О CodeCraft 2020

Общие положения игры и правила проведения турнира

Данное соревнование предоставляет вам возможность проверить свои навыки программирования, создав искусственный интеллект (стратегию), участвующую в игре в специальном игровом мире (подробнее об особенностях мира CodeCraft 2020 можно узнать в следующих пунктах этой главы). В каждой игре вы будете противостоять стратегиям других участников. Задача вашей команды набрать больше очков, чем ваши соперники.

Турнир проводится в несколько этапов (**Раунд 1**, **Раунд 2** и **Финал**), 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 _ _ – 1 мест делится поровну между этими участниками. Например, если два участника делят первое место, то каждый из них получит половину суммы единиц рейтинга за первое и второе 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. За каждую N-ю полную неделю, прошедшую с момента отправки игроком последней стратегии, интервал участия для этого игрока увеличивается на N базовых интервалов тестирования. Учитываются только принятые системой стратегии. За каждое “падение” стратегии в 10 последних играх в **Песочнице** начисляется дополнительный штраф, равный 20% от базового интервала тестирования.

Интервал участия игрока в **Песочнице** не может стать больше суток.

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 попадут не более 1080 участников (если участников меньше, пройдет максимальное количество, кратное 4) с наибольшим рейтингом на момент начала этого этапа турнира (при равенстве рейтинга приоритет отдаётся игроку, раньше отправившему последнюю версию своей стратегии), а также дополнительный набор в следующие этапы турнира, включая **Финал**.

Раунд 1, как и последующие этапы, состоит из двух частей, между которыми будет небольшая пауза (с возобновлением работы Песочницы), позволяющая улучшить стратегию. Последняя отосланная стратегия перед началом каждой части выбирается для игр в соответствующей части. Игры проходят волнами. В каждой волне каждый участник играет ровно одну игру. Количество волн в каждой части определено способностями тестирующей системы, но гарантируется, что их будет не меньше десяти. 300 участников с наибольшим рейтингом проходят в **Раунд 2**. Также в **Раунд 2** проходят дополнительные 60 участников с наибольшим рейтингом в Песочнице (на момент старта **Раунда 2**), среди тех кто не прошел по результатам **Раунда 1**.

По результатам **Раунда 2** 50 лучших стратегий пройдут в **Финал**. Также в **Финал** проходят дополнительные 10 участников с наибольшим рейтингом в Песочнице (на момент старта **Финала**), среди тех кто не прошел по результатам **Раунда 2**.

Система проведения **Финала** имеет свои особенности. Этап будет также разделен на две части, но игры будут проходить не волнами. В каждой части, игры будут проходить между каждой парой участников **Финала**. Если время позволит, операция будет повторена.

Все финалисты сортируются по неубыванию рейтинга после окончания **Финала**. Если рейтинг одинаковый, более высокое место получает тот, чья стратегия была отправлена раньше. Призы по итогам **Финала** раздаются в соответствии с такой сортировкой.

После окончания **Песочницы** все участники, кроме победителей Финала, сортируются по неубыванию рейтинга. Если рейтинги одинаковые, более высокое место получает тот, чья последняя версия стратегии была отправлена раньше. Призы по итогам **Песочницы** раздаются в соответствии с такой сортировкой.

О тестировании и ограничениях стратегии

Время в игре дискретное и поделено на “тики”. В начале каждого тика, игровой симулятор передает состояние мира стратегиям участников, затем получает от них действия и обновляет состояние мира в соответствии с этими действиями и правилами игры. Затем процесс повторяется для следующего тика с обновленным состоянием. Длительность игры ограничена, но игра также может закончиться, если все стратегии “упали”.

“Упавшая” стратегия больше не может управлять действиями своего игрока. Стратегия считается “упавшей”, если:

- Процесс стратегии непредвиденно завершился, или произошла ошибка в протоколе взаимодействия стратегии с игровым сервером.

- Стратегия превысила одно из ограничений по времени. Существует ограничение на время ответа (выбора действия) на каждый тик - 1 секунда реального времени, а также суммарное ограничение по времени на всю игру - 40 секунд процессорного времени.
- Стратегия превысила ограничение памяти - 256 МБ.

Описание игры

Игра CodeCraft 2020 представляет собой стратегию, в которой вам предстоит управлять набором юнитов, собирать ресурсы, строить здания и атаковать противников.

Ваша цель - набрать больше очков чем ваши соперники. Игра заканчивается либо если достигнуто максимальное количество тиков, либо если остался лишь один (или ноль) игроков. В матчах 1 на 1 (Финал), если остается один игрок, ему добавляются очки, достаточные для победы.

Игровое поле представляет собой прямоугольную сетку, разделенную на клетки. Все игровые сущности имеют форму квадрата и находятся в целых координатах. За расстояние между двумя клетками в данной игре, принимается количество клеток по которым нужно пройти чтобы достичь цели, передвигаясь только по соседним клеткам (манхэттенское расстояние).

Поведение игровых сущностей определяется их свойствами.

Одно из важнейших свойств — размер сущности. Все сущности имеют форму квадрата, с длиной стороны равной данной величине.

Некоторые сущности могут двигаться (такие сущности называются юнитами). Юниты всегда имеют размер 1. Они могут передвинуться на соседнюю клетку за один тик, если эта клетка не занята другой сущностью.

Некоторые сущности могут атаковать, и все имеют здоровье и могут быть уничтожены. Если здоровье сущности становится меньше или равно нуля, сущность удаляется из игрового мира. Атакующие сущности имеют ограниченную дальность атаки. Каждый тик, во время которого происходит атака, определенное количество здоровья отнимается у цели.

Также, некоторые сущности могут ремонтировать другие сущности. Ремонтировать можно только соседние сущности (находящиеся на расстоянии 1). Каждый тик действия восстанавливается определенное количество здоровья цели. Во время ремонта здоровье цели не может стать больше максимального значения, указанного в его свойствах. Ремонтировать можно только живые сущности (с положительным здоровьем).

Некоторые атакующие сущности также могут собирать ресурсы с цели. За каждое очко нанесенного урона, фиксированное количество ресурса (обозначенное в свойствах цели) добавляется к игроку, владеющему атакующим.

Собранные ресурсы можно использовать для покупки новых юнитов и строений. Некоторые сущности обладают способностью строить (покупать) новые сущности. Тип новой сущности ограничен возможностями строителя, указанными в его свойствах. Для постройки (покупки) новой сущности необходимо потратить определенное количество ресурсов. Для юнитов (движущихся сущностей) точное количество необходимого ресурса равно значению, указанному в свойствах этого юнита, плюс текущее количество юнитов данного типа. Для остальных сущностей стоимость постройки всегда равна изначальной стоимости. Также нужно выбрать позицию, не занятую другими сущностями и находящуюся рядом со строителем (на расстоянии

1). Изначально построенные сущности будут иметь либо максимальное количество здоровья, либо значение указанное в свойствах строителя.

Когда сущность только была куплена, изначально она неактивна, то есть не может выполнять действий. Для активации сущности она должна сперва достигнуть максимального значения здоровья. Так что, если сущность была построена с неполным здоровьем, ее будет необходимо отремонтировать.

Также есть еще одно ограничение для постройки новых сущностей. Помимо ресурсов, есть еще один параметр, который называется “еда”. Некоторые сущности производят еду, а некоторые ее используют. Для постройки новой сущности, сумма произведенной еды среди активных сущностей игрока должна быть больше или равна сумме еды, потребляемой всеми сущностями игрока, включая новую построенную.

Последнее свойство сущности это расстояние зрения. Если включен туман войны, ваша стратегия видит лишь те сущности, которые расположены не дальше заданного расстояния от какой то из сущностей, контролируемой вами.

Список типов сущностей

Существует фиксированный набор типов сущностей в игре, и сущности одинаковых типов имеют одинаковые свойства. Вот полный список типов:

- Ресурс. Это единственная сущность, не управляемая никаким игроком. Ее размер `1` и она должна быть атакована юнитом-строителем, чтобы добыть ресурс.
- Юнит-строитель. Основная цель этого юнита — добывать ресурсы и строить здания.
- Юнит ближнего боя. Базовый юнит, наносящий урон в ближнем бою (расстояние атаки `1`).
- Юнит дальнего боя. Наносит урон на расстоянии.
- База строителей/юнитов ближнего боя/юнитов дальнего боя. Эти здания позволят купить новых юнитов соответствующего типа. Могут быть построены строителем.
- Стена. Маленькое здание, блокирующее проход для противника.
- Дом. Здание, производящее еду.
- Турель. Здание, способное атаковать врагов. Так как не может двигаться, больше подходит для защиты.

Интерфейс управления

Каждый тик вашей сратегии нужно отдавать приказы своим сущностям. Если вы не отдаете приказ, сущность продолжает выполнять предыдущее действие.

Действие состоит из действий атаки, постройки, ремонта и перемещения, которые имеют приоритет в данном порядке. То есть, если вы укажете несколько действий, только первое из возможных будет выполнено.

Для действия атаки можно указать конкретную цель, либо использовать автоатаку. При использовании автоатаки, вы также можете указать расстояние на которое юнит может искать путь до ближайшей цели.

Для действия ремонта необходимо указать цель.

Для действия постройки нужно указать тип сущности, а также положение. Положение сущности — клетка с минимальными координатами.

Для перемещения нужно указать целевую клетку. Юнит попробует найти путь до нее. Вы можете контролировать поиск пути, указав, следует ли искать ближайшую точку к цели. В противном случае, юнит не будет двигаться если путь до цели не найден. Также вы можете указать, искать ли путь, “пробивающий” сквозь другие сущности, атакуя и уничтожая их на своем пути. Такой алгоритм не будет рассматривать союзные сущности.

Когда игровой сервер производит поиск пути, используется простой алгоритм A* с ограничением на количество посещенных вершин.

Каждый игровой тик, сперва срабатывает алгоритм поиска пути для движущихся сущностей для определения потенциальной следующей позиции. Если целевая позиция действия движения сущности является соседней, поиск пути не выполняется, и запоминается эта позиция. Затем выполняются все действия атаки активных сущностей в случайном порядке. Если корректная цель для атаки не найдена, но в позиции, найденной на предыдущем шаге, находится враг, сущность атакует этого врага. Если здоровье цели было положительным и стало нулевым при атаке, цель считается уничтоженной и очки добавляются нападающему (но цель не удаляется из игры пока что). Далее выполняются все действия постройки в случайном порядке (если сущность выполнила действие атаки в этот тик, то действие постройки она уже выполнять не может). Затем аналогичным образом выполняются все действия ремонта. Ремонтируются только сущности с положительным здоровьем. В конце выполняется перемещение. Перемещение выполняется в несколько шагов. На каждом шаге юниты пытаются переместится в позицию, найденную на этапе поиска пути. Если несколько юнитов пытаются попасть в одну позицию, выбирается случайный. Если ни один юнит не может переместиться, фаза движения заканчивается.

В конце тика, сущности с нулевым здоровьем удаляются из игры, а сущности с полным здоровьем становятся активными.

Специфичные правила этапов

В **Раунде 1** вам предстоит изучить правила игры. Для простоты, не будет тумана войны, а также вам будут даны базы для каждого типа юнитов в начале игры, так что вы сможете сразу начать добывать ресурсы и атаковать врагов. Тем не менее, вы можете экспериментировать с постройками для подготовки к следующим этапам.

В **Раунде 2** вам уже необходимо освоить постройку. В начале вам будут доступны лишь строители. Будет необходимо построить базы для других типов юнитов. Также, будет включен тума войны и будет необходимо исследовать местность перед атакой. Задача осложняется тем, что после **Раунда 1**, часть слабых участников будет отсеяна и вам придется сражаться с более сильными соперниками.

Финал — самый важный этап. После первых двух раундов остаются только сильнейшие. Игры в финале будут 1 на 1. Кроме того, если в какой то момент в игре остался один игрок, ему добавляется достаточное количество очков для победы.

Использование приложения

У вас есть возможность запускать простые тестовые игры локально на своём компьютере. Для этого необходимо скачать архив для вашей операционной системы.

Использование приложения позволит вам тестировать свою стратегию в условиях, аналогичных условиям тестовой игры на сайте, но без каких либо ограничений по количеству создаваемых игр.

При запуске приложения, вы увидите экран конфигурации. Тут вы можете выбирать игроков, участвующих в игре, и настроить некоторые опции игры. Если вы хотите протестировать свою стратегию, выберите игрока TCP, а затем запустите свою стратегию. По умолчанию используется порт `31001`. После успешного подключения, вы сможете начать игру.

Если вы хотите поменять порт подключения, к примеру чтобы подключить несколько стратегий одновременно, при запуске языкового пакета можно передать хост и порт для подключения. К примеру, `./aicup2020 localhost 31002`.

После подключения всех игроков, помимо начала новой игры, возможно также начать новую игру с сохраненного прежде состояния, либо повтор сохраненной игры. При повторе игры, ваша стратегия получает данные об игре, но все действия игнорируются.

Управление в приложении:

- ПКМ / Shift-ЛКМ - перемещение камеры
- СКМ / Ctrl-ЛКМ - вращение камеры
- V - изменить режим визуализации.
- Ctrl-S - сохранить текущую игру в файл (позже можно пересмотреть или повторить)
- Ctrl-Shift-S - сохранить текущее состояние игры (позже можно загрузить это состояние и начать игру с него)
- P - пауза/продолжить
- Left/Right - потиковая перемотка времени (на паузе)

Вы можете также сохранить конфигурацию в файл, после чего запустить приложение с опцией `--config <file>`, пропуская экран конфигурации. Другие опции можно увидеть с помощью запуска с аргументом `--help`.## Значения свойств сущностей

Здесь вы можете ознакомиться с конкретными значениями свойств сущностей:

```
{
    House: (
        size: 3,
        build_score: 50,
        destroy_score: 500,
        can_move: false,
        population_provide: 5,
        population_use: 0,
        max_health: 50,
        initial_cost: 50,
        sight_range: 5,
        resource_per_health: 0,
        build: None,
        attack: None,
        repair: None,
    ),
    RangedUnit: (
        size: 1,
        build_score: 30,
        destroy_score: 300,
        can_move: true,
        population_provide: 0,
        population_use: 1,
```

```
        max_health: 10,
        initial_cost: 30,
        sight_range: 10,
        resource_per_health: 0,
        build: None,
        attack: Some((
            range: 5,
            damage: 5,
            collect_resource: false,
        )),
        repair: None,
    ),
    BuilderUnit: (
        size: 1,
        build_score: 10,
        destroy_score: 100,
        can_move: true,
        population_provide: 0,
        population_use: 1,
        max_health: 10,
        initial_cost: 10,
        sight_range: 10,
        resource_per_health: 0,
        build: Some((
            options: [
                House,
                Wall,
                BuilderBase,
                MeleeBase,
                RangedBase,
                Turret,
            ],
            init_health: Some(5),
        )),
        attack: Some((
            range: 1,
            damage: 1,
            collect_resource: true,
        )),
        repair: Some((
            valid_targets: [
                House,
                Wall,
                BuilderUnit,
                MeleeUnit,
                RangedUnit,
                BuilderBase,
                MeleeBase,
                RangedBase,
                Turret,
            ],
            power: 1,
        )),
    ),
```

```
MeleeUnit: (
    size: 1,
    build_score: 20,
    destroy_score: 200,
    can_move: true,
    population_provide: 0,
    population_use: 1,
    max_health: 50,
    initial_cost: 20,
    sight_range: 10,
    resource_per_health: 0,
    build: None,
    attack: Some((
        range: 1,
        damage: 5,
        collect_resource: false,
    )),
    repair: None,
),
Wall: (
    size: 1,
    build_score: 10,
    destroy_score: 10,
    can_move: false,
    population_provide: 0,
    population_use: 0,
    max_health: 50,
    initial_cost: 10,
    sight_range: 2,
    resource_per_health: 0,
    build: None,
    attack: None,
    repair: None,
),
Resource: (
    size: 1,
    build_score: 0,
    destroy_score: 0,
    can_move: false,
    population_provide: 0,
    population_use: 0,
    max_health: 30,
    initial_cost: 0,
    sight_range: 0,
    resource_per_health: 1,
    build: None,
    attack: None,
    repair: None,
),
Turret: (
    size: 2,
    build_score: 50,
    destroy_score: 500,
    can_move: false,
```

```
        population_provide: 0,
        population_use: 0,
        max_health: 100,
        initial_cost: 50,
        sight_range: 10,
        resource_per_health: 0,
        build: None,
        attack: Some((
            range: 5,
            damage: 5,
            collect_resource: false,
        )),
        repair: None,
    ),
    BuilderBase: (
        size: 5,
        build_score: 500,
        destroy_score: 5000,
        can_move: false,
        population_provide: 5,
        population_use: 0,
        max_health: 300,
        initial_cost: 500,
        sight_range: 5,
        resource_per_health: 0,
        build: Some((
            options: [
                BuilderUnit,
            ],
            init_health: None,
        )),
        attack: None,
        repair: None,
    ),
    RangedBase: (
        size: 5,
        build_score: 500,
        destroy_score: 5000,
        can_move: false,
        population_provide: 5,
        population_use: 0,
        max_health: 300,
        initial_cost: 500,
        sight_range: 5,
        resource_per_health: 0,
        build: Some((
            options: [
                RangedUnit,
            ],
            init_health: None,
        )),
        attack: None,
        repair: None,
    ),
```

```
    MeleeBase: (
        size: 5,
        build_score: 500,
        destroy_score: 5000,
        can_move: false,
        population_provide: 5,
        population_use: 0,
        max_health: 300,
        initial_cost: 500,
        sight_range: 5,
        resource_per_health: 0,
        build: Some((
            options: [
                MeleeUnit,
            ],
            init_health: None,
        )),
        attack: None,
        repair: None,
    ),
}
```

Описание API

В пакете для вашего языка программирования вы можете найти файл `MyStrategy.<ext>`/`my_strategy.<ext`
Этот файл содержит класс `MyStrategy` с методом `get_action`, где должна быть реализована логика вашей стратегии.

Этот метод будет вызываться каждый тик.

Метод принимает следующие аргументы:

- Доступная информация о текущем состоянии игры,
- Отладочный интерфейс — этот объект позволяет отправлять отладочные команды и запрашивать отладочное состояние приложения прямо из кода вашей стратегии. Заметьте, что этот объект недоступен при тестировании на сервере, а также использовании приложения в консольном режиме (batch mode). Он предназначен только для локальной отладки.

Метод должен вернуть действие, которое вы хотите выполнить в данный тик.

Для отладки существует еще один метод — `debug_update`, принимающий такие же параметры. Он вызывается постоянно во время работы приложения (но не в консольном режиме), если клиент находится в ожидании следующего тика. Метод будет вызван хотя бы раз между тиками.

Описание объектов

В этой секции, некоторые поля могут быть опциональными (обозначается как `Option<type>`). Способ реализации зависит от языка. При возможности используется специальный опциональный (nullable) тип, иначе другие методы могут быть использованы (например nullable указатели).

Некоторые объекты могут принимать несколько различных форм. Способ реализации зависит от языка. Если возможно, используется специальный (алгебраический) тип данных, иначе другие методы могут быть использованы (например варианты представлены классами, унаследованными от абстрактного базового класса).

`Vec2Float32`

Двумерный вектор

Поля:

- `x`: `float32` - Координата `x` вектора
- `y`: `float32` - Координата `y` вектора

`Color`

Цвет в формате RGBA

Поля:

- `r`: `float32` - Компонента красного цвета
- `g`: `float32` - Компонента зеленого цвета
- `b`: `float32` - Компонента синего цвета
- `a`: `float32` - Комнонента видимости (непрозрачности)

`ColoredVertex`

Вершина для отладочной отрисовки

Поля:

- `world_pos`: `Option<Vec2Float32>` - Позиция в мировых координатах (если отсутствует, используются координаты (0, 0) экрана)
- `screen_offset`: `Vec2Float32` - Дополнительное смещение в экранных координатах
- `color`: `Color` - Цвет

`PrimitiveType`

Тип примитивов для отладочной отрисовки

Варианты:

- `Lines` - Линии, количество вершин должно делиться на 2
- `Triangles` - Треугольники, количество вершин должно делиться на 3

`DebugData`

Данные для отладки, которые могут быть отображены при помощи приложения

Варианты:

- `Log` - Добавить запись в лог

  Поля:

  - `text`: `string` - Текст лога
- `Primitives` - Отрисовка примитивов

  Поля:

  - `vertices`: `[ColoredVertex]` - Вершины
  - `primitive_type`: `PrimitiveType` - Тип примитивов
- `PlacedText` - Отрисовка текста

  Поля:

  - `vertex`: `ColoredVertex` - Вершина для определения положения и цвета текста
  - `text`: `string` - Текст
  - `alignment`: `float32` - Выравнивание (0 - по левому краю, 0.5 - по центру, 1 - по правому краю)
  - `size`: `float32` - Размер шрифта в пикселях

DebugCommand

Команды, которые могут быть отправлены приложению для помощи в отладке

Варианты:

- `Add` - Добавить отладочные данные в текущий тик

  Поля:

  - `data`: `DebugData` - Данные для добавления
- `Clear` - Очистить отладочные данные текущего тика

  Нет полей

Vec2Int32

Двумерный вектор

Поля:

- `x`: `int32` - Координата `x` вектора
- `y`: `int32` - Координата `y` вектора

MoveAction

Действие перемещения

Поля:

- `target`: `Vec2Int32` - Целевая позиция
- `find_closest_position`: `boolean` - Находить ли ближайшее положение, если до цели путь не найден
- `break_through`: `boolean` - Уничтожать ли враждебные сущности на пути

`EntityType`

Тип сущности

Варианты:

- `Wall` - Стена, может использоваться для блокировки пути противнику
- `House` - Дом, производит еду
- `BuilderBase` - База для покупки юнитов-строителей
- `BuilderUnit` - Юнит-строитель может строить здания
- `MeleeBase` - База для покупки юнитов ближнего боя
- `MeleeUnit` - Юнит ближнего боя
- `RangedBase` - База для покупки юнитов дальнего боя
- `RangedUnit` - Юнит дальнего боя
- `Resource` - Ресурс, может быть собран
- `Turret` - Здание способное атаковать на расстоянии

`BuildAction`

Действие постройки

Поля:

- `entity_type`: `EntityType` - Тип сущности для постройки
- `position`: `Vec2Int32` - Желаемая позиция новой сущности

`AutoAttack`

Настройки автоматической атаки

Поля:

- `pathfind_range`: `int32` - Максимальное расстояние для поиска пути
- `valid_targets`: `[EntityType]` - Список типов сущностей, которые следует атаковать. Если пусто, все типы кроме ресурса будут рассмотрены

`AttackAction`

Действие атаки

Поля:

- `target`: `Option<int32>` - ID цели, если применимо
- `auto_attack`: `Option<AutoAttack>` - Настройки автоматической атаки, если необходимо

`RepairAction`

Действие починки

Поля:

- `target`: `int32` - ID цели

`EntityAction`

Действие сущности

Поля:

- `move_action`: `Option<MoveAction>` - Действие перемещения
- `build_action`: `Option<BuildAction>` - Действие постройки
- `attack_action`: `Option<AttackAction>` - Действие атаки
- `repair_action`: `Option<RepairAction>` - Действие починки

`Action`

Действие игрока

Поля:

- `entity_actions`: `Map<int32 -> EntityAction>` - Новые действия для сущностей. Если сущность не получила новое действие, она будет продолжать выполнять предыдущее

`ClientMessage`

Сообщение отправляемое клиентом

Варианты:

- `DebugMessage` - Отправить отладочную команду приложению

  Поля:

  - `command`: `DebugCommand` - Команда для исполнения
- `ActionMessage` - Ответ на ServerMessage::GetAction

  Поля:

  - `action`: `Action` - Действие игрока
- `DebugUpdateDone` - Сигнализирует окончание отладочного обновления

  Нет полей
- `RequestDebugState` - Запросить отладочное состояние приложения

  Нет полей

`BuildProperties`

Свойства строительства сущности

Поля:

- `options`: `[EntityType]` - Возможные типы новой сущности
- `init_health`: `Option<int32>` - Изначальное здоровье новой сущности. Если отсутствует, новая сущность будет имет полное здоровье

`AttackProperties`

Свойства атаки сущности

Поля:

- `attack_range`: `int32` - Максимальное расстояние атаки
- `damage`: `int32` - Урон наносимый за один тик
- `collect_resource`: `boolean` - Собираются ли ресурсы с цели при атаке

`RepairProperties`

Свойства ремонта сущности

Поля:

- `valid_targets`: `[EntityType]` - Типы сущностей, которые возможно ремонтировать
- `power`: `int32` - Здоровье восстанавливаемое за один тик

`EntityProperties`

Свойства сущности

Поля:

- `size`: `int32` - Размер. Сущности имеют форму квадрата со стороной заданной длины
- `build_score`: `int32` - Количество очков за постройку данной сущности
- `destroy_score`: `int32` - Количество очков за уничтожение данной сущности
- `can_move`: `boolean` - Может ли данная сущность перемещаться
- `population_provide`: `int32` - Количество производимой еды, если сущность активна
- `population_use`: `int32` - Количество потребляемой еды
- `max_health`: `int32` - Максимальное количество очков здоровья
- `initial_cost`: `int32` - Стоимость постройки первой сущности данного типа. Если это юнит (сущность может перемещаться), стоимость увеличивается на 1 за каждого существующего юнита этого типа
- `sight_range`: `int32` - Если включен туман войны, расстояние на котором другие сущности считаются видимыми
- `resource_per_health`: `int32` - Количество ресурса добавляемое нападающему, способному собирать русурсы, за каждую единицу урона
- `build`: `Option<BuildProperties>` - Свойства строительства, если сущность способна строить
- `attack`: `Option<AttackProperties>` - Свойства атаки, если сущность способна атаковать
- `repair`: `Option<RepairProperties>` - Свойства ремонта, если сущность способна ремонтировать

`Player`

Игрок (стратегия, клиент)

Поля:

- `id`: `int32` - ID игрока
- `score`: `int32` - Текущий счет
- `resource`: `int32` - Текущее количество ресурса

`Entity`

Игровая сущность

Поля:

- `id`: `int32` - ID сущности. Уникально для каждой сущности
- `player_id`: `Option<int32>` - ID игрока, владеющего сущностью, если применимо
- `entity_type`: `EntityType` - Тип сущности
- `position`: `Vec2Int32` - Позиция сущности (угол с минимальными координатами)
- `health`: `int32` - Текущее здоровье
- `active`: `boolean` - Если сущность активна, она может выполнять действия

`PlayerView`

Доступная игроку информация

Поля:

- `my_id`: `int32` - ID вашего игрока
- `map_size`: `int32` - Размер карты
- `fog_of_war`: `boolean` - Включен ли туман войны
- `entity_properties`: `Map<EntityType -> EntityProperties>` - Свойства сущностей для каждого типа
- `max_tick_count`: `int32` - Максимальная длительность игры в тиках
- `max_pathfind_nodes`: `int32` - Максимальное количество вершин для поиска пути в игровом симуляторе
- `current_tick`: `int32` - Текущий тик
- `players`: `[Player]` - Список игроков
- `entities`: `[Entity]` - Список сущностей

`ServerMessage`

Сообщение отправляемое сервером

Варианты:

- `GetAction` - Получить действие для следующего тика

  Поля:

  - `player_view`: `PlayerView` - Информация доступная игроку
  - `debug_available`: `boolean` - Доступен ли отладочный интерфейс приложения

- `Finish` - Сигнализирует конец игры

  Нет полей

- `DebugUpdate` - Отладочное обновление

  Поля:

  - `player_view`: `PlayerView` - Информация доступная игроку

`Camera`

Камера используемая для отрисовки

Поля:

- `center`: `Vec2Float32` - Точка на которую смотрит камера
- `rotation`: `float32` - Угол поворота
- `attack`: `float32` - Угол атаки
- `distance`: `float32` - Расстояние до цели
- `perspective`: `boolean` - Применяется ли перспектива

`DebugState`

Состояние для отладки, получаемое из приложения

Поля:

- `window_size`: `Vec2Int32` - Размер окна для отрисовки
- `mouse_pos_window`: `Vec2Float32` - Положение курсора в оконных координатах
- `mouse_pos_world`: `Vec2Float32` - Положение курсора в мировых координатах
- `pressed_keys`: `[string]` - Кнопки, нажатые в данный момент
- `camera`: `Camera` - Текущая камера используемая для отрисовки
- `player_index`: `int32` - Индекс вашего игрока