Magenta Code Game Challenge

Magenta team

Самара, 2016

Оглавление

1 Code Plumber 1
- 1.1 Обзор . . . 1
- 1.2 Правила . . . 2

2 Программирование вашего игрока 5
- 2.1 Обзор . . . 5
- 2.2 Инициализация . . . 5
- 2.3 Движение вашего игрока . . . 6
- 2.4 Примеры простых стратегий . . . 6
  - 2.4.1 Простой Java код (Go to center player) . . . 6
  - 2.4.2 Простой C++ код (Go to center player) . . . 6
- 2.5 Клиент Code Game Challenge . . . 7

3 JavaDoc 9
- 3.1 Interface Bonus . . . 10
  - 3.1.1 Описание . . . 10
  - 3.1.2 Методы . . . 10
- 3.2 Interface Effect . . . 10
  - 3.2.1 Описание . . . 10
  - 3.2.2 Методы . . . 10
- 3.3 Interface Enemy . . . 11
  - 3.3.1 Описание . . . 11
  - 3.3.2 Методы . . . 11
- 3.4 Interface Leakage . . . 11
  - 3.4.1 Описание . . . 12
  - 3.4.2 Методы . . . 12
- 3.5 Interface MovableUnit . . . 12
  - 3.5.1 Описание . . . 12
  - 3.5.2 Методы . . . 12
- 3.6 Interface Obstacle . . . 13
  - 3.6.1 Описание . . . 13
- 3.7 Interface SelfControl . . . 13
  - 3.7.1 Описание . . . 13
  - 3.7.2 Методы . . . 13
- 3.8 Interface Unit . . . 16
  - 3.8.1 Описание . . . 16
  - 3.8.2 Методы . . . 16
- 3.9 Interface World . . . 18
  - 3.9.1 Описание . . . 18
  - 3.9.2 Методы . . . 18

3.10 Class Action 19
3.10.1 Описание 19
3.10.2 Поля 19
3.11 Class BonusType 19
3.11.1 Описание 20
3.11.2 Поля 20
3.12 Class EffectType 20
3.12.1 Описание 20
3.12.2 Поля 20
3.13 Class Player 20
3.13.1 Описание 20
3.13.2 Методы 20
3.14 Class PlayerType 21
3.14.1 Описание 22
3.14.2 Поля 22
3.15 Class ShellType 22
3.15.1 Описание 22
3.15.2 Поля 22

Глава 1

Code Plumber

1.1 Обзор

Code Game Challenge даёт вам возможность проверить свои способности в программировании против других команд в мире Code Plumber. Каждая команда пишет свою игровую стратегию для рабочего-водопроводчика. Ваш рабочий будет помещен в смоделированный мир вместе с рабочими других игроков.

Турнир сталкивает игроков друг с другом в серии матчей. В каждом матче принимают участие четверо рабочих, соревнующиеся друг с другом. В начале у рабочих нулевая скорость.

Водопроводчики могут перемещаться в двумерном прямоугольном мире. Они могут собирать бонусы, чинить протечки и бить соперников гаечным ключом. Рабочие зарабатывают очки за починку протечек. Одновременно в мире существуют ограниченное количество протечек. Протечка появляется в случайном месте и со временем, если её не чинят, начинает расти. Рабочий может чинить любую протечку. Задача игрока — починить как можно больше протечек.

В игровом мире есть бонус «Ускорение работы». Подобравший этот бонус рабочий начинает чинить протечку в 2 раза быстрее. Также есть бонус «Заморозка». Если какой-то рабочий подбирает этот бонус, то все протечки замерзают и перестают расти, но чинить их при этом нельзя.

Если рабочий бьет другого рабочего ключом, то второй выходит из строя на некоторое время. По истечении этого времени рабочий снова получает возможность двигаться и совершать действия.

Главная цель игры — набрать как можно больше очков в течение матча. Очки набираются в момент, когда рабочий чинит протечку.

Часть программирования Code Game Challenge начинается, когда вы получаете доступ к соревнованию. Вы можете использовать один из языков программирования, поддерживаемых платформой, — Java или C++. Вам также будет предоставлено специальное клиентское приложение Code Game Challenge, которое описано в соответствующей главе данного руководства.

У вас есть возможность отправлять в тестирующую систему класс вашего игрока в течение фазы программирования Code Game Challenge. Финальные версии классов всех игроков (последняя успешно скомпилированная посылка, которую вы сделали) будут участвовать в турнире Code Game Show. Все игроки будут состязаться как минимум в двух матчах. Победителем Code Game Challenge станет команда, чья стратегия наберет больше всего очков в финальном блоке турнира. Более подробное описание вы можете найти в секции «Правила».

Следующая секция этой главы описывает игровые правила более детально. Оставша-

яся часть данного руководства показывает, как создать игрока и как воспользоваться его различными возможностями. JavaDoc-файлы доступны как в напечатанном виде (в двух последних главах данного руководства), так и на компьютере, который будет вам предоставлен. В JavaDoc описаны классы и интерфейсы, которые вы можете использовать при программировании вашего игрока.

1.2 Правила

Турнир состоит из трёх блоков. Первые два блока состоят из двух матчей, третий — из трёх. В матче участвуют по 4 игрока. В каждом блоке перед первой серией матчей игроки группируются случайно. Перед второй и третьей сериями матчей игроки перегруппировываются в зависимости от общего количества заработанных очков в данном блоке.

Первый отсев произойдет после отборочного блока. Игроки будут отсортированы по общему количеству очков, и первые 16 пройдут в полуфинал.

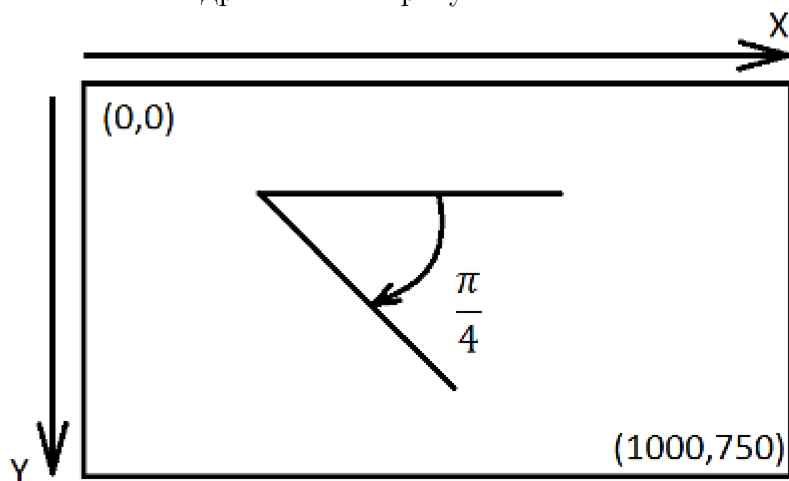
Второй отсев произойдет после полуфинала. Игроки будут отсортированы по общему количеству очков, и первые 4 пройдут в финал.

После каждого блока набранные очки обнуляются.

Игра происходит следующим образом. В начале каждый игрок имеет возможность провести инициализацию. После этого игроки делают свои ходы. Игровое время дискретно, и 1 тик означает 1 ход каждого игрока. Во время инициализации игровое время 0. Во время первого хода каждого игрока игровое время становится равным 1 и так далее. Игра длится 4000 тиков.

Существует временной лимит в 0.25 с на инициализацию и совершение хода игроком.

Игровой мир представляет из себя прямоугольник размером 1000×750, расположенный на плоскости. Подробности на рисунке ниже.



Размер мира не зависит от разрешения экрана и наоборот.

Все заработанные в течение матча очки добавляются к общему количеству очков, заработанных в конкретном блоке турнира.

Существует 6 типов объектов мира (юнитов): рабочий, бонус, протечка, препятствие. Каждый юнит, за исключением препятствия, — это круг с определенным радиусом. Препятствие — это квадрат, в его случае за радиус принимаем радиус вписанной окружности. Радиус протечки может меняться в зависимости от её состояния. Радиусы юнитов представлены в следующей таблице:

Unit	R
Рабочий	30
Бонус	20
Протечка (малая, средняя, большая)	25,35,45
Препятствие	30

Препятствия объединяются в блоки, образуя карту мира. Для каждой игры карта выбирается случайным образом.

Каждый рабочий начинает матч в заданном для его цвета месте.

Два юнита взаимодействуют, если расстояние между их центрами меньше либо равно сумме их радиусов. Результаты взаимодействия рабочего с другими юнитами представлен в таблице ниже:

Unit	**Результат взаимодействия с активным рабочим**
Рабочий	Скорости и направления рабочих меняются
Бонус «Заморозка»	Все протечки, существующие в данный момент в мире, замораживаются на 100 тиков
Бонус «Ускорение работы»	В течение 300 тиков рабочий чинит протечку в 2 раза быстрее.
Протечка	Рабочий получает возможность чинить протечку
Препятствие	Скорость и направление рабочего изменяются.

Когда рабочий сталкивается с границами мира, его скорость изменяется так, как описано ниже.

Вы можете предполагать, что все взаимодействия (включая взаимодействия с границами мира) абсолютно упругие, и скорость объектов изменяется соответственно. Считается, что масса всех рабочих одинакова.

За один тик рабочий может сделать один шаг в одном из направлений: вперёд, назад, вправо, влево. Расстояние, которое может пройти рабочий, делая шаг вперёд или назад, составляет n точек/тик, влево или вправо — $\frac{n}{2}$ точек/тик. После каждого шага рабочий полностью останавливается.

Угол вращения рабочего ограничен и составляет не более α радиан/тик, где α — максимальный угол поворота.

Характеристики рабочего представлены в таблице ниже:

Характеристика	Единицы измерения	Значение
Максимальное расстояние при движении по прямой	точек/тик	2
Максимальное расстояние при движении в стороны	точек/тик	1
Максимальный угол поворота	радиан/тик	0.1

В каждый момент времени в мире существует ограниченное количество протечек. Протечки появляются в случайных местах. У каждой протечки есть параметр HH, обозначающий количество человекочасов, необходимых для её починки. Максимальное количество человекочасов, необходимых для починки, ограничено и равно m. В момент появления протечки в мире этот параметр равен 0. В течение первых 100 тиков его количество не увеличивается. Затем параметр начинает расти на 1 пункт каждые 5 тиков. Когда количество человекочасов становится больше или равно k или $l > k$, радиус протечки изменяется на средний или большой соответственно. Рост параметра прекращается в момент достижения им максимального значения m или же в момент, когда протечку впервые начнут

чинить. После этого значение параметра может только уменьшаться. При уменьшении параметра до значений, меньших l, а затем меньших k, размер протечки уменьшается соответственно до средней и малой. В момент, когда параметр становится меньше 0, протечка считается ликвидированной и исчезает.

Значения параметров представлены в таблице ниже:

Количество человекочасов (параметр HH)	Значение
Значение, при котором протечка изменяет размер на средний (k)	50
Значение, при котором протечка изменяет размер на большой (l)	100
Максимальное значение (m)	200

Рабочий может совершать следующие действия:

- Изменять своё направление и скорость
- Собирать бонусы
- Чинить протечки
- Бить ключом

Если расстояние между рабочим и протечкой меньше чем *(радиус рабочего + радиус протечки + 10)* и угол между рабочим и протечкой меньше или равен $|\frac{\pi}{2}|$, то рабочий получает возможность её чинить. При взаимодействии с протечкой рабочий уменьшает параметр HH с постоянной скоростью 1 пункт/тик. Рабочий получает очки каждый тик за потраченные на ремонт протечки человекочасы.

Если расстояние между рабочими меньше чем *(радиус одного рабочего + радиус другого рабочего + 15)* и угол между управляемым рабочим и рабочим другого игрока меньше или равен $|\frac{\pi}{4}|$, управляемый рабочий может ударить другого ключом. После удара рабочий другого игрока становится неактивным на 200 тиков. В следующий раз управляемый рабочий сможет нанести удар ключом какому-либо другому рабочему не ранее, чем через 40 тиков.

Неактивный рабочий не может двигаться, поворачивать или совершать другие действия. Также после выхода из неактивного состояния рабочий не может ударять ключом в течение 40 тиков.

Глава 2

Программирование вашего игрока

2.1 Обзор

Чтобы запустить клиентское приложение на предоставленном вам компьютере, вам необходимо перейти на диск `W:`. В корневом каталоге этого диска содержится директория `client`. В её поддиректории `Projects` вы найдёте заранее подготовленные проекты для различных IDE, в которых уже выполнены все необходимые настройки.

Разумеется, вы можете сами вручную настроить любую IDE.

Файлы библиотек для Java (`plumber.jar`, `plumber-javadoc.jar`) находятся в каждой из директорий `EclipseProject`, `IdeaProject`, `NetBeansProject` в поддиректории `lib`.

Файл библиотеки и пустую реализацию класса для C++ (`framework.hpp`, `MyStrategy.cpp`) вы можете найти в директории `ForCpp`.

Ваша задача — реализовать класс **MyStrategy**.

Важно! Не переименовывайте этот класс.

Важно! Класс должен лежать вне каких либо пакетов.

Класс **MyStrategy** содержит два метода — **init()** и **move()**. Изменение этих методов — единственный способ персонализации вашего игрока. Вы также можете добавлять другие методы и поля в класс **MyStrategy**.

2.2 Инициализация

Когда ваш рабочий появляется в игровом мире, происходит вызов метода **init()** игровой стратегии. Поместите в данный метод любой код инициализации, который должен быть вызван. Симулятор предоставляет ограниченное количество времени (0.25 с) на выполнение кода инициализации перед началом игры. Если ваш инициализационный код не успел завершиться вовремя или упал с ошибкой (exception) или ошибкой времени выполнения (runtime error), ваша стратегия не будет участвовать в игре. Помните, что вы не должны вызывать метод **getWorld()** в момент инициализации, поскольку мир в этот момент ещё не существует.

Вашей стратегии уже присвоено имя, которое обычно содержит имя команды. Однако вы можете присвоить альтернативное имя вашему игроку. Для этого нужно во время инициализации вызвать **getSelfControl().setName()** с желаемым именем игрока в качестве аргумента. Подробнее об ограничениях именования написано в JavaDoc данного метода.

2.3 Движение вашего игрока

После того как симулятор вызовет методы **init()** каждого игрока, он начинает поочерёдно вызывать методы **move()** каждого игрока. Код в методе **move()** вашего игрока описывает действия, которые будет совершать ваша стратегии во время игры.

Ваш игрок может находить объекты в мире, вызывая метод **getWorld()**. Также вы можете контролировать своего рабочего (например, задавать желаемую скорость), используя возвращаемый методом **getSelfControl()** экземпляр интерфейса **SelfControl**. Обратитесь к JavaDoc за более подробным описанием.

Главная идея заключается в том, что ваш игрок не совершает никаких действий. Он лишь показывает желание совершить действие. Когда симулятор заканчивает опрос метода **move()** всех стратегий в конкретный тик, он симулирует желаемые игроками действия. Все ограничения, введённые правилами, применяются в этот момент. Например, если рабочий не активен, все его желания двигаться будут проигнорированы. После того как совершены все действия всех активных стратегий, наступает следующий тик.

Симулятор предоставляет максимум 0.25 с на выполнение кода метода **move()** каждой стратегии в течение одного тика. Если код не успел завершиться вовремя или упал с ошибкой (exception) или ошибкой времени выполнения (runtime error), ваша стратегия прекращает участие в этой игре.

2.4 Примеры простых стратегий

2.4.1 Простой Java код (Go to center player)

```java
import plumber.interfaces.Player;
import plumber.interfaces.World;

public class MyStrategy extends Player {

        @Override
        public void init() {
                getSelfControl().setName("GoToCenter");
        }

        @Override
        public void move() {
                World world = getWorld();
                getSelfControl().turnTo( world.getWidth() / 2,
                world.getHeight() / 2);
                getSelfControl().stepForward();
        }
}
```

2.4.2 Простой C++ код (Go to center player)

```cpp
#include "framework.hpp"

using namespace std;

class MyStrategy : public Player {
```

```cpp
public:
        MyStrategy(SelfControl &selfControl, World &world) : Player(
           selfControl, world) {}

        void init() override
        {
                getSelfControl().setName("GoToCenter");
        }

        void move() override
        {
                World &world = getWorld();
                getSelfControl().turnTo(world.getWidth()/2,
                world.getHeight()/2);
                getSelfControl().stepForward();
        };
};
```

2.5 Клиент Code Game Challenge

Чтобы запустить клиентское приложение на предоставленном вам компьютере, вам необходимо перейти на диск `W:`. В директории `client` содержится файл `start.bat`, который и запускает клиентское приложение. Используйте выданные вам логин и пароль для авторизации в клиенте.

Клиент даёт вам возможность тестировать вашего игрока и отправлять ваш код на сервер.

Когда у вас появляется версия класса MyStrategy, которую вы хотите протестировать, вы можете использовать вкладки «Локальный запуск» или «Глобальный запуск» клиента.

Обратите внимание, что загружать необходимо только один файл (`.java`, `.cpp`).

Вкладка «Локальный запуск» даёт вам возможность биться против нескольких простых стратегий и стратегий, написанных вами ранее. Выберите стратегии и запустите игру.

Также вы можете просмотреть код загруженной ранее стратегии, нажав на кнопку «открыть» рядом с именем стратегии.

Вкладка «Глобальный запуск» даёт вам возможность бороться со стратегиями других команд. Когда вы отправляете вашу стратегию на вкладке «Глобальный запуск», клиент загружает информацию о последних посылках всех команд. Это даёт вам возможность запускать игру с любыми игроками, имеющимися в настоящий момент в системе, включая вашего игрока.

Обратите внимание на различие между локальным и глобальным запуском. Локальный запуск позволяет вам изменять свою стратегию и запускать её снова и снова без отправки на сервер. Глобальный запуск предоставляет возможность тестировать только последнюю версию вашей стратегии.

Если вы сделали изменения в стратегии и хотите её проверить, вы должны загрузить её заново с помощью вкладки «Локальный запуск» для локального теста или вкладки «Глобальный запуск» для отправки на сервер.

Результат вашего локального/глобального запуска будет доступен только вам. Эти результаты никак не влияют на ваше положение в турнире Code Game Challenge.

На вкладке «Локальный запуск» есть возможность отключить показ начальных титров перед игрой. Для этого необходимо снять соответствующий флажок.

У вас есть возможность удалить свою ранее загруженную стратегию с вкладки «Локальный запуск». Для этого необходимо кликнуть по удаляемой стратегии правой кнопкой мыши и выбрать соответствующий пункт.

С помощью клавиши Pause/Break вы можете приостанавливать и продолжать воспроизведение игры при просмотре.

Во время локального тестирования вы можете использовать логи в консоли клиентского приложения. Для этого можете использовать стандартные потоки вывода ($Java$ — $System.out$, $C++$ — $cout$)

На вкладках локального и глобального запуска вы также можете выбрать карту, на которой будет происходить битва стратегий. Карта определяет расположение препятствий и протечек. Карты разделены на 4 цветовых группы:

- Серая — в эту группу входит только карта типа «Пустое поле»
- Зелёная — лёгкие карты. Мало препятствий. Много протечек
- Жёлтая — средние. Среднее количество препятствий. Среднее количество протечек
- Красная — сложные. Много препятствий. Мало протечек.

Во время турнира карты будут выбираться по следующему принципу: в отборочных играх — лёгкие карты, в полуфиналах — средние, в финале — сложные карты.

Компиляция стратегии при глобальном запуске происходит на сервере. Вы получите сообщение со статусом компиляции после небольшого ожидания. Помните, что вы должны отправлять только файл, содержащий класс MyStrategy. Весь ваш код должен находиться в одном файле. Ваша стратегия не должна использовать какие либо I/O операции. Закомментируйте весь код логирования перед отправкой на сервер.

Существует специальная возможность поиграть в Code Game Challenge в качестве игрока. Для этого необходимо выбрать «Ручное управление» и выбрать цвет игрока, за которого вы хотите играть. Вы получите возможность управлять стратегией не с помощью кода, а с клавиатуры. Все ограничения правил также действуют. Ручное управление доступно только в режиме локального запуска. Для управления используются следующие клавиши:

- W - Шаг вперёд
- S - Шаг назад
- A - Поворот налево
- D - Поворот направо
- Q - Шаг влево
- E - Шаг вправо
- O - Чинить протечку
- SPACE - Удар ключом

Глава 3

JavaDoc

Interfaces

Bonus 10
Интерфейс, описывающий бонус.
Effect 10
Интерфейс, описывающий эффекты, демонстрирующие состояние игрока.
Enemy 11
Интерфейс, предоставляющий доступ к информации о противнике.
Leakage 11
Интерфейс, предоставляющий информацию о протечках
MovableUnit 12
Интерфейс, описывающий движущийся объект
Obstacle 13
Интерфейс, описывающий препятствие.
SelfControl 13
Интерфейс, позволяющий получить информацию об игроке и управлять им.
Unit 16
Базовый интерфейс для всех объектов игрового мира.
World 18
Интерфейс, предоставляющий доступ к свойствам игрового мира.

Classes

Action 19
Типы действий, которые могут быть совершены игроком: Ударить гаечным ключом
BonusType 19
Типы бонусов в игровом мире: скорость работы, заморозка кранов
EffectType 20
Типы эффектов, которые могут быть применены к игроку: не способен двигаться, скорость починки кранов увеличена вдвое
Player 20
Абстрактный класс игрока.
PlayerType 21
Возможные цвета игроков
ShellType 22
Тип ущерба, который игрок может нанести противнику: удар ключом

3.1 Interface Bonus

Интерфейс, описывающий бонус. Является наследником Unit, расширяет этот интерфейс методом получения типа бонуса.

3.1.1 Описание

```
public interface Bonus extends Unit
```

3.1.2 Методы

- **getBonusType**

  ```
  BonusType getBonusType()
  ```

  - **Описание**
    Метод возвращает тип бонуса. Существует два типа бонусов: WORK — увеличивает скорость починки кранов. FREEZE — замораживает краны.
  - **Возвращает** — тип бонуса

3.2 Interface Effect

Интерфейс, описывающий эффекты, демонстрирующие состояние игрока.

3.2.1 Описание

```
public interface Effect
```

3.2.2 Методы

- **getRemainingDuration**

  ```
  int getRemainingDuration()
  ```

  - **Описание**
    Возвращает оставшееся время действия эффекта
  - **Возвращает** — время действия в единицах игрового времени

- **getType**

  ```
  EffectType getType()
  ```

  - **Описание**
    Возвращает тип эффекта. Существует два вида эффектов: эффект DISABLED — игрок не может двигаться; эффект INDUSTRIOUS — игрок быстрее чинит кран.
  - **Возвращает** — тип эффекта.

3.3 Interface Enemy

Интерфейс, предоставляющий доступ к информации о противнике.

3.3.1 Описание

```
public interface Enemy extends MovableUnit
```

3.3.2 Методы

- **getEffects**

  ```
  List<Effect> getEffects()
  ```

  - **Возвращает** — список эффектов, применённых к противнику

- **getName**

  ```
  String getName()
  ```

  - **Возвращает** — Имя противника

- **getPlayerType**

  ```
  PlayerType getPlayerType()
  ```

  - **Возвращает** — `PlayerType` противника.

- **hasEffectOfType**

  ```
  boolean hasEffectOfType(EffectType effectType)
  ```

  - **Описание**
    Проверяет, есть ли некоторый эффект среди эффектов, применённых к противнику.
  - **Параметры**
    * `effectType` — проверяемый эффект
  - **Возвращает** – true, если эффект применяется в настоящее время к противнику, иначе false.

3.4 Interface Leakage

Интерфейс, предоставляющий информацию о протечках

3.4.1 Описание

```
public interface Leakage extends Unit
```

3.4.2 Методы

- **getRemainingLeak**

  ```
  int getRemainingLeak()
  ```

  - **Возвращает** — количество человекочасов, необходимых для починки протечки

3.5 Interface MovableUnit

Интерфейс, описывающий движущийся объект

3.5.1 Описание

```
public interface MovableUnit extends Unit
```

3.5.2 Методы

- **getHorizontalSpeed**

  ```
  float getHorizontalSpeed()
  ```

  - **Описание**
    Возвращает горизонтальную составляющую скорости
  - **Возвращает** — проекция вектора скорости на ось Ox

- **getSpeed**

  ```
  float getSpeed()
  ```

  - **Описание**
    Возвращает абсолютную величину скорости объекта
  - **Возвращает** — длина вектора скорости

- **getSpeedAngle**

  ```
  float getSpeedAngle()
  ```

  - **Описание**
    Возвращает угол между направлением вектора скорости и осью Ox
  - **Возвращает** — угол в радианах

- **getVerticalSpeed**

```
float getVerticalSpeed()
```

  - **Описание**
    Возвращает вертикальную составляющую скорости
  - **Возвращает** — проекция вектора скорости на ось Oy

3.6 Interface Obstacle

Интерфейс, описывающий препятствие. Наследник интерфейса Unit, дополнительных методов не содержит

3.6.1 Описание

```
public interface Obstacle extends Unit
```

3.7 Interface SelfControl

Интерфейс, позволяющий получить информацию об игроке и управлять им. В момент инициализации можно задать имя игрока. Во время игры — изменить направление и скорость движения игрока. Дать команду чинить кран. Ударить соперника ключом

3.7.1 Описание

```
public interface SelfControl extends Enemy
```

3.7.2 Методы

- **getActionCoolDown**

```
int getActionCoolDown(Action action)
```

  - **Описание**
    Определяет, сколько единиц игрового времени осталось до возможности выполнить действие action. Если возвращается значение 0, действие можно выполнять в текущий момент времени.
  - **Параметры**
    * `action` — действие
  - **Возвращает** — количество единиц игрового времени

- **getScore**

```
int getScore()
```

  - **Возвращает** — Количество очков, набранное игроком на момент запроса

- **repair**

```
void repair()
```

  - **Описание**

    Чинит кран если он находится в зоне доступности

    Ограничения:

    1) методы stepForward(), stepBack(), sideRight(), stepLeft() и repair() взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

    2) методы wrenchHit() и repair() — взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

- **setName**

```
void setName(String name)
```

  - **Описание**

    Задаёт имя игрока. Ограничения: имя может состоять только из латинских букв, цифр и символов нижнего подчеркивания. Длина имени должна составлять от 1 до 13 символов; все последующие символы будут обрезаны. Жюри оставляет за собой право изменять имена игроков.

  - **Parameters**

    * `name` — имя игрока

- **stepBack**

```
void stepBack()
```

  - **Описание**

    Задаёт желание игрока сделать шаг назад.

    Ограничения: методы stepForward(), stepBack(), sideRight(), stepLeft() и repair() — взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

- **stepForward**

```
void stepForward()
```

  - **Описание**

    Задаёт желание игрока сделать шаг вперёд.

    Ограничения: методы stepForward(), stepBack(), sideRight(), stepLeft() и repair() — взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

- **stepLeft**

```
void stepLeft()
```

- **Описание**

  Задаёт желание игрока сделать шаг влево.

  Ограничения: методы stepForward(), stepBack(), sideRight(), stepLeft() и repair() — взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

- **stepRight**

```
void stepRight()
```

- **Описание**

  Задаёт желание игрока сделать шаг вправо.

  Ограничения: методы stepForward(), stepBack(), sideRight(), stepLeft() и repair() — взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

- **turn**

```
void turn(float angle)
```

- **Описание**

  Задаёт угол поворота игрока (от его текущего направления движения)

- **Параметры**

  * `angle` – `double` значение от $-\pi$ до π.

- **turnTo**

```
void turnTo(float x, float y)
```

- **Описание**

  Задаёт угол поворота игрока. Устанавливает угол поворота игрока как угол между двумя векторами: вектором, направленным из центра игрока вдоль направления его движения, и вектором, направленным из центра игрока в точку с координатами $(x,\ y)$.

- **Параметры**

  * `x` — координата x точки
  * `y` — координата y точки

- **turnTo**

```
void turnTo(Unit unit)
```

– **Описание**

Задаёт угол поворота игрока. Устанавливает угол поворота игрока как угол между двумя векторами: вектором, направленным из центра игрока вдоль направления его движения, и вектором направленным из центра игрока в центр `unit`.

– **Параметры**

* `unit` — unit, в направлении которого хотим повернуться.

• **wrenchHit**

```
void wrenchHit()
```

– **Описание**

Ударяет гаечным ключом (если возможно)

Ограничения: методы wrenchHit() и repair() — взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним.

3.8 Interface Unit

Базовый интерфейс для всех объектов игрового мира. Каждый объект характеризуется позицией и радиусом (все объекты мира, кроме Obstacle, могут считаться кругами с некоторым радиусом). Интерфейс содержит дополнительные методы для расчёта расстояний и углов между объектами.

Определение: Назовём точку, имеющую в исходном положении координаты $(x{+}radius,\ y)$, передней точкой объекта.

3.8.1 Описание

```
public interface Unit
```

3.8.2 Методы

• **getAngle**

```
float getAngle()
```

– **Описание**

Возвращает угол между исходным положением объекта и его текущим положением. Исходное положение объекта задаётся вектором, направленным из центра объекта в исходное положение передней точки. Текущее положение объекта задаётся вектором, направленным из центра объекта в текущее положение передней точки.

– **Возвращает** — угол в радианах от $-\pi$ до π.

• **getAngleTo**

```
float getAngleTo(float x, float y)
```

– **Описание**
  Возвращает угол между вектором, направленным из центра объекта в точку (x,y), и вектором, направленным из центра объекта в переднюю точку объекта.
– **Параметры**
  * `x` — координата x точки
  * `y` — координата y точки
– **Возвращает** – угол в радианах от $-\pi$ до π.

• **getAngleTo**

```
float getAngleTo(Unit obj)
```

– **Описание**
  Возвращает угол между вектором, направленным из центра текущего объекта к центру объекта obj, и вектором, направленным из центра объекта к передней точке объекта.
– **Параметры**
  * `obj` — объект
– **Возвращает** — угол в радианах от $-\pi$ до π

• **getDistanceTo**

```
float getDistanceTo(float x,float y)
```

– **Описание**
  Возвращает расстояние от центра объекта до точки с координатами (x, y)
– **Параметры**
  * `x` — координата x точки
  * `y` — координата y точки
– **Возвращает** — расстояние до точки

• **getDistanceTo**

```
float getDistanceTo(Unit obj)
```

– **Описание**
  Возвращает расстояние между центрами текущего объекта и объекта obj
– **Параметры**
  * `obj` — объект
– **Возвращает** — расстояние до объекта

• **getRadius**

```
float getRadius()
```

- **Описание**
  Возвращает радиус объекта
- **Возвращает** — радиус

- **getX**

```
float getX()
```

  - **Описание**
    Возвращает координату x объекта
  - **Возвращает** — координата x

- **getY**

```
float getY()
```

  - **Описание**
    Возвращает координату y объекта
  - **Возвращает** — координата y

3.9 Interface World

Интерфейс, предоставляющий доступ к свойствам игрового мира. Мир представляет собой прямоугольник размера getWidth() × getHeight()

3.9.1 Описание

```
public interface World
```

3.9.2 Методы

- **getBonuses**

```
List<Bonus> getBonuses()
```

  - **Возвращает** — список бонусов, присутствующих в данный момент в игровом мире

- **getEnemies**

```
List<Enemy> getEnemies()
```

  - **Возвращает** — список противников, присутствующих в игровом мире

- **getHeight**

```
float getHeight()
```

- **Возвращает** – высота прямоугольника, представляющего мир

- **getLeakages**

```
List<Leakage> getLeakages()
```

- **Возвращает** – список протечек, присутствующих в игровом мире

- **getObstacles**

```
List<Obstacle> getObstacles()
```

- **Возвращает** – список препятствий, присутствующих в игровом мире

- **getTick**

```
int getTick()
```

- **Возвращает** – время, прошедшее от создания игрового мира

- **getWidth**

```
float getWidth()
```

- **Возвращает** – ширина прямоугольника, представляющего мир

3.10 Class Action

Типы действий, которые могут быть совершены игроком:
Ударить гаечным ключом

3.10.1 Описание

```
public enum Action
```

3.10.2 Поля

- `public static final Action` **WRENCH_HIT**

3.11 Class BonusType

Типы бонусов в игровом мире:
скорость работы
заморозка кранов

3.11.1 Описание

```
public enum BonusType
```

3.11.2 Поля

- `public static final BonusType` **WORK**
- `public static final BonusType` **FREEZE**

3.12 Class EffectType

Типы эффектов, которые могут быть применены к рабочему:
не способен двигаться
скорость починки кранов увеличена вдвое

3.12.1 Описание

```
public enum EffectType
```

3.12.2 Поля

- `public static final EffectType` **DISABLED**
- `public static final EffectType` **INDUSTRIOUS**

3.13 Class Player

Абстрактный класс игрока. Участники должны создать наследника данного класса и реализовать методы init() и move().

3.13.1 Описание

```
public abstract class Player
```

3.13.2 Методы

- **getSelfControl**

  ```
  public final SelfControl getSelfControl()
  ```

  - **Описание**
    Доступ к игроку
  - **Возвращает** — SelfControl объект, предоставляющий информацию об игроке

- **getWorld**

  ```
  public final World getWorld()
  ```

– **Описание**

  Доступ к информации об игровом мире и его составляющих.

– **Возвращает** — объект World

- **init**

```
public abstract void init()
```

  – **Описание**

    Абстрактный метод инициализации. Метод должен быть переопределён в классе-наследнике. Вызвается перед первым вызовом метода move().

    Ограничения: запрещено вызывать getWorld() (в момент вызова метода init() объект world не существует). Время выполнения ограничено (0.25с)

- **init**

```
public final void init(SelfControl selfControl ,World world)
```

  – **Описание**

    Метод инициализации игрока. Задаёт значения полей selfControl и world, которые необходимы для управления игроком и получении информации об игровом мире. Участники не должны вызывать этот метод.

  – **Параметры**

    * `selfControl` — объект, предоставляющий информацию об игроке
    * `world` — объект, предоставляющий информацию об игровом мире

- **move**

```
public abstract void move()
```

  – **Описание**

    Абстрактный метод перемещения. Метод должен быть переопределён в классе-наследнике. Вызывается, чтобы сделать следующий ход. Чтобы управлять игроком в этом методе, используйте объект selfControl. Чтобы получить информацию об игровом мире, используйте объект world.

    Ограничения: Время выполнения ограничено (0.25с)

3.14 Class PlayerType

Возможные цвета игроков

3.14.1 Описание

public enum PlayerType

3.14.2 Поля

- `public static final PlayerType` **RED**
- `public static final PlayerType` **BLUE**
- `public static final PlayerType` **GREEN**
- `public static final PlayerType` **YELLOW**

3.15 Class ShellType

Тип ущерба, который игрок может нанести противнику или починка протечки:
удар ключом
починка протечки

3.15.1 Описание

public enum ShellType

3.15.2 Поля

- `public static final ShellType` **WRENCH_HIT**
- `public static final ShellType` **REPAIR**