Code‘N‘Go: Code Troopers

Самара, 2017

Оглавление

Глава 1

Code Trooper

1.1 Обзор

Code‘N‘Go даёт вам возможность проверить свои способности в программировании против других команд в мире Code Trooper. Каждая команда пишет свою игровую стратегию для штурмовика. Ваш штурмовик будет помещен в смоделированный мир вместе со штурмовиками других игроков.

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

Штурмовики могут перемещаться в двумерном прямоугольном мире. Они могут собирать бонусы, стрелять из бластеров, устанавливать и обезвреживать бомбы. Штурмовики зарабатывают очки за уменьшение здоровья других игроков. Боезапас штурмовика ограничен и может быть пополнен на зарядной станции. Также на зарядной станции пополняется здоровье игрока. На поле существует не более 4х станции - по одной для каждого из игроков. Штурмовик может взаимодействовать только со станцией своего цвета. Задача игрока — набрать как можно больше очков.

В игровом мире есть бонус «Щит». Подобравший этот бонус штурмовик перестает получать урон. Также есть бонус «Бомба». Этот бонус дает штурмовику возможность установить бомбу.

Если в штурмовика попадает снаряд то он теряет здоровье. Также он может потерять здоровье если окажется недалеко от бомбы во время взрыва. Если у штурмовика заканчивается здоровье, то он выходит из строя на некоторое время. По истечении этого времени штурмовик получает небольшой запас здоровья, а также возможность двигаться и совершать действия.

Главная цель игры — набрать как можно больше очков в течение матча. Очки набираются в момент, когда снаряд или бомба штурмовика наносит урон другому игроку.

Часть программирования Code‘N‘Go начинается, когда вы получаете доступ к соревнованию. Вы можете использовать только один язык программирования, поддерживаемый платформой, — Java. Вам также будет предоставлено специальное клиентское приложение Code‘N‘Go, которое описано в соответствующей главе данного руководства.

У вас есть возможность отправлять в тестирующую систему класс вашего игрока в течение фазы программирования Code‘N‘Go. Финальные версии классов всех игроков (последняя успешно скомпилированная посылка, которую вы сделали) будут участвовать в турнире Code Show. Все игроки будут состязаться как минимум в двух матчах. Победителем Code‘N‘Go 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 последней главе 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 или 3 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.

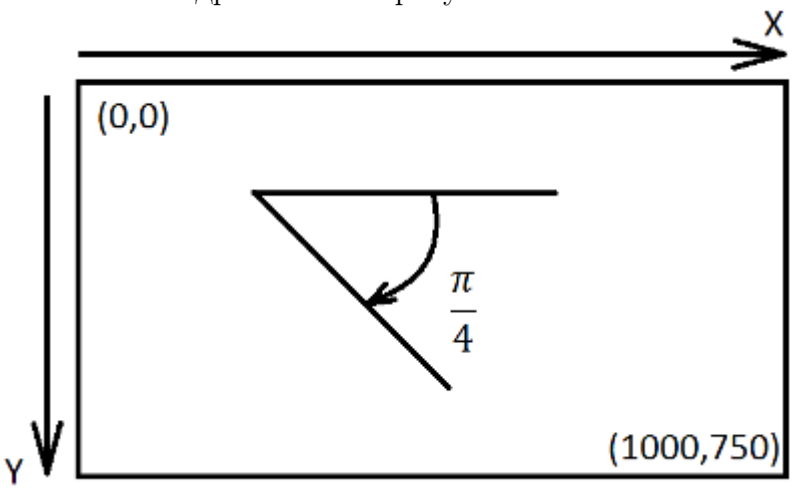
Второй отсев произойдет после полуфинала. Игроки будут отсортированы по общему количеству очков, и первые 4 пройдут в финал.

После каждого блока набранные очки обнуляются.

Игра происходит следующим образом. Сначала 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. В момент 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): штурмовик, бонус, станция подзарядки, бомба, снаряд, препятствие. Каждый юнит, за исключением препятствия, — это круг с определенным радиусом. Препятствие — это квадрат, в его случае за радиус принимаем радиус вписанной окружности. Радиусы юнитов представлены в следующей таблице:

Unit	R
Штурмовик	30
Бонус	20
Станция зарядки	40
Бомба	20
Снаряд	5
Препятствие	15

Препятствия объединяются в блоки, образуя карту мира. Для каждой игры карта выбирается случайным образом. У препятствия есть индикатор урона и оно может быть полностью разрушено.

Каждый штурмовик начинает матч в заданном для его цвета месте.

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

Unit	**Результат взаимодействия с активным штурмовиком**
Штурмовик	Скорости и направления штурмовиков меняются
Бонус «Щит»	В течении 300 тиков штурмовик не получает никакого урона
Бонус «Бомба»	Штурмовик получает возможность установить бомбу. Если на момент взаимодействия у штурмовика была бомба, она автоматически устанавливается
Станция зарядки	Запас здоровья и боеприпасов штурмовика пополняется
Активированная бомба	У штурмовика появляется возможность обезвредить бомбу
Снаряд	Снаряд исчезает из мира. Здоровье штурмовика уменьшается на 10 единиц.
Препятствие	Скорость и направление штурмовика изменяются

Когда штурмовик 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 штурмовиков одинакова.

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

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

Характеристики штурмовика 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

Штурмовик может совершать следующие действия:

- Изменять своё направление и скорость
- Собирать бонусы
- Стрелять из бластера
- Устанавливать бомбы
- Деактивировать бомбы

При возможности штурмовик может в любом месте мира установить бомбу. Бомба устанавливается перед штурмовиком на расстоянии 5 пунктов. Сразу после установки начинается обратный отсчет 300 тиков. По истечении таймера бомба взрывается. При взрыве у всех штурмовиков в радиусе 200 точек уменьшается здоровье. Количество единиц здоровья, потерянных штурмовиком, варьируется от 100 до 0 в зависимости от расстояния до бомбы на момент взрыва. Штурмовик установивший бомбу получает по 1 очку за каждую единицу здоровья снятую с другого игрока.

До истечения таймера любой штурмовик может деактивировать бомбу. Возможность деактивации наступает если расстояние между штурмовиком и бомбой меньше чем *(радиус штурмовика + радиус бомбы + 10)* и угол между штурмовиком и бомбой меньше или равен $|\frac{\pi}{2}|$. Для деактивации бомбы с неё необходимо снять 3 единицы. За один ход одним штурмовиком снимается одна единица. В следующий раз штурмовик сможет инициировать деактивацию бомбы не ранее, чем через 40 тиков.

Штурмовик может совершить выстрел из бластера. В этом случае перед штурмовиком на расстоянии 5 пунктов возникает снаряд. Снаряду придается скорость 10 точек/тик. При попадании снаряда в штурмовика, у него отнимается 10 единиц здоровья. Игроку выпустившему снаряд прибавляется столько очков, сколько единиц здоровья потерял другой игрок. Стрелять из бластера штурмовик может не чаще чем раз в 40 тиков. Боезапас штурмовика ограничен и составляет 25 зарядов.

У каждого препятствия есть стойкость и она составляет 30 единиц. Если в препятствие попал снаряд, то у него отнимается 2 единицы стойкости. Если в радиусе 200 точек от препятствия взорвалась бомба, то у препятствия также отнимаются единицы стойкости. Количество единиц стойкости снимаемое с препятствия при взрыве бомбы варьируется от 30 до 0 в зависимости от расстояния до бомбы на момент взрыва. Если стойкость препятствия падает до 0, то препятствие считается уничтоженным и исчезает с поля.

Если здоровье штурмовика падает до 0 то штурмовик становится неактивным на 300 тиков. Неактивный штурмовик не может двигаться, поворачивать или совершать другие действия. После выхода из неактивного состояния у штурмовика нулевой боезапас и небольшой запас здоровья (50 единиц).

Если штурмовик приближается к своей станции подзарядки то его запас здоровья и боеприпасов начинает постепенно пополняться, пока не дойдет до максимальной отметки.

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 (`trooper.jar`, `trooper-javadoc.jar`) находятся в каждой из директорий `EclipseProject`, `IdeaProject`, `NetBeansProject` в поддиректории `lib`.

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 штурмовик 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 штурмовика (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 штурмовик 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)

```
import trooper.interfaces.Player;
import trooper.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();
        }
}
```

2.5 Клиент Code‘N‘Go

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`).

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'N'Go.

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*)

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

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

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‘N‘Go 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 - Установить бомбу
- P - Обезвредить бомбу
- SPACE - Выстрелить из бластера

Глава 3

JavaDoc

3.1 Interface Bomb

Интерфейс, предоставляющий информацию о бомбах

3.1.1 Описание

```
public interface Bomb
 extends Unit
```

3.1.2 Методы

- **getRemainingDefusePoints**

  ```
  int getRemainingDefusePoints()
  ```

  - **Возвращает** – количество поинтов до обезвреживания бомбы

- **getRemainingTime**

  ```
  int getRemainingTime()
  ```

  - **Возвращает** – количество времени до взрыва бомбы

3.2 Interface Bonus

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

3.2.1 Описание

```
public interface Bonus
 extends Unit
```

3.2.2 Методы

- **getBonusType**

```
BonusType getBonusType()
```

  - **Описание**
    Метод возвращает тип бонуса. Существует два типа бонусов:
    * SHIELD - Даёт игроку щит.
    * BOMB - Даёт игроку бомбу.
  - **Возвращает** – тип бонуса

3.3 Interface Charger

Интерфейс, предоставляющий информацию о зарядных станциях

3.3.1 Описание

```
public interface Charger
 extends Unit
```

3.3.2 Методы

- **getType**

```
PlayerType getType()
```

3.4 Interface Effect

Интерфейс описывает эффекты, демонстрирующие состояние игрока.

3.4.1 Описание

```
public interface Effect
```

3.4.2 Методы

- **getRemainingDuration**

```
int getRemainingDuration()
```

  - **Возвращает** – время действия в единицах игрового времени

- **getType**

```
EffectType getType()
```

– **Описание**
Возвращает тип эффекта. Существует 2 вида эффектов:
эффект PROTECTED - Игрок находится под щитом
эффект DISABLED - Игрок не может совершать никаких действий
– **Возвращает** – тип эффекта.

3.5 Interface Enemy

Интерфейс, предоставляющий доступ к информации о противнике.

3.5.1 Описание

```
public interface Enemy
 extends MovableUnit
```

3.5.2 Методы

- **getEffects**

```
java.util.List getEffects()
```

  – **Возвращает** – список эффектов, применённых к противнику

- **getName**

```
java.lang.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.
  – **Parameters**
    * `effectType` – проверяемый эффект
  – **Возвращает** – true если эффект применяется в настоящее время к противнику, иначе false.

3.6 Interface MovableUnit

Интерфейс описывающий движущийся объект

3.6.1 Описание

```
public interface MovableUnit
 extends Unit
```

3.6.2 Методы

- **getHorizontalSpeed**

  ```
  float getHorizontalSpeed()
  ```

  - **Описание**
    Возвращает горизонтальную составляющую скорости
  - **Возвращает** – проекцию вектора скорости на ось Ox

- **getSpeed**

  ```
  float getSpeed()
  ```

  - **Описание**
    Возвращает абсолютную величину скорости объекта
  - **Возвращает** – длину 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** – проекцию вектора скорости на ось Oy

3.7 Interface Obstacle

Интерфейс, описывающий препятствие.

3.7.1 Описание

```
public interface Obstacle
 extends Unit
```

3.7.2 Методы

- **getRemainingStrength**

  ```
  int getRemainingStrength()
  ```

  - **Возвращает** – количество пунктов до уничтожения препятствия

3.8 Interface SelfControl

Интерфейс, позволяющий получить информацию об игроке и управлять им. Назначение: в момент инициализации можно задать имя игрока, изменить направление и скорость движения игрока,а также дать команды: стрелять из оружия, установить бомбу, обезвредить бомбу

3.8.1 Описание

```
public interface SelfControl
 extends Enemy
```

3.8.2 Методы

- **defuseBomb**

  ```
  void defuseBomb()
  ```

  - **Описание**
    Обезвреживает бомбу
    Ограничения: методы shoot(), setBomb() и defuseBomb() взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним

- **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
  - **Parameters**
    * `action` – действие
  - **Возвращает** – количество единиц игрового времени

- **getScore**

  ```
  int getScore()
  ```

  - **Возвращает** – количество очков, набранное игроком на момент запроса

- **hasBomb**

  ```
  boolean hasBomb()
  ```

  - **Описание**
    Проверяет, имеется ли у игрока бомба
  - **Возвращает** – true при наличии бомбы, иначе false

- **setBomb**

  ```
  void setBomb()
  ```

  - **Описание**
    Устанавливает бомбу
    Ограничения: методы shoot(), setBomb() и defuseBomb() взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним

- **setName**

  ```
  void setName(java.lang.String name)
  ```

  - **Описание**
    Задаёт имя игрока.
    Ограничения: Имя может состоять только из латинских букв, цифр и нижних подчеркиваний. 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

- **shoot**

  ```
  void shoot()
  ```

  - **Описание**
    Стреляет из ружья (если возможно)
    Ограничения: методы shoot(), setBomb() и defuseBomb() взаимоисключающие. При вызове этих методов применяться будет тот, который вызван последним

- **stepBack**

  ```
  void stepBack()
  ```

  - **Описание**

    Задаёт желание игрока сделать шаг назад.

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

- **stepForward**

  ```
  void stepForward()
  ```

  - **Описание**

    Задаёт желание игрока сделать шаг вперед.

    Ограничения: методы stepForward(), stepBack(), sideRight(), stepLeft() 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() 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() 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)

- **Parameters**
  - `angle` – `double` значение от -π 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)
  - **Parameters**
    - `x` - x координата точки
    - `y` - y координата 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`.
  - **Parameters**
    - `unit` – unit в направлении которого хотим повернуться.

3.9 Interface Shell

Интерфейс, описывающий снаряд. Наследник MovableUnit

3.9.1 Описание

```
public interface Shell
 extends MovableUnit
```

3.9.2 Методы

- **getType**

```
ShellType getType()
```

  - **Описание**
    Метод, возвращающий тип снаряда - пуля (Bullet)
  - **Возвращает** – тип снаряда

3.10 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.

3.10.1 Описание

```
public interface Unit
```

3.10.2 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 -π 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 к передней точке объекта.
  - **Parameters**
    - `x` – - координата x точки
    - `y` – - координата y точки
  - **Возвращает** – угол в радианах от -π 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.

- **Parameters**
  - `obj` – - объект
- **Возвращает** – угол в радианах от -π до π

- **getDistanceTo**

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

- **Описание**
  Возвращает расстояние от центра объекта до точки с координатами (x,y)
- **Parameters**
  - `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
- **Parameters**
  - `obj` – объект
- **Возвращает** – расстояние до объекта

- **getRadius**

```
float getRadius()
```

- **Описание**
  Возвращает радиус объекта
- **Возвращает** – радиус

- **getX**

```
float getX()
```

- **Описание**
  Возвращает координату x объекта
- **Возвращает** – координату x

- **getY**

```
float getY()
```

- **Описание**
  Возвращает координату y объекта
- **Возвращает** – координату y

3.11 Interface World

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

3.11.1 Описание

```
public interface World
```

3.11.2 Методы

- **getBombs**

  ```
  java.util.List getBombs()
  ```

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

- **getBonuses**

  ```
  java.util.List getBonuses()
  ```

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

- **getChargers**

  ```
  java.util.List getChargers()
  ```

  - **Возвращает** – список зарядных устройств, присутствующих в игровом мире

- **getEnemies**

  ```
  java.util.List getEnemies()
  ```

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

- **getHeight**

  ```
  float getHeight()
  ```

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

- **getObstacles**

  ```
  java.util.List getObstacles()
  ```

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

- **getShells**

  ```
  java.util.List getShells()
  ```

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

- **getTick**

  ```
  int getTick()
  ```

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

- **getWidth**

  ```
  float getWidth()
  ```

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

3.12 Class Action

Типы действий, которые могут быть совершены игроком: Выстрелить из оружия, Установить бомбу, Обезвредить бомбу

3.12.1 Описание

```
public final class Action
 extends java.lang.Enum
```

3.12.2 Поля

- `public static final Action` **SHOOT**
- `public static final Action` **SET_BOMB**
- `public static final Action` **DEFUSE_BOMB**

3.12.3 Методы

- **valueOf**

  ```
  public static Action valueOf(java.lang.String name)
  ```

- **values**

  ```
  public static Action[] values()
  ```

3.13 Class BonusType

Типы бонусов в игровом мире: Щит, Бомба

3.13.1 Описание

```
public final class BonusType
 extends java.lang.Enum
```

3.13.2 Поля

- `public static final BonusType` **SHIELD**
- `public static final BonusType` **BOMB**

3.13.3 Методы

- **valueOf**

  ```
  public static BonusType valueOf(java.lang.String name)
  ```

- **values**

  ```
  public static BonusType[] values()
  ```

3.14 Class EffectType

Типы эффектов которые могут быть применены к рабочему: Находится под щитом, Недееспособен

3.14.1 Описание

```
public final class EffectType
 extends java.lang.Enum
```

3.14.2 Поля

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

3.14.3 Методы

- **valueOf**

  ```
  public static EffectType valueOf(java.lang.String name)
  ```

- **values**

  ```
  public static EffectType[] values()
  ```

3.15 Class Player

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

3.15.1 Описание

```
public abstract class Player
 extends FrameworkPlayer
```

3.15.2 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() throws java.util.concurrent.
     TimeoutException
  ```

  - **Описание**
    Абстрактный метод инициализации. Метод должен быть переопределен 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 (0.25c)

- **init**

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

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

– **Parameters**

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

- **move**

```
public abstract void move() throws java.util.concurrent.
    TimeoutException
```

– **Описание**

Абстрактный метод перемещения. Метод должен быть переопределен 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)

3.16 Class PlayerType

Возможные цвета игроков

3.16.1 Описание

```
public final class PlayerType
 extends java.lang.Enum
```

3.16.2 Поля

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

3.16.3 Методы

- **valueOf**

```
public static PlayerType valueOf(java.lang.String name)
```

- **values**

```
public static PlayerType[] values()
```

3.17 Class ShellType

Тип снаряда: пуля

3.17.1 Описание

```
public final class ShellType
 extends java.lang.Enum
```

3.17.2 Поля

- `public static final ShellType` **BULLET**

3.17.3 Методы

- **valueOf**

  ```
  public static ShellType valueOf(java.lang.String name)
  ```

- **values**

  ```
  public static ShellType[] values()
  ```