CodeHockey 2014

Правила

Версия 1.0.2

Сентябрь — октябрь, 2014

Оглавление

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 для Участников конкурса в состязании за получение звания Победителя Конкурса, занявшего соответствующее призовое место.

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 8 сентября 2014 года до 24.00 часов 19 октября 2014 года по Московскому времени.

Сроки начала и окончания этапов Конкурса:

- первый этап — с 00 часов 00 минут 27 сентября 2014 года до 24 часов 00 минут 28 сентября 2014 года;
- второй этап — с 00 часов 00 минут 4 октября 2014 года до 24 часов 00 минут 5 октября 2014 года;
- третий этап (заключительный) — с 00 часов 00 минут 11 октября 2014 года до 24 часов 00 минут 12 октября 2014 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 8 сентября 2014 года до 24.00 часов 19 октября 2014 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 разделе 2.1.

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 разделе 2.1 данного документа.

Призовой фонд Конкурса формируется за счет средств Организатора конкурса.

Призовой фонд:

- 1 место — Apple Mac Pro;
- 2 место — Apple Macbook Pro 13.3";
- 3 места — Apple Macbook Air 13.3";
- 4-8 места — Apple iPad mini 7.9"16GB Wi-Fi;
- 1-6 места в квалификации (Песочница) — Apple iPod nano 16GB.

Все участники Конкурса, принявшие участие во втором или третьем этапах, будут вознаграждены футболкой.

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, в течение двух месяцев после окончания финального этапа. Срок доставки приза по почтовому адресу, указанному участником, зависит от сроков доставки Почты России. 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 отчетности Организатора конкурса. Перечень документов, которые Победитель обязан предоставить Организатору конкурса, включает в себя:

- копию паспорта Победителя;
- копию свидетельства о постановки 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 CodeHockey 2014

2.1 Общие положения игры и правила проведения турнира

Данное соревнование предоставляет вам возможность проверить свои навыки программирования, создав искусственный интеллект (стратегию), управляющий командой хоккеистов в специальном игровом мире (подробнее об особенностях мира CodeHockey 2014 можно узнать в следующих разделах). В зависимости от этапа соревнования у вас в команде будет от 2 до 6 полевых хоккеистов (при этом одновременно на площадке может находиться не более трёх, остальные должны сидеть на скамейке запасных), а также вратарь. Полевые хоккеисты могут отличаться друг от друга по ряду параметров, однако гарантируется, что начальное расположение и параметры хоккеистов симметричны[1] для обеих стратегий. Помимо хоккеистов в игре присутствует ещё один тип объектов (юнитов): это хоккейная шайба. Вратарь перемещается автоматически, пытаясь оставаться на одной горизонтальной линии с шайбой, управлять им вы не можете.

В каждой игре вам будет противостоять стратегия другого игрока. Как и в настоящем хоккее, ваша команда должна забрасывать шайбу в ворота противника и мешать попаданию шайбы в свои ворота. Команда, забросившая больше шайб, объявляется победителем. Игра может закончиться и ничьей, если обе команды забили одинаковое количество голов. Тогда командам назначается дополнительное время. Если к этому моменту не было забито ни одного гола, то вратари обеих команд убираются из игрового мира. Первый же гол, забитый в дополнительное время, определяет победителя, игра при этом завершается. Команды могут подтвердить ничью, если за дополнительное время не будет заброшено ни одной шайбы.

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. Если победа или поражение произошли в дополнительное время, то рейтинг меняется значительно меньше обычного. 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 мгновенно упадёт, однако по мере участия в играх быстро восстановится и даже станет выше, если ваша стратегия действительно стала эффективнее.

---

[1] Относительно вертикальной линии, проходящей через центр поля.

Начальное значение рейтинга на каждом основном этапе турнира равно 0. За каждую игру участник получает определённое количество единиц рейтинга в зависимости от занятого в ней места. А именно:

- За победу в основное время участник получает 3 единицы рейтинга, его противник получает 0.
- За победу в дополнительное время участник получает 2 единицы рейтинга, его противник получает 1.
- За ничью оба участника получают по 1 единице рейтинга.

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 1 квалификационной игре за час. Жюри оставляет за собой право изменить этот интервал, исходя из пропускной способности тестирующей системы, однако для всех игроков он остаётся постоянной величиной. Игры в Песочнице проходят по правилам, соответствующим 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 900 участников с наибольшим рейтингом (при его равенстве приоритет отдаётся игроку, раньше отправившему последнюю версию своей стратегии).

Этапы турнира:

- Раунд 1 проверит ваши навыки управления командой из двух хоккеистов. Этап пройдёт по упрощённым правилам, о чем подробнее читайте далее. Этот этап, как и все последующие, 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 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 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 играх Раунда 2 будет участвовать по 3 хоккеиста с каждой стороны. Участникам придётся не только координировать возросшее количество юнитов, но и учитывать разницу между ними: на этом этапе вводится понятие атрибутов хоккеистов. Между этапами будет некоторый перерыв, так что у вас есть возможность доработать стратегию. 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. У игрока в распоряжении находится команда из 6 хоккеистов, из которых одновременно на игровой площадке находятся только трое, а остальные сидят на скамейке запасных. В игру вводится понятие выносливости. Немного выносливости тратится на каждое действие хоккеиста. Выносливость восстанавливается сама по себе для всех хоккеистов, но для сидящих в запасе скорость восстановления заметно выше. 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

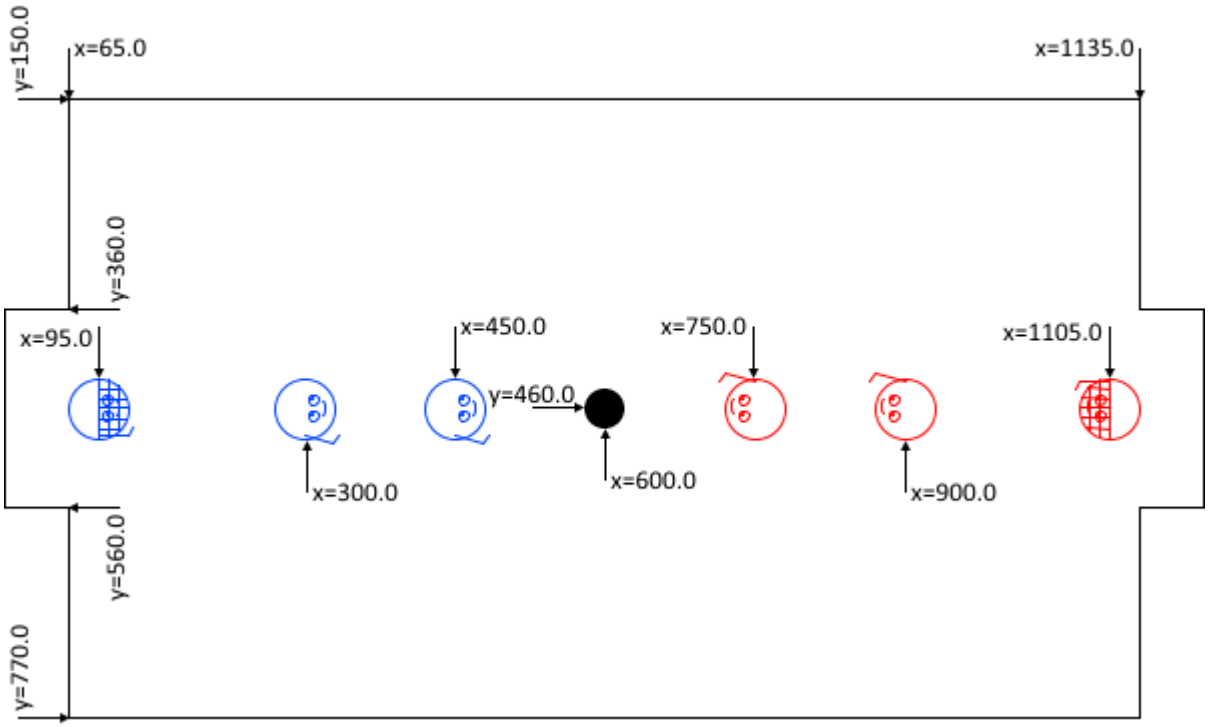
отослана раньше. Призы за Финал распределяются на основании занятого места после этого упорядочивания. Лучшие восемь финалистов награждаются призами:

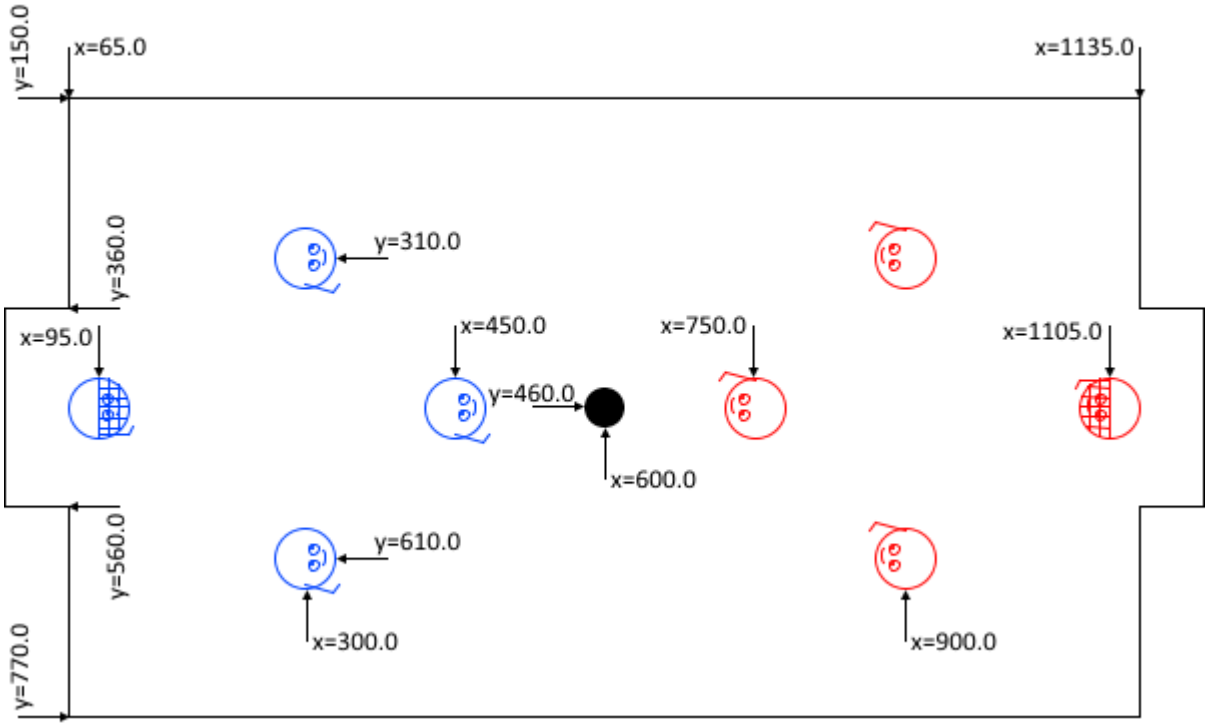
- 1 место — Apple Mac Pro;
- 2 место — Apple Macbook Pro 13.3";
- 3 места — Apple Macbook Air 13.3";
- 4-8 места — Apple iPad mini 7.9"16GB Wi-Fi.

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 представляет собой некоторую двумерную прямоугольную область. Размер области — 1200 × 800. 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. Ниже приведены две схемы игровой площадки с начальным расположением команд из двух и из трёх хоккеистов:





Время в игре дискретное и измеряется в «тиках». В начале каждого «тика» игра получает от стратегий желаемые действия хоккеистов в этот тик и обновляет состояние хоккеистов 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. Базовая длительность каждой игры — 6000 тиков. В случае равного счёта по окончании основного времени игра продлевается на 2000 тиков дополнительного времени. Помимо этого, игра не может закончиться в течение 300 тиков после забитого гола. Эти 300 тиков являются периодом «вне игры», о чём подробнее читайте в следующем разделе. Таким образом, максимально возможная длительность игры составляет 8300 тиков. Если стратегии обоих участников «упали», игра завершается преждевременно.

«Упавшая» стратегия больше не может управлять хоккеистами. 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 ход хоккеиста выделяется не более 2 секунд реального времени. Но в сумме на всю игру процессу стратегии выделяется

  $$50 \times <длительность\_игры\_в\_тиках> \times <колличество\_хоккеистов\_в\_команде> + 2000 \quad (2.1)$$

  миллисекунд реального времени и

  $$15 \times <длительность\_игры\_в\_тиках> \times <колличество\_хоккеистов\_в\_команде> + 2000 \quad (2.2)$$

  миллисекунд процессорного времени. [2] В формуле учитывается только длительность основного времени — 6000 тиков. Ограничение по времени остаётся прежним, даже если реальная длительность

[2] 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.

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.

2.3 Описание юнитов и правила игры

Как уже отмечалось выше, в мире CodeHockey 2014 существует 2 типа юнитов: хоккеисты и шайба. Хоккеисты делятся на полевых хоккеистов и вратарей. Стратегия игрока осуществляет управление полевыми хоккеистами, но не вратарём. В свою очередь полевые хоккеисты также делятся на несколько подтипов (подробнее об этом в следующих разделах). Все юниты являются кругами, их точные характеристики приведены в следующей таблице:

Характеристика юнита	Хоккеист	Вратарь	Шайба
Радиус	30	30	20
Масса	30	∞	5

При столкновении объектов модуль и направление их скоростей меняются естественным образом в зависимости от масс и исходных скоростей объектов. Столкновения не являются абсолютно упругими, и объекты теряют часть скорости. Шайба и полевые хоккеисты (в отличие от вратарей) не взаимодействуют между собой как физические сущности, однако хоккеист может выполнять различные действия для управления движением шайбы.

В арсенале стратегии, управляющей хоккеистом, есть следующие возможности:

- *Придать хоккеисту ускорение.* Стратегия может установить значение `move.speedUp` в интервале от -1.0 до 1.0. Значение `move.speedUp` является не абсолютным, а относительным к максимально возможным изменениям скорости за тик. Положительные значения приводят к ускорению при движении вперёд и торможению при движении назад. Наоборот, отрицательные значения приводят к ускорению при движении назад и торможению при движении вперёд. При относительном ускорении, равном 1.0, модуль абсолютного значения примерно равен 0.116 тиков$^{-2}$, при -1.0 — около 0.069 тиков$^{-2}$. Модуль скорости полевого хоккеиста ограничен значением 15.0 тиков$^{-1}$, однако на практике оно почти недостижимо. Все юниты подвержены действию силы трения и теряют часть своей скорости каждый тик. Чем выше скорость, тем больше потеря.

- *Повернуть хоккеиста.* Стратегия может установить значение `move.turn` в интервале от $-\pi/60.0$ (вращение против часовой стрелки) до $\pi/60.0$ (вращение по часовой стрелке) радиан.

- *Поручить хоккеисту выполнить действие.* Для этого стратегия должна установить значение `move.action`. После каждого действия, за исключением `NONE`, есть задержка («cooldown»), в течение которой хоккеист не может совершать другие действия. Для большинства действий задержка составляет 60 тиков, в остальных случаях задержка указана явно в описании действия. В каждый тик хоккеист может выполнить не более одного действия. Если стратегия устанавливает значение `move.action` несколько раз за один тик, то будет учтено только последнее значение. Ниже приведён список всех возможных действий:

  - `NONE`. *Ничего не делать.*
  - `TAKE_PUCK`. *Попытаться установить контроль над шайбой.* Для этого шайба должна находиться в секторе досягаемости клюшки хоккеиста.[3] В противном случае действие игнорируется и не инициирует задержку. То же происходит и в случае, если хоккеист уже контролирует шайбу. Если шайба контролируется другим хоккеистом, то она будет перехвачена с

[3] Юнит находится в секторе досягаемости клюшки хоккеиста, если расстояние от центра хоккеиста до центра этого юнита не превышает 120.0, а угол к этому юниту относительно угла поворота хоккеиста лежит в интервале от $-\pi/12.0$ до $\pi/12.0$ радиан.

вероятностью 25%[4]Хоккеист, потерявший шайбу, не может совершать действия в течение 10 тиков. Если шайба не контролируется другим хоккеистом и находится в состоянии покоя, то базовый шанс установить над ней контроль равен 160%. Это значение равномерно уменьшается с ростом скорости шайбы, достигая (но не останавливаясь на) 60% при 20.0 тиках$^{-1}$ — скорости, придаваемой шайбе после удара по ней хоккеиста, находящегося в состоянии покоя. В случае успеха действия хоккеист становится владельцем шайбы. Это означает, что игровой симулятор в конце каждого тика устанавливает положение центра шайбы в точку перед хоккеистом на расстоянии 55.0 от его центра.

- `SWING`. *Замахнуться для удара*. Чем больше тиков пройдёт с момента начала замаха до удара, тем большее воздействие будет на попавшие под удар объекты. Максимальное количество учитываемых тиков ограничено 20. Задержка после данного действия равна 10. В состоянии замаха (`SWINGING`[5]) управление перемещением хоккеиста игнорируется, а из действий доступны только `STRIKE` и `CANCEL_STRIKE`.
- `STRIKE`. *Ударить*. Удар может быть совершён как с предварительным замахом (`SWING`), так и без него. Относительная сила удара составляет 1.0 при максимальном замахе и 0.75 без замаха. Удар воздействует одновременно на всех юнитов, находящихся в секторе досягаемости клюшки хоккеиста. Удар не является идеально точным. Для каждого юнита, попавшего под удар, сдвиг значения угла является нормальным случайным числом со стандартным отклонением 2°.

  Если хоккеист, совершающий удар, контролирует шайбу, то удар по ней проходит гарантированно. Если шайба контролируется другим хоккеистом, то она будет выбита у него с вероятностью 75%. При этом хоккеист, потерявший шайбу, не сможет совершать действия в течение 10 тиков. Если шайба не контролируется другим хоккеистом и находится в состоянии покоя, то базовый шанс ударить её равен 175%. Это значение уменьшается с ростом скорости шайбы, достигая 75% при 20.0 тиках$^{-1}$, аналогично шансу действия `TAKE_PUCK`. В случае успеха модуль скорости шайбы мгновенно становится равным

$$20.0 * StrikePower + Speed_{Striker} * cos(Angle_{Striker} - SpeedAngle_{Striker}), \tag{2.3}$$

  где $StrikePower$ — относительная сила удара, а $Speed_{Striker}$, $Angle_{Striker}$ и $SpeedAngle_{Striker}$ — соответственно модуль скорости, угол поворота и угол скорости хоккеиста, совершающего удар; направление скорости шайбы становится равным направлению удара. Другими словами, скорость хоккеиста немного влияет на модуль скорости шайбы, но не на направление.

  Вектор скорости попавшего под удар хоккеиста становится равным сумме вектора его текущей скорости и вектора, модуль которого равен

$$4.0 * StrikePower + Speed_{Striker} * cos(Angle_{Striker} - SpeedAngle_{Striker}), \tag{2.4}$$

  а направление совпадает с направлением удара. При этом есть некоторый шанс, что хоккеист будет сбит ударом с ног (`KNOCKED_DOWN`) на 40 тиков. Этот шанс прямо пропорционально зависит от относительной силы удара и при значении 1.0 составляет 50%. Сбитый с ног хоккеист не может управляться стратегией, а также теряет шайбу, если контролировал её.

  Действие `STRIKE` никак не влияет на вратарей.
- `CANCEL_STRIKE`. *Отменить удар*. Если хоккеист находится в состоянии замаха (`SWINGING`), то он возвратится в состояние по умолчанию (`ACTIVE`), а задержка следующего действия составит 30 тиков. В противном случае действие будет проигнорировано.
- `PASS`. *Отдать пас*. Фактически, пас является направленным аналогом удара (`STRIKE`), однако это действие может быть совершено только хоккеистом, контролирующим шайбу, и воздействует только на неё. Для точной настройки стратегия может установить также относительную силу паса `move.passPower` и относительный угол паса `move.passAngle`. Значение `move.passPower` должно лежать в интервале от 0.0 до 1.0, а `move.passAngle` — от $-\pi/3.0$ до $\pi/3.0$. Как и удар, пас не является идеально точным. Сдвиг значения угла является нормальным случайным числом со стандартным отклонением 1.5°.

---

[4]Для любого вероятностного события в игре действуют следующие ограничения: если шанс свершения события меньше 5%, то он считается равным 5%; если шанс больше 95%, то он считается равным 95%.

[5]Смотрите документацию к классу `HockeyistState`.

После совершения паса хоккеист теряет контроль над шайбой; модуль скорости шайбы становится равным

$$15.0 * PassPower + Speed_{Striker} * cos(Angle_{Striker} + PassAngle - SpeedAngle_{Striker}), \quad (2.5)$$

где $PassPower$ — относительная сила паса, $PassAngle$ — относительный угол паса, а $Speed_{Striker}$, $Angle_{Striker}$ и $SpeedAngle_{Striker}$ — соответственно модуль скорости, угол поворота и угол скорости хоккеиста, совершающего пас; направление скорости шайбы становится равным сумме текущего угла поворота хоккеиста и относительного угла паса.

Если хоккеист не является владельцем шайбы, то действие будет проигнорировано.

– `SUBSTITUTE`. *Выполнить замену*. Не считая вратарей, одновременно на площадке может находиться не более 3 хоккеистов из каждой команды. Если размер команды превышает это значение, то остальные хоккеисты будут сидеть на скамейке запасных. Для выполнения замены хоккеист должен удовлетворять следующим условиям: он должен находиться на своей половине поля, его скорость не должна превышать 1.0 тик$^{-1}$, а расстояние от центра хоккеиста до верхней границы игровой площадки не должно превышать 60.0. Для выполнения замены стратегия также должна указать значение `move.teammateIndex` — индекс хоккеиста, который будет выведен на площадку.

  В случае успеха два участвующих в действии хоккеиста будут поменяны местами, скорость выведенного на площадку хоккеиста будет установлена в ноль, а задержка до следующего действия составит 60 тиков. Если хоккеист контролировал шайбу, её положение не изменится, однако у неё уже не будет владельца.

  Стоит отметить, что за каждым хоккеистом закреплена некоторая начальная позиция, в которую он помещается в начале игры и после каждого сброса шайбы. При выполнении действия `SUBSTITUTE` начальные позиции двух хоккеистов также меняются. Допустим, хоккеисты А и Б находятся на площадке, а хоккеист В — на скамейке запасных. Если хоккеист А будет заменён на хоккеиста В, а затем хоккеист Б — на хоккеиста А, то хоккеисту А после его возвращения в игру будет соответствовать начальная позиция хоккеиста Б, а не его собственная начальная позиция.

Во избежание коллизий, когда, например, два хоккеиста пытаются установить контроль над свободной шайбой в один и тот же тик, используется следующая стратегия. На каждом тике все полевые хоккеисты упорядочиваются случайным образом. Затем, согласно этому порядку, симулятор игры выполняет все действия (`move.action`), порученные хоккеистам управляющими стратегиями. После этого для всех хоккеистов применяются значения `move.speedUp` и `move.turn`, и игровой симулятор обновляет положение всех игровых объектов к следующему тику.

Стратегия не может управлять вратарём, однако вратарь защищает ворота самостоятельно. Он пытается оставаться на одной горизонтальной линии с шайбой, но его скорость ограничена значением 6 тиков$^{-1}$. Вратарь также не может сместиться со своей вертикальной линии и ограничен штангами ворот сверху и снизу.

Если шайба попадает в ворота (центр шайбы оказывается внутри прямоугольника ворот), тогда игрок, владеющий этими воротами, считается пропустившим гол, а другой игрок — забившим гол. Счётчик `goalCount` забившего игрока увеличивается на единицу. Следующие 300 тиков после забитого гола считаются состоянием вне игры. Новые голы в течение этого времени игнорируются, однако стратегии не теряют управление и могут выполнять любые действия, которые сочтут нужными, например, замену одного хоккеиста на другого. По прошествии этих 300 тиков все полевые хоккеисты и шайба возвращаются на свои начальные позиции и игра продолжается. При этом скорость всех юнитов сбрасывается в ноль, а хоккеисты возвращаются в состояние `ACTIVE`. Игра не может закончиться в течение этих 300 тиков, даже если основное время истекло.

Хоккеист не может въехать внутрь ворот, а именно, если хотя бы одна точка хоккеиста находится внутри ворот, то игровой симулятор будет выталкивать его по горизонтали в сторону центра поля.

2.4 Изменение правил в Раунде 2

На этом этапе турнира в каждую команду добавляется по одному хоккеисту. Таким образом, число полевых хоккеистов, находящихся под контролем стратегии каждого игрока, будет равно 3. Помимо этого в Раунде 2 вводится понятие атрибутов хоккеистов. Атрибуты — это базовые характеристики, влияющие на эффективность практически любого действия, совершаемого хоккеистом. В зависимости от доминирования того или иного атрибута, хоккеист может совершать некоторые действия с опорой как на силовую, так и на техническую составляющую. Всего в мире CodeHockey 2014 есть 4 различных атрибута, образующих условный квадрат:

- `STRENGTH`. *Сила*. Влияет на силу удара и вероятность проведения силовых приёмов в отношении других хоккеистов. Таким образом, формула расчёта модуля скорости попавшей под удар шайбы приобретает вид

  $$20.0 * StrikePower * \frac{Strength_{Striker}}{100} + Speed_{Striker} * cos(Angle_{Striker} - SpeedAngle_{Striker}), \quad (2.6)$$

  а формула расчёта модуля добавочной скорости попавшего под удар хоккеиста —

  $$4.0 * StrikePower * \frac{Strength_{Striker}}{100} + Speed_{Striker} * cos(Angle_{Striker} - SpeedAngle_{Striker}). \quad (2.7)$$

- `ENDURANCE`. *Стойкость*. Влияет на вероятность проведения силовых приёмов другими хоккеистами в отношении этого хоккеиста. Как правило, противопоставляется атрибуту сила другого хоккеиста.

- `DEXTERITY`. *Ловкость*. Влияет на точность удара/паса и вероятность проведения технических приёмов в отношении других хоккеистов. Зная значение атрибута ловкость, стандартное отклонение сдвига угла удара можно получить по формуле $2^\circ * \frac{100}{Dexterity_{Striker}}$, а паса — $1.5^\circ * \frac{100}{Dexterity_{Striker}}$.

- `AGILITY`. *Подвижность*. Влияет на скорость движения и поворота хоккеиста, а также вероятность проведения технических приёмов другими хоккеистами в отношении этого хоккеиста. Как правило, противопоставляется атрибуту ловкость другого хоккеиста. К значению `move.speedUp`, установленному стратегией, применяется поправочный коэффициент $\frac{Agility}{100}$, а ограничениями значения `move.turn` теперь являются $-\pi/60.0 * \frac{Agility}{100}$ и $\pi/60.0 * \frac{Agility}{100}$.

Теперь рассмотрим подробнее влияние атрибутов на различные вероятностные действия:

- `TAKE_PUCK`. Вероятность перехвата шайбы, контролируемой другим хоккеистом, составляет

  $$25\% + \max(Strength_{Attacker}, Dexterity_{Attacker})\% - \max(Endurance_{Target}, Agility_{Target})\%, \quad (2.8)$$

  где $Attacker$ — хоккеист, совершающий действие, а $Target$ — хоккеист, контролирующий шайбу на данный момент. Оба хоккеиста, в зависимости от своих способностей, могут использовать как силовой способ, так и какой-либо технический приём для отбора/защиты шайбы. Вероятность установления контроля над свободной шайбой составляет

  $$60\% + \max(Dexterity_{Hockeyist}, Agility_{Hockeyist})\% - \frac{Speed_{Puck}}{20} * 100\%. \quad (2.9)$$

- `STRIKE`. Вероятность выбить шайбу, контролируемую другим хоккеистом, составляет

  $$75\% + \max(Strength_{Attacker}, Dexterity_{Attacker})\% - \max(Endurance_{Target}, Agility_{Target})\%, \quad (2.10)$$

  а вероятность успешного удара по свободной шайбе определяется формулой

  $$75\% + \max(Dexterity_{Hockeyist}, Agility_{Hockeyist})\% - \frac{Speed_{Puck}}{20} * 100\%. \quad (2.11)$$

Как уже указывалось, если хоккеист, совершающий удар, контролирует шайбу, то удар по ней проходит гарантированно.

Вероятность, что хоккеист, попавший под удар, будет сбит с ног, равна

$$50 * StrikePower\% + \max\left(Strength_{Attacker}, Dexterity_{Attacker}\right)\% - Endurance_{Target}\%, \quad (2.12)$$

а время восстановления сбитого хоккеиста составит $40 * \frac{100}{Agility}$ тиков.

Базовое значение атрибута хоккеиста равно 100, минимально допустимым значением является 80, а максимально допустимым — 120. Значения всех атрибутов хоккеистов из Раунда 1 равны 100. Несложно заметить, что изменения Раунда 2 никак на них не влияют, а формулы сводятся к своим более простым аналогам из предыдущего раздела. Таким образом, стратегии участников, адаптированные под новые правила, являются также обратно совместимыми и могут легко участвовать в играх старого формата.

В Раунде 2 в каждой команде присутствует по одному полевому хоккеисту следующих 3 типов:

- `VERSATILE`. *Хоккеист-универсал.* Аналогичен хоккеистам из Раунда 1.
- `FORWARD`. *Нападающий*. Его основным достоинством является сила удара. Стойкость является слабой стороной.
- `DEFENCEMAN`. *Защитник*. Его основным достоинством является стойкость. Ловкость является слабой стороной.

В следующей таблице приведены значения атрибутов для указанных типов хоккеистов:

Атрибут	Универсал	Нападающий	Защитник
Сила	100	110	105
Стойкость	100	80	110
Ловкость	100	105	80
Подвижность	100	105	105

2.5 Изменение правил в Финале

В Финале каждая команда будет состоять из 6 полевых хоккеистов. В каждый тик, трое из них находятся на площадке, а остальные отдыхают на скамейке запасных. Все эти хоккеисты принадлежат к типу `RANDOM`. Это означает, что каждый их атрибут является случайным числом, лежащим в интервале от 80 до 120. Симулятор устанавливает атрибуты в начале игры симметрично для обеих команд.

На этом этапе турнира вводится понятие выносливости («stamina») хоккеиста. Выносливость влияет на все действия хоккеиста опосредованно через атрибуты. Формально, для расчёта любого действия используется не исходное значение атрибута, а его «эффективное» значение, определяемое по формуле

$$0.75 * Attribute + 0.25 * Attribute * \frac{Stamina}{2000.0}, \quad (2.13)$$

где $Attribute$ — исходное значение атрибута хоккеиста, а $Stamina$ — текущее значение выносливости. Начальное и максимальное значение выносливости хоккеиста равно 2000.0. Значение выносливости не может упасть ниже нуля. Таким образом, эффективное значение атрибута равно исходному при максимальном значении выносливости и падает до 75%, если выносливость полностью истрачена.

Выносливость тратится при ускорении/замедлении хоккеиста на модуль значения `move.speedUp`. Выносливость тратится при повороте хоккеиста на абсолютное значение отношения реального угла поворота к максимально возможному углу поворота за один тик для данного хоккеиста.

Затраты выносливости на различные действия:

- `TAKE_PUCK`. Затрачивает 10.0 единиц выносливости.
- `SWING`. Затрачивает 10.0 единиц выносливости.
- `STRIKE`. Удар без замаха затрачивает 20.0 единиц выносливости. Удар с замахом, помимо затрат на замах, расходует базово 20.0 единиц выносливости и дополнительно 0.5 единиц выносливости за каждый тик замаха, но не более 10.0.
- `PASS`. Затрачивает 40.0 единиц выносливости.

Выносливость не тратится в состоянии вне игры (300 тиков после гола).

Каждый игровой тик выносливость восполняется сама собой на 0.5 для каждого хоккеиста, находящегося в состоянии по умолчанию (`ACTIVE`), и на 1.0 для каждого отдыхающего (`RESTING`) хоккеиста.

Выносливость всех хоккеистов в играх формата Раунда 1 и Раунда 2 всегда равна своему максимальному значению.

Глава 3

Создание стратегии

3.1 Техническая часть

Сперва для создания стратегии вам необходимо выбрать один из ряда поддерживаемых языков программирования[6]: Java (Oracle JDK 7), C# (Mono 2), C++ (GNU MinGW C++ 4), Python 2 (Python 2.7+), Python 3 (Python 3.4+), Pascal (Free Pascal 2), Ruby (JRuby 1.7+, Oracle JDK 7). 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)[7]. 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 окно.

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, создать несколько коротких (по 200 тиков) игр разных форматов: 2 на 2 (2×2), 3 на 3 (2×3) и 6 на 6 (2×6). Длительность дополнительного времени в этих играх также составит 200 тиков. Для управления каждой командой 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>`» и «`<имя_игрока> (2)`».

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 более подробной информации). Также вам становится доступна функция создания собственных игр, в которых в качестве соперника можно выбирать любую стратегию любого игрока (в том числе и вашу собственную), созданную до момента вашей последнего 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.

---

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

[7] 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.

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 боях в Песочнице и боях в этапах турнира, также невозможно собственноручно создавать бои с её участием. Однако после принятия данной посылки система автоматически добавляет тестовую игру с двумя участниками (формат 2 на 2): непосредственно тестовой посылкой и стратегией из раздела «Быстрый старт». Тестовая игра видна только участнику, сделавшему данную тестовую посылку. Длительность основного времени 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 бои. Для этого нужно нажать кнопку «Смотреть» в списке боёв либо нажать кнопку «Посмотреть игру» на странице боя.

Если вы смотрите бой 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 данного боя. 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 CodeHockey 2014 (раздел «Документация» → «Утилита Repeater») и разархивируйте. Для запуска Repeater вам необходимо установленное ПО Java 7 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. 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. Длительность основного времени подобных локальных игр составляет стандартные 6000 тиков. 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 (для Windows или *n*x систем), затем запустите свою стратегию из среды разработки (или любым другим удобным вам способом) и смотрите бой. 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 10 минут. По прошествии этого времени он посчитает стратегию «упавшей» и продолжит работу без неё.

3.2 Управление хоккеистом

Для каждого полевого хоккеиста в вашей команде в начале игры создаётся отдельный экземпляр класса `MyStrategy`, в полях которого стратегия может хранить информацию о данном хоккеисте. Общую для всей команды информацию, в зависимости от языка программирования, можно хранить в статических полях или глобальных переменных.

Управление хоккеистом осуществляется с помощью метода `move` стратегии, который вызывается один раз за тик для каждого хоккеиста. Методу передаются следующие параметры:

- хоккеист `self`, для которого вызывается метод;
- текущее состояние мира `world`;
- набор игровых констант `game`;

- объект `move`, устанавливая свойства которого, стратегия и определяет поведение хоккеиста.

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 придают хоккеисту ускорение назад, одновременно с этим поворачивая его направо и выполняя действие «удар». Полную документацию классов и методов для языка Java можно найти в следующих главах.

3.3.1 Пример для Java

```
import model.*;

import static java.lang.StrictMath.PI;

public final class MyStrategy implements Strategy {
    @Override
    public void move(Hockeyist self, World world, Game game, Move move) {
        move.setSpeedUp(-1.0D);
        move.setTurn(PI);
        move.setAction(ActionType.STRIKE);
    }
}
```

3.3.2 Пример для C#

```
using System;
using Com.CodeGame.CodeHockey2014.DevKit.CSharpCgdk.Model;

namespace Com.CodeGame.CodeHockey2014.DevKit.CSharpCgdk {
    public sealed class MyStrategy : IStrategy {
        public void Move(Hockeyist self, World world, Game game, Move move) {
            move.SpeedUp = -1.0D;
            move.Turn = Math.PI;
            move.Action = ActionType.Strike;
        }
    }
}
```

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 Hockeyist& self, const World& world, const Game& game, Move& move) {
    move.setSpeedUp(-1.0);
    move.setTurn(PI);
    move.setAction(STRIKE);
}

MyStrategy::MyStrategy() { }
```

3.3.4 Пример для Python 2

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

```
from math import *
from model.ActionType import ActionType


class MyStrategy:
    def move(self, me, world, game, move):
        move.speed_up = -1.0
        move.turn = pi
        move.action = ActionType.STRIKE
```

3.3.5 Пример для Python 3

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

```
from math import *
from model.ActionType import ActionType
from model.Game import Game
from model.Move import Move
from model.Hockeyist import Hockeyist
from model.World import World


class MyStrategy:
    def move(self, me: Hockeyist, world: World, game: Game, move: Move):
        move.speed_up = -1.0
        move.turn = pi
        move.action = ActionType.STRIKE
```

3.3.6 Пример для Pascal

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

```
unit MyStrategy;

interface

uses
    StrategyControl, HockeyistControl, WorldControl, GameControl, MoveControl,
    HockeyistTypeControl, HockeyistStateControl, ActionTypeControl;

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

    end;

implementation

uses
    Math;

procedure TMyStrategy.Move(me: THockeyist; world: TWorld; game: TGame; move: TMove);
begin
    move.SetSpeedUp(-1.0);
    move.SetTurn(PI);
    move.SetAction(STRIKE);
end;

end.
```

3.3.7 Пример для Ruby

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

```
require './model/action_type'
require './model/game'
require './model/move'
require './model/hockeyist'
require './model/world'

class MyStrategy
  # @param [Hockeyist] me
  # @param [World] world
  # @param [Game] game
  # @param [Move] move
  def move(me, world, game, move)
    move.speed_up = -1.0
    move.turn = Math::PI
    move.action = ActionType::STRIKE
  end
end
```

Глава 4

Package model

Package Contents *Page*

Classes

4.1 Classes

4.1.1 CLASS **ActionType**

Возможные действия хоккеиста.

Хоккеист может совершить действие, если он не сбит с ног (`HockeyistState.KNOCKED_DOWN`), не отдыхает (`HockeyistState.RESTING`) и уже восстановился после своего предыдущего действия (значение `hockeyist.remainingCooldownTicks` равно `0`).

Если хоккеист замахивается клюшкой (`HockeyistState.SWINGING`), то из действий ему доступны только `ActionType.STRIKE` и `ActionType.CANCEL_STRIKE`.

DECLARATION

public final class ActionType

extends Enum

FIELDS

- public static final ActionType NONE
  - Ничего не делать.
- public static final ActionType TAKE_PUCK
  - Взять шайбу.
    Если хоккеист уже контролирует шайбу, либо шайба находится вне зоны досягаемости клюшки хоккеиста (смотрите документацию к значениям `game.stickLength` и `game.stickSector`), то действие игнорируется.
    В противном случае хоккеист попытается установить контроль над шайбой и, с определённой вероятностью, это сделает ((смотрите документацию к `game.pickUpPuckBaseChance` и `game.takePuckAwayBaseChance`)).
- public static final ActionType SWING
  - Замахнуться для удара.
    Хоккеист замахивается для увеличения силы удара. Чем больше тиков пройдёт с момента начала замаха до удара, тем большее воздействие будет на попавшие под удар объекты. Максимальное количество учитываемых тиков ограничено значением `game.maxEffectiveSwingTicks`.
- public static final ActionType STRIKE
  - Ударить.
    Хоккеист наносит размашистый удар по всем объектам, находящимся в зоне досягаемости его клюшки. Удар может быть совершён как с предварительным замахом (`SWING`), так и без него (в этом случае сила удара будет меньше).
    Объекты (шайба и хоккеисты, кроме вратарей), попавшие под удар, приобретут некоторый импульс в направлении, совпадающим с направлением удара. При ударе по хоккеисту есть также некоторый шанс сбить его с ног.

- public static final ActionType CANCEL_STRIKE
  - Отменить удар.

    Хоккеист выходит из состояния замаха (`SWING`), не совершая удар. Это позволяет сэкономить немного выносливости, а также быстрее совершить новое действие (смотрите документацию к `game.cancelStrikeActionCooldownTicks`).

    Если хоккеист не совершает замах клюшкой, то действие игнорируется.
- public static final ActionType PASS
  - Отдать пас.

    Хоккеист пытается передать контролируемую им шайбу другому хоккеисту. Для этого необходимо указать относительную силу паса (`move.passPower`) и его направление (`move.passAngle`). В противном случае пас будет отдан в направлении, соответствующем направлению хоккеиста, с максимально возможной силой.

    Если хоккеист не контролирует шайбу, то действие игнорируется.
- public static final ActionType SUBSTITUTE
  - Заменить активного хоккеиста сидящим на скамейке запасных.

    Замена выполняется только на своей половине поля, при этом расстояние от центра хоккеиста до верхней границы игровой площадки не должно превышать `game.substitutionAreaHeight`. Дополнительно нужно указать индекс хоккеиста (`move.teammateIndex`), на которого будет произведена замена.

    Если указан некорректный индекс, или скорость хоккеиста превышает `game.maxSpeedToAllowSubstitute`, то действие будет проигнорировано.

METHODS

- *valueOf*
  `public static ActionType valueOf( String  name )`
- *values*
  `public static ActionType[] values( )`

4.1.2 CLASS **Game**

Предоставляет доступ к различным игровым константам.

DECLARATION

---

public class Game **extends** Object

- *getActionCooldownTicksAfterLosingPuck*
`public int` **getActionCooldownTicksAfterLosingPuck**`( )`

  - **Returns** - Возвращает длительность задержки, применяемой к хоккеисту в случае потери шайбы вследствие воздействия других хоккеистов. В течение этого времени хоккеист не может совершать действия.

- *getActiveHockeyistStaminaGrowthPerTick*
`public double` **getActiveHockeyistStaminaGrowthPerTick**`( )`

  - **Returns** - Возвращает значение, на которое увеличивается выносливость хоккеиста за каждый тик в состоянии `HockeyistType.ACTIVE`.

- *getAfterGoalStateTickCount*
`public int` **getAfterGoalStateTickCount**`( )`

  - **Returns** - Возвращает длительность состояния вне игры после гола. В течение этого времени новые забитые голы игнорируются, а действия не требуют затрат выносливости.

- *getCancelStrikeActionCooldownTicks*
`public int` **getCancelStrikeActionCooldownTicks**`( )`

  - **Returns** - Возвращает длительность задержки, применяемой к хоккеисту после отмены им удара (`ActionType.CANCEL_STRIKE`). В течение этого времени хоккеист не может совершать новые действия.

- *getCancelStrikeStaminaCost*
`public double` **getCancelStrikeStaminaCost**`( )`

  - **Returns** - Возвращает количество выносливости, которое необходимо затратить на совершение действия `ActionType.CANCEL_STRIKE`.

- *getDefaultActionCooldownTicks*
`public int` **getDefaultActionCooldownTicks**`( )`

  - **Returns** - Возвращает длительность задержки, применяемой к хоккеисту после совершения им большинства действий (`move.action`). В течение этого времени хоккеист не может совершать новые действия.

- *getDefencemanHockeyistAgility*
`public int` **getDefencemanHockeyistAgility**`( )`

  - **Returns** - Возвращает значение атрибута подвижность для защитника.

- *getDefencemanHockeyistDexterity*
`public int` **getDefencemanHockeyistDexterity**`( )`

  - **Returns** - Возвращает значение атрибута ловкость для защитника.

- *getDefencemanHockeyistEndurance*
`public int` **getDefencemanHockeyistEndurance**`( )`

  - **Returns** - Возвращает значение атрибута стойкость для защитника.

- *getDefencemanHockeyistStrength*
`public int` **getDefencemanHockeyistStrength**`( )`

  - **Returns** - Возвращает значение атрибута сила для защитника.

- *getForwardHockeyistAgility*
`public int` **getForwardHockeyistAgility**`( )`

  - **Returns** - Возвращает значение атрибута подвижность для нападающего.

- *getForwardHockeyistDexterity*
`public int` **getForwardHockeyistDexterity**`( )`

  - **Returns** - Возвращает значение атрибута ловкость для нападающего.

- *getForwardHockeyistEndurance*
`public int` **getForwardHockeyistEndurance**`( )`

  - **Returns** - Возвращает значение атрибута стойкость для нападающего.

- *getForwardHockeyistStrength*
`public int` **getForwardHockeyistStrength**`( )`

  - **Returns** - Возвращает значение атрибута сила для нападающего.

- *getGoalieMaxSpeed*
`public double` **getGoalieMaxSpeed**`( )`

  - **Returns** - Возвращает максимальную скорость перемещения вратаря.

- *getGoalNetHeight*
`public double` **getGoalNetHeight**`( )`

  - **Returns** - Возвращает высоту ворот.

- *getGoalNetTop*
`public double` **getGoalNetTop**`( )`

  - **Returns** - Возвращает ординату верхней штанги ворот.

- *getGoalNetWidth*
`public double` **getGoalNetWidth**`( )`

  - **Returns** - Возвращает ширину ворот.

- *getHockeyistAttributeBaseValue*
`public int` **getHockeyistAttributeBaseValue**`( )`

  - **Returns** - Возвращает базовое значение атрибута хоккеиста. Данная величина используется как коэффициент в различных игровых формулах.

- *getHockeyistMaxSpeed*
`public double` **getHockeyistMaxSpeed**`( )`

  - **Returns** - Возвращает максимальную скорость перемещения полевого хоккеиста.

- *getHockeyistMaxStamina*
`public double` **getHockeyistMaxStamina**`( )`

  - **Returns** - Возвращает максимальное значение выносливости хоккеиста. Выносливость тратится на перемещение и совершение хоккеистом различных действий. Каждый тик может восстановиться небольшое количество выносливости в зависимости от состояния хоккеиста (`hockeyist.state`). По мере расходования выносливости все атрибуты (соответственно, и эффективность всех действий) хоккеиста равномерно уменьшаются и достигают значения `zeroStaminaHockeyistEffectivenessFactor` (от начальных показателей) при падении выносливости до нуля. Хоккеист не восстанавливает выносливость в состояниях `HockeyistState.SWINGING` и `HockeyistState.KNOCKED_DOWN`.

- *getHockeyistSpeedDownFactor*
`public double` **getHockeyistSpeedDownFactor**`( )`

  - **Returns** - Возвращает модуль ускорения, приобретаемого хоккеистом, при `move.speedUp` равном -1.0, базовом значении атрибута подвижность и максимальном запасе выносливости. Направление ускорения противоположно направлению хоккеиста. В игре отсутствует специальное ограничение на максимальную скорость хоккеиста, однако все игровые объекты подвержены воздействию силы трения, которая уменьшает модуль их скорости каждый тик. Чем больше скорость, тем на большую величину она уменьшается.

- *getHockeyistSpeedUpFactor*
`public double` **getHockeyistSpeedUpFactor**`( )`

  - **Returns** - Возвращает модуль ускорения, приобретаемого хоккеистом, при `move.speedUp` равном 1.0, базовом значении атрибута подвижность и максимальном запасе выносливости. Направление ускорения совпадает с направлением хоккеиста. В игре отсутствует специальное ограничение на максимальную скорость хоккеиста, однако все игровые объекты подвержены воздействию силы трения, которая уменьшает модуль их скорости каждый тик. Чем больше скорость, тем на большую величину она уменьшается.

- *getHockeyistTurnAngleFactor*
`public double` **getHockeyistTurnAngleFactor**`( )`

  - **Returns** - Возвращает максимальный модуль угла поворота хоккеиста за тик при базовом значении атрибута подвижность и максимальном запасе выносливости.

- *getKnockdownChanceFactor*
`public double` **getKnockdownChanceFactor**`( )`

  - **Returns** - Возвращает шанс ударом (`ActionType.STRIKE`) сбить с ног другого хоккеиста при максимальной длительности замаха. Среднее значение атрибутов сила и ловкость хоккеиста, совершающего удар, увеличивает шанс сбить с ног. Значение атрибута стойкость атакуемого хоккеиста уменьшает шанс на падение.

- *getKnockdownTicksFactor*
`public double` **getKnockdownTicksFactor**`( )`

  - **Returns** - Возвращает количество тиков, по прошествии которого хоккеист восстановится после падения при базовом значении атрибута подвижность. Чем выше подвижность, тем быстрее восстановление.

- *getMaxActionChance*
`public double` **getMaxActionChance**`( )`

  - **Returns** - Возвращает максимальный шанс на совершение любого вероятностного действия.

- *getMaxEffectiveSwingTicks*
`public int` **getMaxEffectiveSwingTicks**`( )`

  - **Returns** - Возвращает длительность замаха, после достижения которой сила удара не увеличивается.

- *getMaxRandomHockeyistParameter*
`public int` **getMaxRandomHockeyistParameter**`( )`

  - **Returns** - Возвращает максимально возможное значение любого атрибута для хоккеиста со случайными параметрами.

- *getMaxSpeedToAllowSubstitute*
`public double` **getMaxSpeedToAllowSubstitute**`( )`

- **Returns** - Возвращает максимальную допустимую скорость для выполнения замены хоккеиста.

- *getMinActionChance*
`public double` **getMinActionChance**`( )`

  - **Returns** - Возвращает минимальный шанс на совершение любого вероятностного действия.

- *getMinRandomHockeyistParameter*
`public int` **getMinRandomHockeyistParameter**`( )`

  - **Returns** - Возвращает минимально возможное значение любого атрибута для хоккеиста со случайными параметрами.

- *getOvertimeTickCount*
`public int` **getOvertimeTickCount**`( )`

  - **Returns** - Возвращает длительность дополнительного времени. Дополнительное время наступает в случае ничейного счёта на момент окончания основного времени. Если за основное время не было забито ни одного гола, вратари обоих игроков убираются с поля.

- *getPassAngleDeviation*
`public double` **getPassAngleDeviation**`( )`

  - **Returns** - Возвращает стандартное отклонение распределения Гаусса для угла паса (`ActionType.PASS`) хоккеиста при базовом значении атрибута ловкость. Чем выше ловкость конкретного хоккеиста, тем точнее его пас.

- *getPassPowerFactor*
`public double` **getPassPowerFactor**`( )`

  - **Returns** - Возвращает коэффициент силы паса. Умножается на устанавливаемое стратегией в интервале [`0.0`, `1.0`] значение силы паса (`move.passPower`).

- *getPassSector*
`public double` **getPassSector**`( )`

  - **Returns** - Возвращает сектор, ограничивающий направление паса.

- *getPassStaminaCost*
`public double` **getPassStaminaCost**`( )`

  - **Returns** - Возвращает количество выносливости, которое необходимо затратить на совершение действия `ActionType.PASS`.

- *getPickUpPuckBaseChance*
`public double` **getPickUpPuckBaseChance**`( )`

  - **Returns** - Возвращает шанс подобрать шайбу, не контролируемую другим хоккеистом, без учёта влияния атрибутов хоккеиста и скорости шайбы. Равен шансу подобрать шайбу в случае, когда влияющие на действие атрибуты хоккеиста равны `hockeyistAttributeBaseValue`, а шайба двигается со скоростью `struckPuckInitialSpeedFactor`.
  Максимальный из атрибутов ловкость и подвижность хоккеиста увеличивает шанс на захват. Скорость шайбы уменьшает шанс на захват.

- *getPuckBindingRange*
`public double` **getPuckBindingRange**`( )`

  - **Returns** - Возвращает расстояние от центра хоккеиста, контролирующего шайбу, до центра шайбы.

- *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.

- *getRestingHockeyistStaminaGrowthPerTick*
`public double` **getRestingHockeyistStaminaGrowthPerTick**`( )`

  - **Returns** - Возвращает значение, на которое увеличивается выносливость хоккеиста за каждый тик в состоянии `HockeyistType.RESTING`.

- *getRinkBottom*
`public double` **getRinkBottom**`( )`

  - **Returns** - Возвращает ординату нижней границы игрового поля.

- *getRinkLeft*
`public double` **getRinkLeft**`( )`

  - **Returns** - Возвращает абсциссу левой границы игрового поля.

- *getRinkRight*
`public double` **getRinkRight**`( )`

  - **Returns** - Возвращает абсциссу правой границы игрового поля.

- *getRinkTop*
`public double` **getRinkTop**`( )`

  - **Returns** - Возвращает ординату верхней границы игрового поля.

- *getSpeedUpStaminaCostFactor*
`public double` **getSpeedUpStaminaCostFactor**`( )`

  - **Returns** - Возвращает количество выносливости, которое необходимо затратить на максимальное по модулю ускорение/замедление хоккеиста (`move.speedUp`) за 1 тик. Для меньших значений ускорения затраты выносливости пропорционально падают.

- *getStickLength*
`public double` **getStickLength**`( )`

  - **Returns** - Возвращает длину клюшки хоккеиста. Хоккеист может воздействовать на игровой объект, если и только если расстояние от центра хоккеиста до центра объекта не превышает эту величину.

- *getStickSector*
`public double` **getStickSector**`( )`

  - **Returns** - Возвращает сектор клюшки хоккеиста. Хоккеист может воздействовать на игровой объект, если и только если угол между направлением хоккеиста и вектором из центра хоккеиста в центр объекта не превышает половину этой величины.

- *getStrikeAngleDeviation*
`public double` **getStrikeAngleDeviation**`( )`

  - **Returns** - Возвращает стандартное отклонение распределения Гаусса для угла удара (`ActionType.STRIKE`) хоккеиста при базовом значении атрибута ловкость. Чем выше ловкость конкретного хоккеиста, тем точнее его удар.

- *getStrikePowerBaseFactor*
`public double` **getStrikePowerBaseFactor**`( )`

  - **Returns** - Возвращает коэффициент силы удара без замаха.

- *getStrikePowerGrowthFactor*
`public double` **getStrikePowerGrowthFactor**`( )`

  - **Returns** - Возвращает увеличение коэффициента силы удара за каждый тик замаха. Максимальное количество учитываемых тиков ограничено значением `maxEffectiveSwingTicks`.

- *getStrikePuckBaseChance*
`public double` **getStrikePuckBaseChance**`( )`

  - **Returns** - Возвращает базовый шанс ударить шайбу. Базовый шанс не зависит от того, контролирует шайбу другой хоккеист или нет, однако на результирующий шанс удара по свободной и контролируемой шайбе влияют разные атрибуты хоккеиста (смотрите документацию к `pickUpPuckBaseChance` и `takePuckAwayBaseChance`). Если хоккеист, совершающий удар, контролирует шайбу, то вероятность удара всегда будет 100%.

- *getStrikeStaminaBaseCost*
`public double` **getStrikeStaminaBaseCost**`( )`

  - **Returns** - Возвращает базовое количество выносливости, которое необходимо затратить на совершение действия `ActionType.STRIKE`.

- *getStrikeStaminaCostGrowthFactor*
`public double` **getStrikeStaminaCostGrowthFactor**`( )`

  - **Returns** - Возвращает увеличение затрат выносливости на удар (`ActionType.STRIKE`) за каждый тик замаха. Максимальное количество учитываемых тиков ограничено значением `maxEffectiveSwingTicks`.

- *getStruckHockeyistInitialSpeedFactor*
`public double` **getStruckHockeyistInitialSpeedFactor**`( )`

  - **Returns** - Возвращает модуль скорости, добавляемой хоккеисту, попавшему под удар силы 1.0.

- *getStruckPuckInitialSpeedFactor*
`public double` **getStruckPuckInitialSpeedFactor**`( )`

  - **Returns** - Возвращает модуль скорости, устанавливаемой шайбе, попавшей под удар силы 1.0.

- *getSubstitutionAreaHeight*
`public double` **getSubstitutionAreaHeight**`( )`

  - **Returns** - Возвращает высоту зоны, в которой может быть выполнена замена хоккеиста. Зона расположена вдоль верхней границы игровой площадки. Замена может быть выполнена только на своей половине поля.

- *getSwingActionCooldownTicks*
`public int` **getSwingActionCooldownTicks**`( )`

  - **Returns** - Возвращает длительность задержки, применяемой к хоккеисту после совершения им действия замах (`ActionType.SWING`). В течение этого времени хоккеист не может совершать новые действия.

- *getSwingStaminaCost*
`public double` **getSwingStaminaCost**`( )`

  - **Returns** - Возвращает количество выносливости, которое необходимо затратить на совершение действия `ActionType.SWING`.

- *getTakePuckAwayBaseChance*
`public double` **getTakePuckAwayBaseChance**`( )`

- **Returns** - Возвращает базовый шанс отнять шайбу у другого хоккеиста.
  Максимальный из атрибутов сила и ловкость хоккеиста, отнимающего шайбу, увеличивает шанс на захват. Максимальный из атрибутов стойкость и подвижность текущего владельца шайбы уменьшает шанс на её потерю.

- *getTakePuckStaminaCost*
  `public double` **getTakePuckStaminaCost**`( )`

  - **Returns** - Возвращает количество выносливости, которое необходимо затратить на совершение действия `ActionType.TAKE_PUCK`.

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

  - **Returns** - Возвращает длительность игры в тиках.

- *getTurnStaminaCostFactor*
  `public double` **getTurnStaminaCostFactor**`( )`

  - **Returns** - Возвращает количество выносливости, которое необходимо затратить на максимальный по модулю угол поворота хоккеиста (`move.turn`) за 1 тик. Для меньших значений угла поворота затраты выносливости пропорционально падают.

- *getVersatileHockeyistAgility*
  `public int` **getVersatileHockeyistAgility**`( )`

  - **Returns** - Возвращает значение атрибута подвижность для хоккеиста-универсала.

- *getVersatileHockeyistDexterity*
  `public int` **getVersatileHockeyistDexterity**`( )`

  - **Returns** - Возвращает значение атрибута ловкость для хоккеиста-универсала.

- *getVersatileHockeyistEndurance*
  `public int` **getVersatileHockeyistEndurance**`( )`

  - **Returns** - Возвращает значение атрибута стойкость для хоккеиста-универсала.

- *getVersatileHockeyistStrength*
  `public int` **getVersatileHockeyistStrength**`( )`

  - **Returns** - Возвращает значение атрибута сила для хоккеиста-универсала.

- *getWorldHeight*
  `public double` **getWorldHeight**`( )`

  - **Returns** - Возвращает высоту игрового мира.

- *getWorldWidth*
  `public double` **getWorldWidth**`( )`

  - **Returns** - Возвращает ширину игрового мира.

- *getZeroStaminaHockeyistEffectivenessFactor*
  `public double` **getZeroStaminaHockeyistEffectivenessFactor**`( )`

  - **Returns** - Возвращает коэффициент эффективности действий хоккеиста при падении его выносливости до нуля.

4.1.3 CLASS **Hockeyist**

Класс, определяющий хоккеиста. Содержит также все свойства юнита.

DECLARATION

public class Hockeyist

extends Unit

METHODS

- *getAgility*
`public int` **getAgility**`( )`
  - **Returns** - Возвращает значение атрибута подвижность.

- *getDexterity*
`public int` **getDexterity**`( )`
  - **Returns** - Возвращает значение атрибута ловкость.

- *getEndurance*
`public int` **getEndurance**`( )`
  - **Returns** - Возвращает значение атрибута стойкость.

- *getLastAction*
`public ActionType` **getLastAction**`( )`
  - **Returns** - Возвращает последнее действие (`move.action`), совершённое хоккеистом, или `null` (`UNKNOWN_ACTION` в пакетах некоторых языков) в случае, если хоккеист ещё не совершил ни одного действия.

- *getLastActionTick*
`public Integer` **getLastActionTick**`( )`
  - **Returns** - Возвращает номер тика, в который хоккеист совершил своё последние действие (`move.action`), или `null` (`-1` в пакетах некоторых языков) в случае, если хоккеист ещё не совершил ни одного действия.

- *getOriginalPositionIndex*
`public int` **getOriginalPositionIndex**`( )`
  - **Returns** - Возвращает индекс исходной позиции хоккеиста или `-1` для вратаря или хоккеиста, отдыхающего за пределами игрового поля. На эту позицию хоккеист будет помещён при разыгрывании шайбы. При выполнении действия замена `ActionType.SUBSTITUTE` индексы исходных позиций хоккеистов, участвующих в замене, меняются местами.

- *getPlayerId*
`public long` **getPlayerId**`( )`

- **Returns** - Возвращает идентификатор игрока, в команду которого входит хоккеист.

- *getRemainingCooldownTicks*
`public int` **getRemainingCooldownTicks**`( )`

  - **Returns** - Возвращает количество тиков, по прошествии которого хоккеист сможет совершить какое-либо действие (`move.action`), или `0`, если хоккеист может совершить действие в данный тик.

- *getRemainingKnockdownTicks*
`public int` **getRemainingKnockdownTicks**`( )`

  - **Returns** - Возвращает количество тиков, по прошествии которого хоккеист восстановится после падения, или `0`, если хоккеист не сбит с ног.

- *getStamina*
`public double` **getStamina**`( )`

  - **Returns** - Возвращает текущее значение выносливости.

- *getState*
`public HockeyistState` **getState**`( )`

  - **Returns** - Возвращает состояние хоккеиста.

- *getStrength*
`public int` **getStrength**`( )`

  - **Returns** - Возвращает значение атрибута сила.

- *getSwingTicks*
`public int` **getSwingTicks**`( )`

  - **Returns** - Для хоккеиста, находящегося в состоянии замаха (`HockeyistState.SWINGING`), возвращает количество тиков, прошедших от начала замаха. В противном случае возвращает `0`.

- *getTeammateIndex*
`public int` **getTeammateIndex**`( )`

  - **Returns** - Возвращает 0-индексированный номер хоккеиста в команде.

- *getType*
`public HockeyistType` **getType**`( )`

  - **Returns** - Возвращает тип хоккеиста.

- *isTeammate*
`public boolean` **isTeammate**`( )`

  - **Returns** - Возвращает `true`, если и только если данный хоккеист входит в команду вашего игрока.

4.1.4 CLASS **HockeyistState**

Состояние хоккеиста.

DECLARATION

public final class HockeyistState

extends Enum

FIELDS

- public static final HockeyistState ACTIVE
  - Хоккеист находится на игровом поле.
- public static final HockeyistState SWINGING
  - Хоккеист находится на игровом поле и делает замах клюшкой.
    Во время замаха стратегия не может управлять движением хоккеиста, а из действий доступны только `ActionType.STRIKE` и `ActionType.CANCEL_STRIKE`.
- public static final HockeyistState KNOCKED_DOWN
  - Хоккеист находится на игровом поле, но сбит с ног. Стратегия игрока не может им управлять.
- public static final HockeyistState RESTING
  - Хоккеист отдыхает вне игрового поля. Стратегия игрока не может им управлять.

METHODS

- *valueOf*
  `public static HockeyistState` **valueOf**`( String` **name** `)`
- *values*
  `public static HockeyistState[]` **values**`( )`

4.1.5 CLASS **HockeyistType**

Тип хоккеиста.

В Раунде 1 чемпионата стратегия игрока управляет двумя хоккеистами-универсалами (`VERSATILE`).

В Раунде 2 команда состоит из одного `VERSATILE`, одного `FORWARD` и одного `DEFENCEMAN`.

В Групповом отборе и Финале у игрока в распоряжении имеется шесть хоккеистов типа `RANDOM`, однако одновременно в игре могут участвовать только трое из них. Остальные должны находиться на скамейке запасных.

DECLARATION

public final class HockeyistType

extends Enum

FIELDS

- public static final HockeyistType GOALIE
  - Вратарь.
    Автоматически управляемый хоккеист. Самостоятельно перемещается вдоль вертикального отрезка, расположенного перед воротами. Старается держаться на одной горизонтальной линии с шайбой. Скорость вратаря ограничена значением `game.goalieMaxSpeed`.
    Вратарь единственный среди хоккеистов взаимодействует с шайбой напрямую (как физический объект).
- public static final HockeyistType VERSATILE
  - Хоккеист со сбалансированными параметрами.
- public static final HockeyistType FORWARD
  - Нападающий.
    Хоккеист, основным достоинством которого является сила удара. Его слабой стороной является стойкость.
- public static final HockeyistType DEFENCEMAN
  - Защитник.
    Хоккеист, основным достоинством которого является стойкость. Его слабой стороной является точность удара.
- public static final HockeyistType RANDOM
  - Хоккеист со случайными параметрами.

METHODS

- *valueOf*
  `public static HockeyistType` **valueOf**`( String  name )`
- *values*
  `public static HockeyistType[]` **values**`( )`

4.1.6 CLASS **Move**

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

DECLARATION

public class Move

extends Object

METHODS

- *getAction*
`public ActionType` **getAction**`( )`
  - **Returns** - Возвращает текущее действие хоккеиста.

- *getPassAngle*
`public double` **getPassAngle**`( )`
  - **Returns** - Возвращает текущее направление паса.

- *getPassPower*
`public double` **getPassPower**`( )`
  - **Returns** - Возвращает текущую силу паса.

- *getSpeedUp*
`public double` **getSpeedUp**`( )`
  - **Returns** - Возвращает текущее ускорение хоккеиста.

- *getTeammateIndex*
`public int` **getTeammateIndex**`( )`
  - **Returns** - Возвращает текущий индекс хоккеиста, на которого будет произведена замена, или `-1`, если хоккеист не был указан.

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

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

- *setPassAngle*
`public void` **setPassAngle**`( double` **passAngle** `)`
  - **Usage**
    - Устанавливает направление паса (`ActionType.PASS`).
      Направление паса задаётся в радианах относительно текущего направления хоккеиста и должно лежать в интервале от `-0.5 * game.passSector` до `0.5 * game.passSector`.
      Значения, выходящие за указанный интервал, будут приведены к ближайшей его границе.

- *setPassPower*
`public void` **setPassPower**`( double` **passPower** `)`

- **Usage**
  - Устанавливает силу паса (`ActionType.PASS`).
    Сила паса является относительной величиной и должна лежать в интервале от `0.0` до `1.0`. Значения, выходящие за указанный интервал, будут приведены к ближайшей его границе. К значению реальной силы паса применяется также поправочный коэффициент `game.passPowerFactor`.

- *setSpeedUp*
`public void` **setSpeedUp**`( double` **speedUp** `)`

  - **Usage**
    - Устанавливает ускорение хоккеиста.
      Ускорение является относительным и должно лежать в интервале от `-1.0` до `1.0`. Значения, выходящие за указанный интервал, будут приведены к ближайшей его границе.

- *setTeammateIndex*
`public void` **setTeammateIndex**`( int` **teammateIndex** `)`

  - **Usage**
    - Устанавливает индекс хоккеиста для выполнения замены (`ActionType.SUBSTITUTE`).
      Индексация начинается с нуля. Значением по умолчанию является `-1`. Если в команде игрока не существует хоккеиста с указанным индексом, то замена произведена не будет.

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

  - **Usage**
    - Устанавливает угол поворота хоккеиста.
      Угол поворота задаётся в радианах относительно текущего направления хоккеиста и для хоккеиста с базовым значением атрибута подвижность и максимальным запасом выносливости ограничен интервалом от `-game.hockeyistTurnAngleFactor` до `game.hockeyistTurnAngleFactor`. Значения, выходящие за указанный интервал, будут приведены к ближайшей его границе. Положительные значения соответствуют повороту по часовой стрелке.

4.1.7 CLASS **Player**

Содержит данные о текущем состоянии игрока.

DECLARATION

---

public class Player **extends** Object

METHODS

- *getGoalCount*
`public int` **getGoalCount**`( )`

  - **Returns** - Возвращает количество шайб, заброшенных хоккеистами данного игрока в сетку противника. Шайбы, заброшенные во время состояния вне игры, не влияют на этот счётчик.

- *getId*
`public long` **getId**`( )`

  - **Returns** - Возвращает уникальный идентификатор игрока.

- *getName*
`public String` **getName**`( )`

  - **Returns** - Возвращает имя игрока.

- *getNetBack*
`public double` **getNetBack**`( )`

  - **Returns** - Возвращает абсциссу дальней от вратаря вертикальной границы ворот. Соответствует одному из значений `netLeft` или `netRight`.

- *getNetBottom*
`public double` **getNetBottom**`( )`

  - **Returns** - Возвращает ординату нижней штанги ворот.

- *getNetFront*
`public double` **getNetFront**`( )`

  - **Returns** - Возвращает абсциссу ближайшей к вратарю вертикальной границы ворот. Соответствует одному из значений `netLeft` или `netRight`.

- *getNetLeft*
`public double` **getNetLeft**`( )`

  - **Returns** - Возвращает абсциссу левой границы ворот.

- *getNetRight*
`public double` **getNetRight**`( )`

  - **Returns** - Возвращает абсциссу правой границы ворот.

- *getNetTop*
`public double` **getNetTop**`( )`

  - **Returns** - Возвращает ординату верхней штанги ворот.

- *isJustMissedGoal*
`public boolean` **isJustMissedGoal**`( )`

  - **Returns** - Возвращает `true` в том и только в том случае, если игрок только что пропустил гол. Вместе с установленным флагом `justScoredGoal` другого игрока означает, что сейчас состояние вне игры и новые голы не будут засчитаны. Длительность состояния вне игры составляет `game.afterGoalStateTickCount` тиков.

- *isJustScoredGoal*
`public boolean` **isJustScoredGoal**`( )`

  - **Returns** - Возвращает `true` в том и только в том случае, если игрок только что забил гол. Вместе с установленным флагом `justMissedGoal` другого игрока означает, что сейчас состояние вне игры и новые голы не будут засчитаны. Длительность состояния вне игры составляет `game.afterGoalStateTickCount` тиков.

- *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.

4.1.8 CLASS **Puck**

Класс, определяющий хоккейную шайбу. Содержит также все свойства юнита.

DECLARATION

public class Puck **extends** Unit

METHODS

- *getOwnerHockeyistId*
`public long` **getOwnerHockeyistId**`( )`
  - **Returns** - Возвращает идентификатор хоккеиста, контролирующего шайбу, или `-1`.

- *getOwnerPlayerId*
`public long` **getOwnerPlayerId**`( )`
  - **Returns** - Возвращает идентификатор игрока, контролирующего шайбу, или `-1`.

4.1.9 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.

- *getAngularSpeed*
`public double` **getAngularSpeed**`( )`
  - **Returns** - Возвращает скорость вращения объекта. Положительные значения соответствуют вращению по часовой стрелке.

- *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.

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

- *getMass*
`public double` **getMass**`( )`
  - **Returns** - Возвращает массу объекта в единицах массы.

- *getRadius*
`public double` **getRadius**`( )`

- **Returns** - Возвращает радиус объекта.

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

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

- *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 свеху вниз.

4.1.10 CLASS **World**

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

DECLARATION

public class World **extends** Object

METHODS

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

- *getHockeyists*
`public Hockeyist[]` **getHockeyists**`( )`
  - **Returns** - Возвращает список хоккеистов (в случайном порядке), включая вратарей и хоккеиста стратегии, вызвавшей этот метод. После каждого тика объекты, задающие хоккеистов, пересоздаются.

- *getMyPlayer*
`public Player` **getMyPlayer**`( )`
  - **Returns** - Возвращает вашего игрока.

- *getOpponentPlayer*
`public Player` **getOpponentPlayer**`( )`
  - **Returns** - Возвращает игрока, соревнующегося с вами.

- *getPlayers*
`public Player[]` **getPlayers**`( )`
  - **Returns** - Возвращает список игроков (в случайном порядке). После каждого тика объекты, задающие игроков, пересоздаются.

- *getPuck*
`public Puck` **getPuck**`( )`
  - **Returns** - Возвращает шайбу.

- *getTick*
`public int` **getTick**`( )`
  - **Returns** - Возвращает номер текущего тика.

- *getTickCount*
`public int` **getTickCount**`( )`
  - **Returns** - Возвращает базовую длительность игры в тиках. Реальная длительность может отличаться от этого значения в большую сторону.

- *getWidth*
`public double` **getWidth**`( )`
  - **Returns** - Возвращает ширину мира.

Глава 5

Package <none>

Package Contents *Page*

Interfaces

Стратегия — интерфейс, содержащий описание методов искусственного интеллекта хоккеиста.

5.1 Interfaces

5.1.1 INTERFACE Strategy

Стратегия — интерфейс, содержащий описание методов искусственного интеллекта хоккеиста. 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**`( Hockeyist` **self**`, World` **world**`, Game` **game**`, Move` **move** `)`
  - **Usage**
    - Основной метод стратегии, осуществляющий управление хоккеистом. Вызывается каждый тик для каждого хоккеиста.
  - **Parameters**
    - `self` - Хоккеист, которым данный метод будет осуществлять управление.
    - `world` - Текущее состояние мира.
    - `game` - Различные игровые константы.
    - `move` - Результатом работы метода является изменение полей данного объекта.