быть расширены в зависимости от изученных волшебником умений, от действия некоторых аур, а также в случае действия статуса `HASTENED`.

      Значения, выходящие за интервал, будут приведены к ближайшей его границе. Положительные значения соответствуют движению направо.

      Если `hypot(speed / maxSpeed, strafeSpeed / maxStrafeSpeed)` больше `1.0`, то обе установки скорости перемещения (`speed` и `strafeSpeed`) будут поделены игровым симулятором на это значение.

- *setTurn*
`public void` **setTurn**`( double` **turn** `)`

  - **Usage**
    * Устанавливает угол поворота волшебника.
      Угол поворота задаётся в радианах относительно текущего направления волшебника и обычно ограничен интервалом от `-game.wizardMaxTurnAngle` до `game.wizardMaxTurnAngle`. Если на волшебника действует магический статус `HASTENED`, то нижнюю и правую границу интервала необходимо умножить на `1.0 + game.hastenedRotationBonusFactor`.

      Значения, выходящие за интервал, будут приведены к ближайшей его границе. Положительные значения соответствуют повороту по часовой стрелке.

4.1.15 CLASS **Player**

Содержит данные о текущем состоянии игрока.

DECLARATION

---

public class Player **extends** Object

METHODS

- *getFaction*
`public Faction` **getFaction**`( )`

  - **Returns** - Возвращает фракцию, к которой принадлежит данный игрок.

- *getId*
`public long` **getId**`( )`
  - **Returns** - Возвращает уникальный идентификатор игрока.

- *getName*
`public String` **getName**`( )`
  - **Returns** - Возвращает имя игрока.

- *getScore*
`public int` **getScore**`( )`
  - **Returns** - Возвращает количество баллов, набранное игроком.

- *isMe*
`public boolean` **isMe**`( )`
  - **Returns** - Возвращает `true` в том и только в том случае, если этот игрок ваш.

- *isStrategyCrashed*
`public boolean` **isStrategyCrashed**`( )`
  - **Returns** - Возвращает специальный флаг — показатель того, что стратегия игрока «упала». Более подробную информацию можно найти в документации к игре.

4.1.16 CLASS **Projectile**

Класс, определяющий снаряд. Содержит также все свойства круглого юнита.

DECLARATION

public class Projectile

extends CircularUnit

METHODS

- *getOwnerPlayerId*
`public long` **getOwnerPlayerId**`( )`
  - **Returns** - Возвращает идентификатор игрока, юнит которого создал данный снаряд или `-1`.

- *getOwnerUnitId*
`public long` **getOwnerUnitId**`( )`
  - **Returns** - Возвращает идентификатор юнита, создавшего данный снаряд.

- *getType*
`public ProjectileType` **getType**`( )`
  - **Returns** - Возвращает тип снаряда.

4.1.17 CLASS **ProjectileType**

Тип снаряда.

DECLARATION

public final class ProjectileType

extends Enum

FIELDS

- public static final ProjectileType MAGIC_MISSILE
  - Магическая ракета. Небольшой сгусток чистой энергии, который наносит урон при прямом попадании.
- public static final ProjectileType FROST_BOLT
  - Ледяная стрела. Наносит урон при прямом попадании, а также замораживает цель на `game.frozenDurationTicks` тиков. Замороженная цель не может перемещаться и совершать какие-либо действия.
- public static final ProjectileType FIREBALL
  - Огненный шар. Взрывается при достижении максимальной дальности полёта или при столкновении с живым объектом. Наносит урон всем живым объектам, если расстояние от центра шара до центра объекта не превышает `game.fireballExplosionMinDamageRange + livingUnit.radius`, а также поджигает их (`BURNING`). Мгновенный урон уменьшается по мере удаления от эпицентра взрыва.
- public static final ProjectileType DART
  - Дротик. Заострённый предмет, летящий на большой скорости. Наносит урон при прямом попадании.

4.1.18 CLASS **SkillType**

Тип умения. Изучение умений может быть доступно не во всех режимах игры (смотрите документацию к `game.skillsEnabled`).

Умения делятся на три категории: активные, пассивные и ауры.

- Активные умения наделяют волшебника способностью использовать определённое действие, недоступное ранее.
- Пассивные умения действуют постоянно, улучшая одну из характеристик волшебника на некоторое значение. При наличии нескольких пассивных умений, влияющих на одну характеристику, учитывается только то, которое даёт максимальный эффект.

- Ауры действуют постоянно, улучшая на некоторое значение одну из характеристик самого волшебника, а также всех союзных волшебников на расстоянии, не превышающем `game.auraSkillRange`. При наличии нескольких аур, влияющих на одну характеристику, учитывается только та, которая даёт максимальный эффект.

DECLARATION

public final class SkillType

extends Enum

FIELDS

- public static final SkillType RANGE_BONUS_PASSIVE_1
  - Пассивное умение. Увеличивает максимально возможную дальность полёта магического снаряда, а также дальность применения магических статусов на `game.rangeBonusPerSkillLevel`.
- public static final SkillType RANGE_BONUS_AURA_1
  - Аура. Увеличивает максимально возможную дальность полёта магического снаряда, а также дальность применения магических статусов на `game.rangeBonusPerSkillLevel`.

    Требуется предварительно изучить умение `RANGE_BONUS_PASSIVE_1`.
- public static final SkillType RANGE_BONUS_PASSIVE_2
  - Пассивное умение. Увеличивает максимально возможную дальность полёта магического снаряда, а также дальность применения магических статусов на `2.0 * game.rangeBonusPerSkillLevel`.

    Требуется предварительно изучить умение `RANGE_BONUS_AURA_1`.
- public static final SkillType RANGE_BONUS_AURA_2
  - Аура. Увеличивает максимально возможную дальность полёта магического снаряда, а также дальность применения магических статусов на `2.0 * game.rangeBonusPerSkillLevel`.

    Требуется предварительно изучить умение `RANGE_BONUS_PASSIVE_2`.
- public static final SkillType ADVANCED_MAGIC_MISSILE
  - Пассивное умение. Убирает задержку на применение действия `MAGIC_MISSILE`. Общая задержка на действия волшебника `game.wizardActionCooldownTicks` всё ещё применяется.

    Требуется предварительно изучить умение `RANGE_BONUS_AURA_2`.
- public static final SkillType MAGICAL_DAMAGE_BONUS_PASSIVE_1
  - Пассивное умение. Увеличивает урон, наносимый при прямом попадании магического снаряда, на `game.magicalDamageBonusPerSkillLevel`.
- public static final SkillType MAGICAL_DAMAGE_BONUS_AURA_1
  - Аура. Увеличивает урон, наносимый при прямом попадании магического снаряда, на `game.magicalDamageBonusPerSkillLevel`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_BONUS_PASSIVE_1`.

- public static final SkillType MAGICAL_DAMAGE_BONUS_PASSIVE_2
  - Пассивное умение. Увеличивает урон, наносимый при прямом попадании магического снаряда, на `2.0 * game.magicalDamageBonusPerSkillLevel`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_BONUS_AURA_1`.
- public static final SkillType MAGICAL_DAMAGE_BONUS_AURA_2
  - Аура. Увеличивает урон, наносимый при прямом попадании магического снаряда, на `2.0 * game.magicalDamageBonusPerSkillLevel`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_BONUS_PASSIVE_2`.
- public static final SkillType FROST_BOLT
  - Активное умение. Позволяет волшебнику использовать действие `FROST_BOLT`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_BONUS_AURA_2`.
- public static final SkillType STAFF_DAMAGE_BONUS_PASSIVE_1
  - Пассивное умение. Увеличивает урон, наносимый при ударе посохом, на `game.staffDamageBonusPerSkillLevel`.
- public static final SkillType STAFF_DAMAGE_BONUS_AURA_1
  - Аура. Увеличивает урон, наносимый при ударе посохом, на `game.staffDamageBonusPerSkillLevel`.

    Требуется предварительно изучить умение `STAFF_DAMAGE_BONUS_PASSIVE_1`.
- public static final SkillType STAFF_DAMAGE_BONUS_PASSIVE_2
  - Пассивное умение. Увеличивает урон, наносимый при ударе посохом, на `2.0 * game.staffDamageBonusPerSkillLevel`.

    Требуется предварительно изучить умение `STAFF_DAMAGE_BONUS_AURA_1`.
- public static final SkillType STAFF_DAMAGE_BONUS_AURA_2
  - Аура. Увеличивает урон, наносимый при ударе посохом, на `2.0 * game.staffDamageBonusPerSkillLevel`.

    Требуется предварительно изучить умение `STAFF_DAMAGE_BONUS_PASSIVE_2`.
- public static final SkillType FIREBALL
  - Активное умение. Позволяет волшебнику использовать действие `FIREBALL`.

    Требуется предварительно изучить умение `STAFF_DAMAGE_BONUS_AURA_2`.
- public static final SkillType MOVEMENT_BONUS_FACTOR_PASSIVE_1
  - Пассивное умение. Увеличивает скорость перемещения в `1.0 + game.movementBonusFactorPerSkillLevel` раз.

    Увеличение скорости от изучения пассивных умений и увеличение скорости в результате действия аур аддитивны. Таким образом, умения `MOVEMENT_BONUS_FACTOR_PASSIVE_2` и `MOVEMENT_BONUS_FACTOR_AURA_2` суммарно увеличат скорость перемещения в `1.0 + 4.0 * game.movementBonusFactorPerSkillLevel` раз.
- public static final SkillType MOVEMENT_BONUS_FACTOR_AURA_1
  - Аура. Увеличивает скорость перемещения в `1.0 + game.movementBonusFactorPerSkillLevel` раз.

    Требуется предварительно изучить умение `MOVEMENT_BONUS_FACTOR_PASSIVE_1`.

- public static final SkillType MOVEMENT_BONUS_FACTOR_PASSIVE_2
  - Пассивное умение. Увеличивает скорость перемещения в `1.0 + 2.0 * game.movementBonusFactorPerSkillLevel` раз.

    Требуется предварительно изучить умение `MOVEMENT_BONUS_FACTOR_AURA_1`.
- public static final SkillType MOVEMENT_BONUS_FACTOR_AURA_2
  - Аура. Увеличивает скорость перемещения в `1.0 + 2.0 * game.movementBonusFactorPerSkillLevel` раз.

    Требуется предварительно изучить умение `MOVEMENT_BONUS_FACTOR_PASSIVE_2`.
- public static final SkillType HASTE
  - Активное умение. Позволяет волшебнику использовать действие `HASTE`.

    Требуется предварительно изучить умение `MOVEMENT_BONUS_FACTOR_AURA_2`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_PASSIVE_1
  - Пассивное умение. Уменьшает урон, получаемый при прямом попадании магического снаряда, на `game.magicalDamageAbsorptionPerSkillLevel`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_AURA_1
  - Аура. Уменьшает урон, получаемый при прямом попадании магического снаряда, на `game.magicalDamageAbsorptionPerSkillLevel`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_ABSORPTION_PASSIVE_1`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_PASSIVE_2
  - Пассивное умение. Уменьшает урон, получаемый при прямом попадании магического снаряда, на `2.0 * game.magicalDamageAbsorptionPerSkillLevel`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_ABSORPTION_AURA_1`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_AURA_2
  - Аура. Уменьшает урон, получаемый при прямом попадании магического снаряда, на `2.0 * game.magicalDamageAbsorptionPerSkillLevel`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_ABSORPTION_PASSIVE_2`.
- public static final SkillType SHIELD
  - Активное умение. Позволяет волшебнику использовать действие `SHIELD`.

    Требуется предварительно изучить умение `MAGICAL_DAMAGE_ABSORPTION_AURA_2`.

4.1.19 CLASS **Status**

Магический статус, влияющий на живого юнита.

DECLARATION

public class Status

extends Object

METHODS

- *getId*
`public long` **getId**`( )`
  - **Returns** - Возвращает уникальынй идентификатор статуса.

- *getPlayerId*
`public long` **getPlayerId**`( )`
  - **Returns** - Возвращает идентификатор игрока, волшебник которого наложил данный статус, или {code -1}.

- *getRemainingDurationTicks*
`public int` **getRemainingDurationTicks**`( )`
  - **Returns** - Возвращает оставшуюся длительность действия статуса.

- *getType*
`public StatusType` **getType**`( )`
  - **Returns** - Возвращает тип магического статуса.

- *getWizardId*
`public long` **getWizardId**`( )`
  - **Returns** - Возвращает идентификатор волшебника, наложившего данный статус, или {code -1}.

4.1.20 CLASS **StatusType**

Тип магического статуса, влияющего на живого юнита.

DECLARATION

public final class StatusType

extends Enum

FIELDS

- public static final StatusType BURNING
  - Юнит горит. Каждый тик ему наносится некоторый урон.
- public static final StatusType EMPOWERED
  - Юнит наносит больше урона, чем обычно. Не применимо к урону, растянутому во времени.
- public static final StatusType FROZEN
  - Юнит заморожен. Он не может перемещаться и выполнять какие-либо действия.
- public static final StatusType HASTENED
  - Скорость поворота и перемещения юнита увеличена.
- public static final StatusType SHIELDED
  - Юнит получает меньше урона, чем обычно. Не применимо к урону, растянутому во времени.

4.1.21 CLASS **Tree**

Класс, определяющий дерево. Содержит также все свойства живого юнита.

DECLARATION

public class Tree

extends LivingUnit

4.1.22 CLASS **Unit**

Базовый класс для определения объектов («юнитов») на игровом поле.

DECLARATION

public abstract class Unit

extends Object

- *getAngle*
`public final double` **getAngle**`( )`
  - **Returns** - Возвращает угол поворота объекта в радианах. Нулевой угол соответствует направлению оси абсцисс. Положительные значения соответствуют повороту по часовой стрелке.

- *getAngleTo*
`public double` **getAngleTo**`( double  x, double  y )`
  - **Parameters**
    - `x` - X-координата точки.
    - `y` - Y-координата точки.
  - **Returns** - Возвращает ориентированный угол `[-PI, PI]` между направлением данного объекта и вектором из центра данного объекта к указанной точке.

- *getAngleTo*
`public double` **getAngleTo**`( Unit  unit )`
  - **Parameters**
    - `unit` - Объект, к центру которого необходимо определить угол.
  - **Returns** - Возвращает ориентированный угол `[-PI, PI]` между направлением данного объекта и вектором из центра данного объекта к центру указанного объекта.

- *getDistanceTo*
`public double` **getDistanceTo**`( double  x, double  y )`
  - **Parameters**
    - `x` - X-координата точки.
    - `y` - Y-координата точки.
  - **Returns** - Возвращает расстояние до точки от центра данного объекта.

- *getDistanceTo*
`public double` **getDistanceTo**`( Unit  unit )`
  - **Parameters**
    - `unit` - Объект, до центра которого необходимо определить расстояние.
  - **Returns** - Возвращает расстояние от центра данного объекта до центра указанного объекта.

- *getFaction*
`public Faction` **getFaction**`( )`
  - **Returns** - Возвращает фракцию, к которой принадлежит данный юнит.

- *getId*
`public long` **getId**`( )`
  - **Returns** - Возвращает уникальный идентификатор объекта.

- *getSpeedX*
`public final double` **getSpeedX**`( )`
  - **Returns** - Возвращает X-составляющую скорости объекта. Ось абсцисс направлена слева направо.

    Для юнитов, способных мгновенно менять свою скорость, возвращается значение перемещения за последний тик.

- *getSpeedY*
`public final double` **`getSpeedY`**`( )`
  - **Returns** - Возвращает Y-составляющую скорости объекта. Ось ординат направлена сверху вниз.

    Для юнитов, способных мгновенно менять свою скорость, возвращается значение перемещения за последний тик.

- *getX*
`public final double` **`getX`**`( )`
  - **Returns** - Возвращает X-координату центра объекта. Ось абсцисс направлена слева направо.

- *getY*
`public final double` **`getY`**`( )`
  - **Returns** - Возвращает Y-координату центра объекта. Ось ординат направлена сверху вниз.

4.1.23 CLASS **Wizard**

Класс, определяющий волшебника. Содержит также все свойства живого юнита.

DECLARATION

public class Wizard **extends** LivingUnit

METHODS

- *getCastRange*
`public double` **`getCastRange`**`( )`
  - **Returns** - Возвращает максимальное расстояние (от центра волшебника), которое может преодолеть выпущенный им магический снаряд.

    Также является максимально возможной дальностью применения заклинаний, накладывающих на цель магический статус (`HASTE` и `SHIELD`).

- *getLevel*
`public int` **`getLevel`**`( )`
  - **Returns** - Возвращает текущий уровень волшебника.

    Начальный уровень каждого волшебника равен `0`, а максимальный — `game.levelUpXpValues.length`.

    В некоторых режимах игры рост уровня волшебника может быть заблокирован.

- *getMana*
`public int` **getMana**`( )`
  - **Returns** - Возвращает текущее количество магической энергии волшебника.

- *getMaxMana*
`public int` **getMaxMana**`( )`
  - **Returns** - Возвращает максимальное количество магической энергии волшебника.

- *getMessages*
`public Message[]` **getMessages**`( )`
  - **Returns** - Возвращает сообщения, предназначенные данному волшебнику, если есть право на их просмотр.

    Стратегия может просматривать только сообщения, адресатом которых является управляемый ею волшебник.

- *getOwnerPlayerId*
`public long` **getOwnerPlayerId**`( )`
  - **Returns** - Возвращает идентификатор игрока, которому принадлежит волшебник.

- *getRemainingActionCooldownTicks*
`public int` **getRemainingActionCooldownTicks**`( )`
  - **Returns** - Возвращает количество тиков, оставшееся до любого следующего действия.

    Для совершения произвольного действия `actionType` необходимо, чтобы оба значения `remainingActionCooldownTicks` и `remainingCooldownTicksByAction[actionType.ordinal()]` были равны нулю.

- *getRemainingCooldownTicksByAction*
`public int[]` **getRemainingCooldownTicksByAction**`( )`
  - **Returns** - Возвращает массив целых неотрицательных чисел. Каждая ячейка массива содержит значение количества тиков, оставшегося до совершения следующего действия с соответствующим индексом.

    Например, `remainingCooldownTicksByAction[0]` соответствует действию `NONE` и всегда равно нулю. `remainingCooldownTicksByAction[1]` соответствует действию `STAFF` и равно количеству тиков, оставшемуся до совершения данного действия. `remainingCooldownTicksByAction[2]` соответствует действию `MAGIC_MISSILE` и так далее.

    Для совершения произвольного действия `actionType` необходимо, чтобы оба значения `remainingActionCooldownTicks` и `remainingCooldownTicksByAction[actionType.ordinal()]` были равны нулю.

- *getSkills*
`public SkillType[]` **getSkills**`( )`
  - **Returns** - Возвращает умения, изученные волшебником.

- *getVisionRange*
`public double` **getVisionRange**`( )`
  - **Returns** - Возвращает максимальное расстояние (от центра до центра), на котором данный волшебник обнаруживает другие объекты.

- *getXp*
`public int` **getXp**`( )`
  - **Returns** - Возвращает количество очков опыта, полученное волшебником в процессе игры.

- *isMaster*
`public boolean` **isMaster**`( )`
  - **Returns** - Возвращает `true` в том и только том случае, если этот волшебник является верховным.

    Количество верховных волшебников в каждой фракции строго равно одному.

- *isMe*
`public boolean` **isMe**`( )`
  - **Returns** - Возвращает `true` в том и только том случае, если этот волшебник ваш.

4.1.24 CLASS **World**

Этот класс описывает игровой мир. Содержит также описания всех игроков и игровых объектов («юнитов»).

DECLARATION

public class World **extends** Object

METHODS

- *getBonuses*
`public Bonus[]` **getBonuses**`( )`
  - **Returns** - Возвращает список видимых бонусов (в случайном порядке). После каждого тика объекты, задающие бонусы, пересоздаются.

- *getBuildings*
`public Building[]` **getBuildings**`( )`
  - **Returns** - Возвращает список видимых строений (в случайном порядке). После каждого тика объекты, задающие строения, пересоздаются.

- *getHeight*
`public double` **getHeight**`( )`
  - **Returns** - Возвращает высоту мира.

- *getMinions*
`public Minion[]` **getMinions**`( )`

- **Returns** - Возвращает список видимых последователей (в случайном порядке). После каждого тика объекты, задающие последователей, пересоздаются.

- *getMyPlayer*
`public Player` **getMyPlayer**`( )`

  - **Returns** - Возвращает вашего игрока.

- *getPlayers*
`public Player[]` **getPlayers**`( )`

  - **Returns** - Возвращает список игроков (в случайном порядке). После каждого тика объекты, задающие игроков, пересоздаются.

- *getProjectiles*
`public Projectile[]` **getProjectiles**`( )`

  - **Returns** - Возвращает список видимых магических снарядов (в случайном порядке). После каждого тика объекты, задающие снаряды, пересоздаются.

- *getTickCount*
`public int` **getTickCount**`( )`

  - **Returns** - Возвращает базовую длительность игры в тиках. Реальная длительность может отличаться от этого значения в меньшую сторону. Эквивалентно `game.tickCount`.

- *getTickIndex*
`public int` **getTickIndex**`( )`

  - **Returns** - Возвращает номер текущего тика.

- *getTrees*
`public Tree[]` **getTrees**`( )`

  - **Returns** - Возвращает список видимых деревьев (в случайном порядке). После каждого тика объекты, задающие деревья, пересоздаются.

- *getWidth*
`public double` **getWidth**`( )`

  - **Returns** - Возвращает ширину мира.

- *getWizards*
`public Wizard[]` **getWizards**`( )`

  - **Returns** - Возвращает список видимых волшебников (в случайном порядке). После каждого тика объекты, задающие волшебников, пересоздаются.

Глава 5

Package <none>

Package Contents *Page*

Interfaces

Strategy 66

Стратегия — интерфейс, содержащий описание методов искусственного интеллекта волшебника.

5.1 Interfaces

5.1.1 INTERFACE Strategy

Стратегия — интерфейс, содержащий описание методов искусственного интеллекта волшебника. Каждая пользовательская стратегия должна реализовывать этот интерфейс. Может отсутствовать в некоторых языковых пакетах, если язык не поддерживает интерфейсы.

DECLARATION

```
public interface Strategy
```

METHODS

- *move*
  `public void` **move**`( Wizard` **self**`, World` **world**`, Game` **game**`, Move` **move** `)`
  - **Usage**
    - Основной метод стратегии, осуществляющий управление волшебником. Вызывается каждый тик для каждого волшебника.
  - **Parameters**
    - `self` - Волшебник, которым данный метод будет осуществлять управление.
    - `world` - Текущее состояние мира.
    - `game` - Различные игровые константы.
    - `move` - Результатом работы метода является изменение полей данного объекта.